Computability and Logic

Fifth Edition

GEORGE S. BOOLOS

JOHN P. BURGESS
Princeton University

RICHARD C. JEFFREY

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press
The Edinburgh Building, Cambridge CB2 8RU, UK
Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9780521877527

First published in print format 2007

ISBN-13	978-0-511-36668-0	eBook (EBL)
ISBN-10	0-511-36668-X	eBook (EBL)
ISBN-13	978-0-521-87752-7	hardback
ISBN-10	0-521-87752-0	hardback
ISBN-13	978-0-521-70146-4	paperback
ISBN-10	0-521-70146-5	paperback

Cambridge University Press has no responsibility for the persistence or accuracy of urls for external or third-party internet websites referred to in this publication, and does not guarantee that any content on such websites is, or will remain, accurate or appropriate.

Contents

COMPUTABILITY THEORY

BASIC METALOGIC

FURTHER TOPICS

Preface to the Fifth Edition

The original authors of this work, the late George Boolos and Richard Jeffrey, stated in the preface to the first edition that the work was intended for students of philosophy, mathematics, or other fields who desired a more advanced knowledge of logic than is supplied by an introductory course or textbook on the subject, and added the following:

> The aim has been to present the principal fundamental theoretical results *about* logic, and to cover certain other meta-logical results whose proofs are not easily obtainable elsewhere. We have tried to make the exposition as readable as was compatible with the presentation of complete proofs, to use the most elegant proofs we knew of, to employ standard notation, and to reduce *hair* (as it is technically known).

Such have remained the aims of all subsequent editions.

The "principal fundamental theoretical results *about* logic" are primarily the theorems of Gödel, the completeness theorem, and especially the incompleteness theorems, with their attendant lemmas and corollaries. The "other meta-logical results" included have been of two kinds. On the one hand, filling roughly the first third of the book, there is an extended exposition by Richard Jeffrey of the theory of Turing machines, a topic frequently alluded to in the literature of philosophy, computer science, and cognitive studies but often omitted in textbooks on the level of this one. On the other hand, there is a varied selection of theorems on (in-)definability, (un-)decidability, (in-)completeness, and related topics, to which George Boolos added a few more items with each successive edition, until by the third, the last to which he directly contributed, it came to fill about the last third of the book.

When I undertook a revised edition, my special aim was to increase the pedagogical usefulness of the book by adding a selection of problems at the end of each chapter and by making more chapters independent of one another, so as to increase the range of options available to the instructor or reader as to what to cover and what to defer. Pursuit of the latter aim involved substantial rewriting, especially in the middle third of the book. A number of the new problems and one new section on undecidability were taken from Boolos's *Nachlass*, while the rewriting of the précis of first-order logic – summarizing the material typically covered in a more leisurely way in an introductory text or course and introducing the more abstract modes of reasoning that distinguish intermediate- from introductory-level logic – was undertaken in consultation with Jeffrey. Otherwise, the changes have been my responsibility alone.

The book runs now in outline as follows. The basic course in intermediate logic culminating in the first incompleteness theorem is contained in Chapters 1, 2, 6, 7, 9, 10, 12, 15, 16, and 17, minus any sections of these chapters starred as optional. Necessary background on enumerable and nonenumerable sets is supplied in Chapters 1 and 2. All the material on computability (recursion theory) that is strictly needed for the incompleteness theorems has now been collected in Chapters 6 and 7, which may, if desired, be postponed until after the needed background material in logic. That material is presented in Chapters 9, 10, and 12 (for readers who have not had an introductory course in logic including a proof of the completeness theorem, Chapters 13 and 14 will also be needed). The machinery needed for the proof of the incompleteness theorems is contained in Chapter 15 on the arithmetization of syntax (though the instructor or reader willing to rely on Church's thesis may omit all but the first section of this chapter) and in Chapter 16 on the representability of recursive functions. The first completeness theorem itself is proved in Chapter 17. (The second incompleteness theorem is discussed in Chapter 18.)

A semester course should allow time to take up several supplementary topics in addition to this core material. The topic given the fullest exposition is the theory of Turing machines and their relation to recursive functions, which is treated in Chapters 3 through 5 and 8 (with an application to logic in Chapter 11). This now includes an account of Turing's theorem on the existence of a universal Turing machine, one of the intellectual landmarks of the last century. If this material is to be included, Chapters 3 through 8 would best be taken in that order, either after Chapter 2 or after Chapter 12 (or 14).

Chapters 19 through 21 deal with topics in general logic, and any or all of them might be taken up as early as immediately after Chapter 12 (or 14). Chapter 19 is presupposed by Chapters 20 and 21, but the latter are independent of each other. Chapters 22 through 26, all independent of one another, deal with topics related to formal arithmetic, and any of them could most naturally be taken up after Chapter 17. Only Chapter 27 presupposes Chapter 18. Users of the previous edition of this work will find essentially all the material in it still here, though not always in the same place, apart from some material in the former version of Chapter 27 that has, since the last edition of this book, gone into *The Logic of Provablity*.

All these changes were made in the fourth edition. In the present fifth edition, the main change to the body of the text (apart from correction of errata) is a further revision and simplification of the treatment of the representability of recursive functions, traditionally one of the greatest difficulties for students. The version now to be found in section 16.2 represents the distillation of more than twenty years' teaching experience trying to find ever easier ways over this hump. Section 16.4 on Robinson arithmetic has also been rewritten. In response to a suggestion from Warren Goldfarb, an explicit discussion of the distinction between two different kinds of appeal to Church's thesis, avoidable and unavoidable, has been inserted at the end of section 7.2. The avoidable appeals are those that consist of omitting the verification that certain obviously effectively computable functions are recursive; the unavoidable appeals are those involved whenever a theorem about recursiveness is converted into a conclusion about effective computability in the intuitive sense.

On the one hand, it should go without saying that in a textbook on a classical subject, only a small number of the results presented will be original to the authors. On the other hand, a textbook is perhaps not the best place to go into the minutiæ of the history of a field. Apart from a section of remarks at the end of Chapter 18, we have indicated the history of the field for the student or reader mainly by the names attached to various theorems. See also the annotated bibliography at the end of the book.

There remains the pleasant task of expressing gratitude to those (beyond the dedicatees) to whom the authors have owed personal debts. By the third edition of this work the original authors already cited Paul Benacerraf, Burton Dreben, Hartry Field, Clark Glymour, Warren Goldfarb, Simon Kochen, Paul Kripke, David Lewis, Paul Mellema, Hilary Putnam, W. V. Quine, T. M. Scanlon, James Thomson, and Peter Tovey, with special thanks to Michael J. Pendlebury for drawing the "mop-up" diagram in what is now section 5.2.

In connection with the fourth edition, my thanks were due collectively to the students who served as a trial audience for intermediate drafts, and especially to my very able assistants in instruction, Mike Fara, Nick Smith, and Caspar Hare, with special thanks to the last-named for the "scoring function" example in section 4.2. In connection with the present fifth edition, Curtis Brown, Mark Budolfson, John Corcoran, Sinan Dogramaci, Hannes Eder, Warren Goldfarb, Hannes Hutzelmeyer, David Keyt, Brad Monton, Jacob Rosen, Jada Strabbing, Dustin Tucker, Joel Velasco, Evan Williams, and Richard Zach are to be thanked for errata to the fourth edition, as well as for other helpful suggestions.

Perhaps the most important change connected with this fifth edition is one not visible in the book itself: It now comes supported by an instructor's manual. The manual contains (besides any errata that may come to light) suggested hints to students for odd-numbered problems and solutions to all problems. Resources are available to students and instructors at www.cambridge.org/us/9780521877527.

January 2007 JOHN P. BURGESS

1

Enumerability

Our ultimate goal will be to present some celebrated theorems about inherent limits on what can be computed and on what can be proved. Before such results can be established, we need to undertake an analysis of computability and an analysis of provability. Computations involve positive integers 1, 2, 3, . . . in the first instance, while proofs consist of sequences of symbols from the usual alphabet A, B, C, *. . . or some other. It will turn out to be important for the analysis both of computability and of provability to understand the relationship between positive integers and sequences of symbols, and background on that relationship is provided in the present chapter. The main topic is a distinction between two different kinds of infinite sets, the enumerable and the nonenumerable. This material is just a part of a larger theory of the infinite developed in works on set theory: the part most relevant to computation and proof. In section 1.1 we introduce the concept of enumerability. In section 1.2 we illustrate it by examples of enumerable sets. In the next chapter we give examples of nonenumerable sets.*

1.1 Enumerability

An *enumerable*, or *countable*, set is one whose members can be enumerated: arranged in a single list with a first entry, a second entry, and so on, so that every member of the set appears sooner or later on the list. Examples: the set P of positive integers is enumerated by the list

$$1, 2, 3, 4, \ldots$$

and the set N of natural numbers is enumerated by the list

$$0, 1, 2, 3, \ldots$$

while the set P^- of negative integers is enumerated by the list

$$-1, -2, -3, -4, \ldots.$$

Note that the entries in these lists are not numbers but numerals, or names of numbers. In general, in listing the members of a set you manipulate names, not the things named. For instance, in enumerating the members of the United States Senate, you don't have the senators form a queue; rather, you arrange their *names* in a list, perhaps alphabetically. (An arguable exception occurs in the case where the members of the set being enumerated are themselves linguistic expressions. In this case we can plausibly speak of arranging the members themselves in a list. But we might also speak of the entries in the list as *names of themselves* so as to be able to continue to insist that in enumerating a set, it is *names* of members of the set that are arranged in a list.)

By courtesy, we regard as enumerable the empty set, ∅, which has no members. (*The* empty set; there is only one. The terminology is a bit misleading: It suggests comparison of empty sets with empty containers. But sets are more aptly compared with contents, and it should be considered that all empty containers have the same, null content.)

A list that enumerates a set may be finite or unending. An infinite set that is enumerable is said to be *enumerably infinite* or *denumerable*. Let us get clear about what things count as infinite lists, and what things do not. The positive integers can be arranged in a single infinite list as indicated above, but the following is not acceptable as a list of the positive integers:

$$1, 3, 5, 7, \ldots, 2, 4, 6, \ldots$$

Here, all the odd positive integers are listed, and then all the even ones. This will not do. In an acceptable list, each item must appear sooner or later as the nth entry, for some *finite n*. But in the unacceptable arrangement above, none of the even positive integers are represented in this way. Rather, they appear (so to speak) as entries number $\infty + 1$, $\infty + 2$, and so on.

To make this point perfectly clear we might define an enumeration of a set not as a listing, but as an arrangement in which each member of the set is *associated with* one of the positive integers 1, 2, 3, ... Actually, a list *is* such an arrangement. The thing named by the first entry in the list is associated with the positive integer 1, the thing named by the second entry is associated with the positive integer 2, and in general, the thing named by the nth entry is associated with the positive integer n.

In mathematical parlance, an infinite list determines a *function* (call it f) that takes positive integers as *arguments* and takes members of the set as *values*. [Should we have written: 'call it "f",' rather than 'call it f'? The common practice in mathematical writing is to use special symbols, including even italicized letters of the ordinary alphabet when being used as special symbols, as names for themselves. In case the special symbol happens also to be a name for something else, for instance, a function (as in the present case), we have to rely on context to determine when the symbol is being used one way and when the other. In practice this presents no difficulties.] The value of the function f for the argument n is denoted $f(n)$. This value is simply the thing denoted by the nth entry in the list. Thus the list

$$2, 4, 6, 8, \ldots$$

which enumerates the set E of even positive integers determines the function f for which we have

$$f(1) = 2, \qquad f(2) = 4, \qquad f(3) = 6, \qquad f(4) = 8, \qquad f(5) = 10, \ldots.$$

And conversely, the function f determines the list, except for notation. (The same list would look like this, in Roman numerals: II, IV, VI, VIII, X, ..., for instance.) Thus, we might have defined the function f first, by saying that for any positive integer n, the value of f is $f(n) = 2n$; and then we could have described the list by saying that for each positive integer n, its nth entry is the decimal representation of the number $f(n)$, that is, of the number $2n$.

Then we may speak of sets as being enumerated by functions, as well as by lists. Instead of enumerating the odd positive integers by the list 1, 3, 5, 7, ..., we may enumerate them by the function that assigns to each positive integer n the value $2n - 1$. And instead of enumerating the set P of all positive integers by the list 1, 2, 3, 4, ..., we may enumerate P by the function that assigns to each positive integer n the value n itself. This is the *identity function*. If we call it id, we have $\text{id}(n) = n$ for each positive integer n.

If one function enumerates a nonempty set, so does some other; and so, in fact, do infinitely many others. Thus the set of positive integers is enumerated not only by the function id, but also by the function (call it g) determined by the following list:

$$2, 1, 4, 3, 6, 5, \ldots.$$

This list is obtained from the list 1, 2, 3, 4, 5, 6, ... by interchanging entries in pairs: 1 with 2, 3 with 4, 5 with 6, and so on. This list is a strange but perfectly acceptable enumeration of the set P: every positive integer shows up in it, sooner or later. The corresponding function, g, can be defined as follows:

$$g(n) = \begin{cases} n + 1 & \text{if } n \text{ is odd} \\ n - 1 & \text{if } n \text{ is even.} \end{cases}$$

This definition is not as neat as the definitions $f(n) = 2n$ and $\text{id}(n) = n$ of the functions f and id, but it does the job: It does indeed associate one and only one member of P with each positive integer n. And the function g so defined does indeed enumerate P: For each member m of P there is a positive integer n for which we have $g(n) = m$.

In enumerating a set by listing its members, it is perfectly all right if a member of the set shows up more than once on the list. The requirement is rather that each member show up *at least once*. It does not matter if the list is redundant: All we require is that it be complete. Indeed, a redundant list can always be thinned out to get an irredundant list, since one could go through and erase the entries that repeat earlier entries. It is also perfectly all right if a list has gaps in it, since one could go through and close up the gaps. The requirement is that every element of the set being enumerated be associated with some positive integer, not that every positive integer have an element of the set associated with it. Thus flawless enumerations of the positive integers are given by the following repetitive list:

$$1, 1, 2, 2, 3, 3, 4, 4, \ldots$$

and by the following gappy list:

$$1, -, 2, -, 3, -, 4, -, \ldots.$$

The function corresponding to this last list (call it h) assigns values corresponding to the first, third, fifth, ... entries, but assigns no values corresponding to the gaps (second, fourth, sixth, ... entries). Thus we have $h(1) = 1$, but $h(2)$ is nothing at all, for the function h is *undefined* for the argument 2; $h(3) = 2$, but $h(4)$ is undefined; $h(5) = 3$, but $h(6)$ is undefined. And so on: h is a *partial function* of positive integers; that is, it is defined only for positive integer arguments, but not for all such arguments. Explicitly, we might define the partial function h as follows:

$$h(n) = (n + 1)/2 \quad \text{if } n \text{ is odd.}$$

Or, to make it clear we haven't simply forgotten to say what values h assigns to even positive integers, we might put the definition as follows:

$$h(n) = \begin{cases} (n + 1)/2 & \text{if } n \text{ is odd} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Now the partial function h is a strange but perfectly acceptable enumeration of the set P of positive integers.

It would be perverse to choose h instead of the simple function id as an enumeration of P; but other sets are most naturally enumerated by partial functions. Thus, the set E of even integers is conveniently enumerated by the partial function (call it j) that agrees with id for even arguments, and is undefined for odd arguments:

$$j(n) = \begin{cases} n & \text{if } n \text{ is even} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The corresponding gappy list (in decimal notation) is

$$-, 2, -, 4, -, 6, -, 8, \ldots.$$

Of course the function f considered earlier, defined by $f(n) = 2n$ for all positive integers n, was an equally acceptable enumeration of E, corresponding to the gapless list 2, 4, 6, 8, and so on.

Any set S of positive integers is enumerated quite simply by a partial function s, which is defined as follows:

$$s(n) = \begin{cases} n & \text{if } n \text{ is in the set } S \\ \text{undefined} & \text{otherwise.} \end{cases}$$

It will be seen in the next chapter that although every set of positive integers is enumerable, there are sets of others sorts that are not enumerable. To say that a set A is enumerable is to say that there is a function all of whose arguments are positive integers and all of whose values are members of A, and that each member of A is a value of this function: For each member a of A there is at least one positive integer n to which the function assigns a as its value.

Notice that nothing in this definition requires A to be a set of positive integers or of numbers of any sort. Instead, A might be a set of people; or a set of linguistic expressions; or a set of sets, as when A is the set $\{P, E, \varnothing\}$. Here A is a set with three members, each of which is itself a set. One member of A is the infinite set P of all positive integers; another member of A is the infinite set E of all even positive integers; and the third is the empty set $\varnothing$. The set A is certainly enumerable, for example, by the following finite list:P, E, $\varnothing$. Each entry in this list names a member of A, and every member of A is named sooner or later on this list. This list determines a function (call it f), which can be defined by the three statements: $f(1) = P$, $f(2) = E$, $f(3) = \varnothing$. To be precise, f is a *partial function* of positive integers, being undefined for arguments greater than 3.

In conclusion, let us straighten out our terminology. A *function* is an assignment of *values* to *arguments*. The set of all those arguments to which the function assigns values is called the *domain* of the function. The set of all those values that the function assigns to its arguments is called the *range* of the function. In the case of functions whose arguments are positive integers, we distinguish between *total* functions and *partial* functions. A total function of positive integers is one whose domain is the whole set P of positive integers. A partial function of positive integers is one whose domain is something less than the whole set P. From now on, when we speak simply of a *function of positive integers*, we should be understood as leaving it open whether the function is total or partial. (This is a departure from the usual terminology, in which *function* of positive integers always means *total function*.) A set is *enumerable* if and only if it is the range of some function of positive integers. We said earlier we wanted to count the empty set $\varnothing$ as enumerable. We therefore have to count as a partial function the *empty function e* of positive integers that is undefined for all arguments. Its domain and its range are both $\varnothing$.

It will also be important to consider functions with two, three, or more positive integers as arguments, notably the addition function $\mathrm{sum}(m, n) = m + n$ and the multiplication function $\mathrm{prod}(m, n) = m \cdot n$. It is often convenient to think of a two-argument or two-place function on positive integers as a one-argument function on *ordered pairs* of positive integers, and similarly for many-argument functions. A few more notions pertaining to functions are defined in the first few problems at the end of this chapter. In general, *the problems at the end should be read as part of each chapter*, even if not all are going to be worked.

1.2 Enumerable Sets

We next illustrate the definition of the preceding section by some important examples. The following sets are enumerable.

1.1 Example (The set of integers). The simplest list is $0, 1, -1, 2, -2, 3, -3, \ldots$. Then if the corresponding function is called f, we have $f(1) = 0$, $f(2) = 1$, $f(3) = -1$, $f(4) = 2$, $f(5) = -2$, and so on.

1.2 Example (The set of ordered pairs of positive integers). The enumeration of pairs will be important enough in our later work that it may be well to indicate two different ways of accomplishing it. The first way is this. As a preliminary to enumerating them, we organize them into a rectangular array. We then traverse the array in *Cantor's zig-zag* manner indicated in Figure 1.1. This gives us the list

$$(1, 1), (1, 2), (2, 1), (1, 3), (2, 2), (3, 1), (1, 4), (2, 3), (3, 2), (4, 1), \ldots.$$

If we call the function involved here G, then we have $G(1) = (1, 1)$, $G(2) = (1, 2)$, $G(3) = (2, 1)$, and so on. The pattern is: First comes the pair the sum of whose entries is 2, then

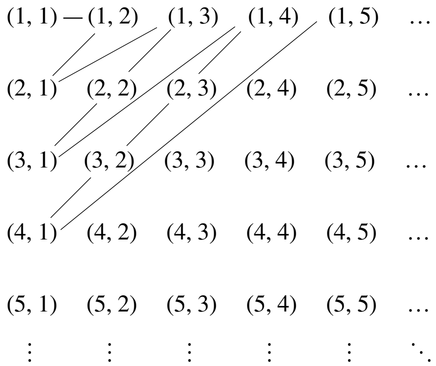


Figure 1-1. Enumerating pairs of positive integers.

come the pairs the sum of whose entries is 3, then come the pairs the sum of whose entries is 4, and so on. Within each block of pairs whose entries have the same sum, pairs appear in order of increasing first entry.

As for the second way, we begin with the thought that while an ordinary hotel may have to turn away a prospective guest because all rooms are full, a hotel with an enumerable infinity of rooms would always have room for one more: The new guest could be placed in room 1, and every other guest asked to move over one room. But actually, a little more thought shows that with foresight the hotelier can be prepared to accommodate a busload with an enumerable infinity of new guests each day, without inconveniencing any old guests by making them change rooms. Those who arrive on the first day are placed in *every other* room, those who arrive on the second day are placed in every other room *among those remaining vacant*, and so on. To apply this thought to enumerating pairs, let us use up every other place in listing the pairs $(1, n)$, every other place then remaining in listing the pairs $(2, n)$, every other place then remaining in listing the pairs $(3, n)$, and so on. The result will look like this:

$$(1, 1), (2, 1), (1, 2), (3, 1), (1, 3), (2, 2), (1, 4), (4, 1), (1, 5), (2, 3), \ldots.$$

If we call the function involved here g, then $g(1) = (1, 1)$, $g(2) = (2, 1)$, $g(3) = (1, 2)$, and so on.

Given a function f enumerating the pairs of positive integers, such as G or g above, an a such that $f(a) = (m, n)$ may be called a *code number* for the pair (m, n). Applying the function f may be called *decoding*, while going the opposite way, from the pair to a code for it, may be called *encoding*. It is actually possible to derive mathematical formulas for the encoding functions J and j that go with the decoding functions G and g above. (Possible, but not necessary: What we have said so far more than suffices as a proof that the set of pairs is enumerable.)

Let us take first J. We want $J(m, n)$ to be the number p such that $G(p) = (m, n)$, which is to say the place p where the pair (m, n) comes in the enumeration corresponding to G. Before we arrive at the pair (m, n), we will have to pass the pair whose entries sum to 2, the two pairs whose entries sum to 3, the three pairs whose entries sum to 4, and so on, up through the $m + n - 2$ pairs whose entries sum to $m + n - 1$.

The pair (m, n) will appear in the mth place after all of these pairs. So the position of the pair (m, n) will be given by

$$[1 + 2 + \cdots + (m + n - 2)] + m.$$

At this point we recall the formula for the sum of the first k positive integers:

$$1 + 2 + \cdots + k = k(k + 1)/2.$$

(Never mind, for the moment, where this formula comes from. Its derivation will be recalled in a later chapter.) So the position of the pair (m, n) will be given by

$$(m + n - 2)(m + n - 1)/2 + m.$$

This simplifies to

$$J(m, n) = (m^2 + 2mn + n^2 - m - 3n + 2)/2.$$

For instance, the pair (3, 2) should come in the place

$$(3^2 + 2 \cdot 3 \cdot 2 + 2^2 - 3 - 3 \cdot 2 + 2)/2 = (9 + 12 + 4 - 3 - 6 + 2)/2 = 18/2 = 9$$

as indeed it can be seen (looking back at the enumeration as displayed above) that it does: $G(9) = (3, 2)$.

Turning now to j, we find matters a bit simpler. The pairs with first entry 1 will appear in the places whose numbers are odd, with $(1, n)$ in place $2n - 1$. The pairs with first entry 2 will appear in the places whose numbers are twice an odd number, with $(2, n)$ in place $2(2n - 1)$. The pairs with first entry 3 will appear in the places whose numbers are four times an odd number, with $(3, n)$ in place $4(2n - 1)$. In general, in terms of the powers of two ($2^0 = 1$, $2^1 = 2$, $2^2 = 4$, and so on), (m, n) will appear in place $j(m, n) = 2^{m-1}(2n - 1)$. Thus (3, 2) should come in the place $2^{3-1}(2 \cdot 2 - 1) = 2^2(4 - 1) = 4 \cdot 3 = 12$, as indeed it does: $g(12) = (3, 2)$.

The series of examples to follow shows how more and more complicated objects can be coded by positive integers. Readers may wish to try to find proofs of their own before reading ours; and for this reason we give the statements of all the examples first, and collect all the proofs afterwards. As we saw already with Example 1.2, several equally good codings may be possible.

1.3 Example. The set of positive rational numbers

1.4 Example. The set of rational numbers

1.5 Example. The set of ordered triples of positive integers

1.6 Example. The set of ordered k-tuples of positive integers, for any fixed k

1.7 Example. The set of finite sequences of positive integers less than 10

1.8 Example. The set of finite sequences of positive integers less than b, for any fixed b

1.9 Example. The set of finite sequences of positive integers

1.10 Example. The set of finite sets of positive integers

1.11 Example. Any subset of an enumerable set

1.12 Example. The union of any two enumerable sets

1.13 Example. The set of finite strings from a finite or enumerable alphabet of symbols

Proofs

Example 1.3. A positive rational number is a number that can be expressed as a *ratio* of positive integers, that is, in the form m/n where m and n are positive integers. Therefore we can get an enumeration of all positive rational numbers by starting with our enumeration of all pairs of positive integers and replacing the pair (m, n) by the rational number m/n. This gives us the list

$$1/1, 1/2, 2/1, 1/3, 2/2, 3/1, 1/4, 2/3, 3/2, 4/1, 1/5, 2/4, 3/3, 4/2, 5/1, 1/6, \ldots$$

or, simplified,

$$1, 1/2, 2, 1/3, 1, 3, 1/4, 2/3, 3/2, 4, 1/5, 1/2, 1, 2, 5/1, 1/6, \ldots.$$

Every positive rational number in fact appears infinitely often, since for instance $1/1 = 2/2 = 3/3 = \cdots$ and $1/2 = 2/4 = \cdots$ and $2/1 = 4/2 = \cdots$ and similarly for every other rational number. But that is all right: our definition of enumerability permits repetitions.

Example 1.4. We combine the ideas of Examples 1.1 and 1.3. You know from Example 1.3 how to arrange the positive rationals in a single infinite list. Write a zero in front of this list, and then write the positive rationals, backwards and with minus signs in front of them, in front of that. You now have

$$\ldots, -1/3, -2, -1/2, -1, 0, 1, 1/2, 2, 1/3, \ldots$$

Finally, use the method of Example 1.1 to turn this into a proper list:

$$0, 1, -1, 1/2, -1/2, 2, -2, 1/3, -1/3, \ldots$$

Example 1.5. In Example 1.2 we have given two ways of listing all pairs of positive integers. For definiteness, let us work here with the first of these:

$$(1, 1), (1, 2), (2, 1), (1, 3), (2, 2), (3, 1), \ldots.$$

Now go through this list, and in each pair replace the second entry or component n with the pair that appears in the nth place on this very list. In other words, replace each 1 that appears in the second place of a pair by (1, 1), each 2 by (1, 2), and so on. This gives the list

$$(1, (1, 1)), (1, (1, 2)), (2, (1, 1)), (1, (2, 1)), (2, (1, 2)), (3, (1, 1)), \ldots$$

and that gives a list of triples

$$(1, 1, 1), (1, 1, 2), (2, 1, 1), (1, 2, 1), (2, 1, 2), (3, 1, 1), \ldots.$$

In terms of functions, this enumeration may be described as follows. The original enumeration of pairs corresponds to a function associating to each positive integer n

a pair $G(n) = (K(n), L(n))$ of positive integers. The enumeration of triples we have just defined corresponds to assigning to each positive integer n instead the triple

$$(K(n), K(L(n)), L(L(n))).$$

We do not miss any triples (p, q, r) in this way, because there will always be an $m = J(q, r)$ such that $(K(m), L(m)) = (q, r)$, and then there will be an $n = J(p, m)$ such that $(K(n), L(n)) = (p, m)$, and the triple associated with this n will be precisely (p, q, r).

Example 1.6. The method by which we have just obtained an enumeration of triples from an enumeration of pairs will give us an enumeration of quadruples from an enumeration of triples. Go back to the original enumeration pairs, and replace each second entry n by the triple that appears in the nth place in the enumeration of triples, to get a quadruple. The first few quadruples on the list will be

$$(1, 1, 1, 1), (1, 1, 1, 2), (2, 1, 1, 1), (1, 2, 1, 1), (2, 1, 1, 2), \ldots.$$

Obviously we can go on from here to quintuples, sextuples, or k-tuples for any fixed k.

Example 1.7. A finite sequence whose entries are all positive integers less than 10, such as (1, 2, 3), can be read as an ordinary decimal or base-10 numeral 123. The number this numeral denotes, one hundred twenty-three, could then be taken as a code number for the given sequence. Actually, for later purposes it proves convenient to modify this procedure slightly and write the sequence *in reverse* before reading it as a numeral. Thus (1, 2, 3) would be coded by 321, and 123 would code (3, 2, 1). In general, a sequence

$$s = (a_0, a_1, a_2, \ldots, a_k)$$

would be coded by

$$a_0 + 10a_1 + 100a_2 + \cdots + 10^k a_k$$

which is the number that the decimal numeral $a_k \cdots a_2 a_1 a_0$ represents. Also, it will be convenient henceforth to call the initial entry of a finite sequence the 0th entry, the next entry the 1st, and so on. To decode and obtain the ith entry of the sequence coded by n, we take the quotient on dividing by 10^i, and then the remainder on dividing by 10. For instance, to find the 5th entry of the sequence coded by 123 456 789, we divide by 10^5 to obtain the quotient 1234, and then divide by 10 to obtain the remainder 4.

Example 1.8. We use a decimal, or base-10, system ultimately because human beings typically have 10 fingers, and counting began with counting on fingers. A similar base-b system is possible for any $b > 1$. For a *binary*, or base-2, system only the ciphers 0 and 1 would be used, with $a_k \ldots a_2 a_1 a_0$ representing

$$a_0 + 2a_1 + 4a_2 + \cdots + 2^k a_k.$$

So, for instance, 1001 would represent $1 + 2^3 = 1 + 8 = 9$. For a duodecimal, or base-12, system, two additional ciphers, perhaps * and # as on a telephone, would be needed for ten and eleven. Then, for instance, 1*# would represent $11 + 12 \cdot 10 + 144 \cdot 1 = 275$. If we applied the idea of the previous problem using base 12 instead of base 10, we could code finite sequences of positive integers less than 12, and not just finite sequences of positive integers less than 10. More generally, we can code a finite sequence

$$s = (a_0, a_1, a_2, \ldots, a_k)$$

of positive integers less than b by

$$a_0 + ba_1 + b^2a_2 + \cdots + b^k a_k.$$

To obtain the ith entry of the sequence coded by n, we take the quotient on dividing by b^i and then the remainder on dividing by b. For example, when working with base 12, to obtain the 5th entry of the sequence coded by 123 456 789, we divide 123 456 789 by 12^5 to get the quotient 496. Now divide by 12 to get remainder 4. In general, working with base b, the ith entry—counting the initial one as the 0th—of the sequence coded by (b, n) will be

$$\text{entry}(i, n) = \text{rem}(\text{quo}(n, b^i), b)$$

where quo(x, y) and rem(x, y) are the quotient and remainder on dividing x by y.

Example 1.9. Coding finite sequences will be important enough in our later work that it will be appropriate to consider several different ways of accomplishing this task. Example 1.6 showed that we can code sequences whose entries may be of any size but that are *of fixed length*. What we now want is an enumeration of *all* finite sequences—pairs, triples, quadruples, and so on—in a single list; and for good measure, let us include the 1-tuples or 1-term sequences (1), (2), (3), . . . as well. A first method, based on Example 1.6, is as follows. Let $G_1(n)$ be the 1-term sequence (n). Let $G_2 = G$, the function enumerating all 2-tuples or pairs from Example 1.2. Let G_3 be the function enumerating all triples as in Example 1.5. Let G_4, $G_5, \ldots,$ be the enumerations of triples, quadruples, and so on, from Example 1.6. We can get a coding of all finite sequences by *pairs* of positive integers by letting any sequence s of length k be coded by the pair (k, a) where $G_k(a) = s$. Since pairs of positive integers can be coded by single numbers, we indirectly get a coding of sequences of numbers. Another way to describe what is going on here is as follows. We go back to our original listing of pairs, and replace the pair (k, a) by the ath item on the list of k-tuples. Thus (1, 1) would be replaced by the first item (1) on the list of 1-tuples (1), (2), (3), . . . ; while (1, 2) would be replaced by the second item (2) on the same list; whereas (2, 1) would be replaced by the first item (1, 1) on the list of all 2-tuples or pairs; and so on. This gives us the list

$$(1), (2), (1, 1), (3), (1, 2), (1, 1, 1), (4), (2, 1), (1, 1, 2), (1, 1, 1, 1), \ldots.$$

(If we wish to include also the 0-tuple or empty sequence ( ), which we may take to be simply the empty set $\varnothing$, we can stick it in at the head of the list, in what we may think of as the 0th place.)

Example 1.8 showed that we can code sequences of any length whose entries are *less than some fixed bound*, but what we now want to do is show how to code sequences of any length whose entries may be *of any size*. A second method, based on Example 1.8, is to begin by coding sequences by pairs of positive integers. We take a sequence

$$s = (a_0, a_1, a_2, \dots, a_k)$$

to be coded by any *pair* (b, n) such that all a_i are less than b, and n codes s in the sense that

$$n = a_0 + b \cdot a_1 + b^2 a_2 + \cdots + b^k a_k.$$

Thus (10, 275) would code (5, 7, 2), since $275 = 5 + 7 \cdot 10 + 2 \cdot 10^2$, while (12, 275) would code (11, 10, 1), since $275 = 11 + 10 \cdot 12 + 1 \cdot 12^2$. Each sequence would have many codes, since for instance (10, 234) and (12, 328) would equally code (4, 3, 2), because $4 + 3 \cdot 10 + 2 \cdot 10^2 = 234$ and $4 + 3 \cdot 12 + 2 \cdot 12^2 = 328$. As with the previous method, since pairs of positive integers can be coded by single numbers, we indirectly get a coding of sequences of numbers.

A third, and totally different, approach is possible, based on the fact that every integer greater than 1 can be written in one and only one way as a product of powers of larger and larger primes, a representation called its *prime decomposition*. This fact enables us to code a sequence $s = (i, j, k, m, n, \dots)$ by the number $2^i 3^j 5^k 7^m 11^n \dots$. Thus the code number for the sequence (3, 1, 2) is $2^3 3^1 5^2 = 8 \cdot 3 \cdot 25 = 600$.

Example 1.10. It is easy to get an enumeration of finite sets from an enumeration of finite sequences. Using the first method in Example 1.9, for instance, we get the following enumeration of sets:

$$\{1\}, \{2\}, \{1, 1\}, \{3\}, \{1, 2\}, \{1, 1, 1\}, \{4\}, \{2, 1\}, \{1, 1, 2\}, \{1, 1, 1, 1\}, \dots .$$

The set {1, 1} whose only elements are 1 and 1 is just the set {1} whose only element is 1, and similarly in other cases, so this list can be simplified to look like this:

$$\{1\}, \{2\}, \{1\}, \{3\}, \{1, 2\}, \{1\}, \{4\}, \{1, 2\}, \{1, 2\}, \{1\}, \{5\}, \dots .$$

The repetitions do not matter.

Example 1.11. Given any enumerable set A and a listing of the elements of A:

$$a_1, a_2, a_3, \dots$$

we easily obtain a gappy listing of the elements of any subset B of A simply by erasing any entry in the list that does not belong to B, leaving a gap.

Example 1.12. Let A and B be enumerable sets, and consider listings of their elements:

$$a_1, a_2, a_3, \dots \qquad b_1, b_2, b_3, \dots .$$

Imitating the *shuffling* idea of Example 1.1, we obtain the following listing of the elements of the union $A \cup B$ (the set whose elements are all and only those items that are elements either of A or of B or of both):

$$a_1, b_1, a_2, b_2, a_3, b_3, \dots .$$

If the intersection $A \cap B$ (the set whose elements of both A and B) is not empty, then there will be redundancies on this list: If $a_m = b_n$, then that element will appear both at place $2m - 1$ and at place $2n$, but this does not matter.

Example 1.13. Given an 'alphabet' of any finite number, or even an enumerable infinity, of symbols $S_1, S_2, S_3, \ldots$ we can take as a code number for any finite string

$$S_{a_0} S_{a_1} S_{a_2} \cdots S_{a_k}$$

the code number for the finite sequence of positive integers

$$(a_1, a_2, a_3, \ldots, a_k)$$

under any of the methods of coding considered in Example 1.9. (We are usually going to use the third method.) For instance, with the ordinary alphabet of 26 symbols letters $S_1 =$ 'A', $S_2 =$ 'B', and so on, the string or word 'CAB' would be coded by the code for (3, 1, 2), which (on the third method of Example 1.9) would be $2^3 \cdot 3 \cdot 5^2 = 600$.

Problems

1.1 A (total or partial) *function* f from a set A to a set B is an assignment for (some or all) elements a of A of an associated element $f(a)$ of B. If $f(a)$ is defined for *every* element a of A, then the function f is called *total*. If every element b of B is assigned to some element a of A, then the function f is said to be *onto*. If no element b of B is assigned to more than one element a of A, then the function f is said to be *one-to-one*. The *inverse* function f^{-1} from B to A is defined by letting $f^{-1}(b)$ be the one and only a such that $f(a) = b$, if any such a exists; $f^{-1}(b)$ is undefined if there is no a with $f(a) = b$ or more than one such a. Show that if f is a one-to-one function and f^{-1}its inverse function, then f^{-1} is total if and only if f is onto, and conversely, f^{-1} is onto if and only if f is total.

1.2 Let f be a function from a set A to a set B, and g a function from the set B to a set C. The *composite* function $h = gf$ from A to C is defined by $h(a) = g(f(a))$. Show that:

(a) If f and g are both total, then so is gf.

(b) If f and g are both onto, then so is gf.

(c) If f and g are both one-to-one, then so is gf.

1.3 A *correspondence* between sets A and B is a one-to-one total function from A onto B. Two sets A and B are said to be *equinumerous* if and only if there is a correspondence between A and B. Show that equinumerosity has the following properties:

(a) Any set A is equinumerous with itself.

(b) If A is equinumerous with B, then B is equinumerous with A.

(c) If A is equinumerous with B and B is equinumerous with C, then A is equinumerous with C.

1.4 A set A has n elements, where n is a positive integer, if it is equinumerous with the set of positive integers up to n, so that its elements can be listed as $a_1, a_2, \ldots, a_n$. A nonempty set A is finite if it has n elements for some positive integer n. Show that any enumerable set is either finite or equinumerous with the set of all positive integers. (In other words, given an *enumeration*, which is to say a function from the set of positive integers onto a set A, show that if A is not finite, then there is a *correspondence*, which is to say a *one-to-one*, *total* function, from the set of positive integers onto A.)

1.5 Show that the following sets are equinumerous:

(a) The set of rational numbers with denominator a power of two (when written in lowest terms), that is, the set of rational numbers $\pm m/n$ where $n = 1$ or 2 or 4 or 8 or some higher power of 2.

(b) The set of those sets of positive integers that are either finite or cofinite, where a set S of positive integers is *cofinite* if the set of all positive integers n that are *not* elements of S is finite.

1.6 Show that the set of all finite subsets of an enumerable set is enumerable.

1.7 Let $A = \{A_1, A_2, A_3, \ldots\}$ be an enumerable family of sets, and suppose that each A_i for $i =$ 1, 2, 3, and so on, is enumerable. Let $\cup A$ be the union of the family A, that is, the set whose elements are precisely the elements of the elements of A. Is $\cup A$ enumerable?

2
Diagonalization

In the preceding chapter we introduced the distinction between enumerable and nonenumerable sets, and gave many examples of enumerable sets. In this short chapter we give examples of nonenumerable sets. We first prove the existence of such sets, and then look a little more closely at the method, called diagonalization, *used in this proof.*

Not all sets are enumerable: some are too big. For example, consider the set of *all sets of positive integers*. This set (call it P^*) contains, as a member, each finite and each infinite set of positive integers: the empty set $\varnothing$, the set P of all positive integers, and every set between these two extremes. Then we have the following celebrated result.

2.1 Theorem (Cantor's Theorem). The set of all sets of positive integers is not enumerable.

Proof: We give a method that can be applied to *any* list L of sets of positive integers in order to discover a set $\Delta(L)$ of positive integers which is not named in the list. If you then try to repair the defect by adding $\Delta(L)$ to the list as a new first member, the same method, applied to the augmented list L^* will yield a different set $\Delta(L^*)$ that is likewise not on the augmented list.

The method is this. Confronted with any infinite list L

$$S_1, S_2, S_3 \ldots.$$

of sets of positive integers, we define a set $\Delta(L)$ as follows:

(*) For each positive integer n, n is in $\Delta(L)$ if and only if n is *not* in S_n.

It should be clear that this genuinely defines a set $\Delta(L)$; for, given any positive integer n, we can tell whether n is in $\Delta(L)$ if we can tell whether n is in the nth set in the list L. Thus, if S_3 happens to be the set E of even positive integers, the number 3 is not in S_3 and therefore it *is* in $\Delta(L)$. As the notation $\Delta(L)$ indicates, the composition of the set $\Delta(L)$ depends on the composition of the list L, so that different lists L may yield different sets $\Delta(L)$.

To show that the set $\Delta(L)$ that this method yields is never in the given list L, we argue by *reductio ad absurdum*: we suppose that $\Delta(L)$ does appear somewhere in list L, say as entry number m, and deduce a contradiction, thus showing that the supposition must be false. Here we go. *Supposition*: For some positive integer m,

$$S_m = \Delta(L).$$

[Thus, if 127 is such an m, we are supposing that $\Delta(L)$ and S_{127} are the same set under different names: we are supposing that a positive integer belongs to $\Delta(L)$ if and only if it belongs to the 127th set in list L.] To deduce a contradiction from this assumption we apply definition (*) to the particular positive integer m: with $n = m$, (*) tells us that

$$m \text{ is in } \Delta(L) \text{ if and only if } m \text{ is } \textit{not} \text{ in } S_m.$$

Now a contradiction follows from our supposition: if S_m and $\Delta(L)$ are one and the same set we have

$$m \text{ is in } \Delta(L) \text{ if and only if } m \text{ is in } S_m.$$

Since this is a flat self-contradiction, our supposition must be false. For no positive integer m do we have $S_m = \Delta(L)$. In other words, the set $\Delta(L)$ is named nowhere in list L.

So the method works. Applied to any list of sets of positive integers it yields a set of positive integers which was not in the list. Then no list enumerates all sets of positive integers: the set P^* of all such sets is not enumerable. This completes the proof.

Note that results to which we might wish to refer back later are given reference numbers 1.1, 1.2, . . . consecutively through the chapter, to make them easy to locate. Different words, however, are used for different kinds of results. The most important general results are dignified with the title of ‘theorem’. Lesser results are called ‘lemmas’ if they are steps on the way to a theorem, ‘corollaries’ if they follow directly upon some theorem, and ‘propositions’ if they are free-standing. In contrast to all these, ‘examples’ are particular rather than general. The most celebrated of the theorems have more or less traditional names, given in parentheses. The fact that 2.1 has been labelled ‘Cantor’s theorem’ is an indication that it is a famous result. The reason is not—we hope the reader will agree!—that its proof is especially difficult, but that the *method* of the proof (*diagonalization*) was an important innovation. In fact, it is so important that it will be well to look at the proof again from a slightly different point of view, which allows the entries in the list L to be more readily visualized.

Accordingly, we think of the sets S_1, $S_2, \ldots$ as represented by functions s_1, $s_2, \ldots$ of positive integers that take the numbers 0 and 1 as values. The relationship between the set S_n and the corresponding function s_n is simply this: for each positive integer p we have

$$s_n(p) = \begin{cases} 1 & \text{if } p \text{ is in } S_n \\ 0 & \text{if } p \text{ is not in } S_n. \end{cases}$$

Then the list can be visualized as an infinite rectangular array of zeros and ones, in which the nth row represents the function s_n and thus represents the set S_n. That is,

$$\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & \\
\hline
s_1 & s_1(1) & s_1(2) & s_1(3) & s_1(4) & \\
s_2 & s_2(1) & s_2(2) & s_2(3) & s_2(4) & \\
s_3 & s_3(1) & s_3(2) & s_3(3) & s_3(4) & \\
s_4 & s_4(1) & s_4(2) & s_4(3) & s_4(4) & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Figure 2-1. A list as a rectangular array.

the nth row

$$s_n(1)s_n(2)s_n(3)s_n(4)\ldots$$

is a sequence of zeros and ones in which the pth entry, $s_n(p)$, is 1 or 0 according as the number p is or is not in the set S_n. This array is shown in Figure 2-1.

The entries in the diagonal of the array (upper left to lower right) form a sequence of zeros and ones:

$$s_1(1)\,s_2(2)\,s_3(3)\,s_4(4)\ldots$$

This sequence of zeros and ones (the *diagonal sequence*) determines a set of positive integers (the *diagonal set*). *The diagonal set may well be among those listed in L.* In other words, there may well be a positive integer d such that the set S_d is none other than our diagonal set. The sequence of zeros and ones in the dth row of Figure 2-1 would then agree with the diagonal sequence entry by entry:

$$s_d(1) = s_1(1), \qquad s_d(2) = s_2(2), \qquad s_d(3) = s_3(3), \ldots.$$

That is as may be: the diagonal set may or may not appear in the list L, depending on the detailed makeup of the list. What we want is a set we can rely upon *not* to appear in L, no matter how L is composed. Such a set lies near to hand: it is the *antidiagonal set*, which consists of the positive integers not in the diagonal set. The corresponding *antidiagonal sequence* is obtained by changing zeros to ones and ones to zeros in the diagonal sequence. We may think of this transformation as a matter of subtracting each member of the diagonal sequence from 1: we write the antidiagonal sequence as

$$1 - s_1(1), 1 - s_2(2), 1 - s_3(3), 1 - s_4(4), \ldots.$$

This sequence can be relied upon not to appear as a row in Figure 2-1, for if it did appear—say, as the mth row—we should have

$$s_m(1) = 1 - s_1(1), \qquad s_m(2) = 1 - s_2(2), \ldots, \qquad s_m(m) = 1 - s_m(m), \ldots.$$

But the mth of these equations cannot hold. [*Proof*: $s_m(m)$ must be zero or one. If zero, the mth equation says that $0 = 1$. If one, the mth equation says that $1 = 0$.] Then the antidiagonal sequence differs from every row of our array, and so the antidiagonal set differs from every set in our list L. This is no news, for the antidiagonal set is simply the set $\Delta(L)$. We have merely repeated with a diagram—Figure 2-1—our proof that $\Delta(L)$ appears nowhere in the list L.

Of course, it is rather strange to say that the members of an infinite set 'can be arranged' in a single list. By whom? Certainly not by any human being, for nobody has that much time or paper; and similar restrictions apply to machines. In fact, to call a set enumerable is simply to say that it is the range of some total or partial function of positive integers. Thus, the set E of even positive integers is enumerable because there are functions of positive integers that have E as their range. (We had two examples of such functions earlier.) Any such function can then be thought of as a program that a superhuman enumerator can follow in order to arrange the members of the set in a single list. More explicitly, the program (the set of instructions) is: 'Start counting from 1, and never stop. As you reach each number n, write a name of $f(n)$ in your list. [Where $f(n)$ is undefined, leave the nth position blank.]' But there is no need to refer to the list, or to a superhuman enumerator: anything we need to say about enumerability can be said in terms of the functions themselves; for example, to say that the set P^* is not enumerable is simply to deny the existence of any function of positive integers which has P^* as its range.

Vivid talk of lists and superhuman enumerators may still aid the imagination, but in such terms the theory of enumerability and diagonalization appears as a chapter in mathematical theology. To avoid treading on any living toes we might put the whole thing in a classical Greek setting: Cantor proved that there are sets which even Zeus cannot enumerate, no matter how fast he works, or how long (even, infinitely long).

If a set *is* enumerable, Zeus can enumerate it in one second by writing out an infinite list faster and faster. He spends $1/2$ second writing the first entry in the list; $1/4$ second writing the second entry; $1/8$ second writing the third; and in general, he writes each entry in half the time he spent on its predecessor. At no point *during* the one-second interval has he written out the whole list, but when one second has passed, the list is complete. On a time scale in which the marked divisions are sixteenths of a second, the process can be represented as in Figure 2-2.

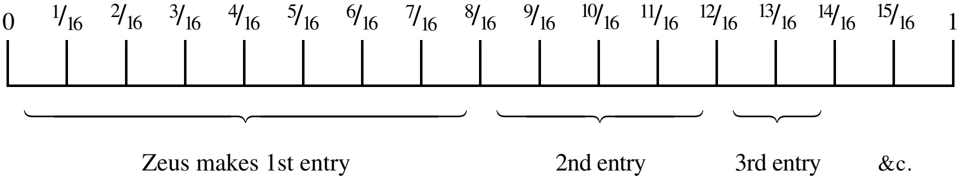


Figure 2-2. Completing an infinite process in finite time.

To speak of writing out an infinite list (for example, of all the positive integers, in decimal notation) is to speak of such an enumerator either working faster and faster as above, or taking all of infinite time to complete the list (making one entry per second, perhaps). Indeed, Zeus could write out an infinite sequence of infinite lists if he chose to, taking only one second to complete the job. He could simply allocate the first half second to the business of writing out the first infinite list ($1/4$ second for the first entry, $1/8$ second for the next, and so on); he could then write out the whole second list in the following quarter second ($1/8$ for the first entry, $1/16$ second for the next, and so on); and in general, he could write out each subsequent list in just half the time he spent on its predecessor, so that after one second had passed he would have written out every entry in every list, in order. But the result does not count as a single infinite list, in our sense of the term. In our sort of list, each entry must come some *finite* number of places after the first.

As we use the term 'list', Zeus has not produced a list by writing infinitely many infinite lists one after another. But he could perfectly well produce a genuine list which exhausts the entries in all the lists, by using some such device as we used in the preceeding chapter to enumerate the positive rational numbers. Nevertheless, Cantor's *diagonal argument* shows that neither this nor any more ingenious device is available, even to a god, for arranging all the sets of positive integers into a single infinite list. Such a list would be as much an impossibility as a round square: the impossibility of enumerating all the sets of positive integers is as absolute as the impossibility of drawing a round square, even for Zeus.

Once we have one example of a nonenumerable set, we get others.

2.2 Corollary. The set of real numbers is not enumerable.

Proof: If ξ is a real number and $0 < \xi < 1$, then ξ has a decimal expansion $.x_1x_2x_3\ldots$ where each x_i is one of the cyphers 0–9. Some numbers have two decimal expansions, since for instance $.2999\ldots = .3000\ldots$; so if there is a choice, choose the one with the 0s rather than the one with the 9s. Then associate to ξ the set of all positive integers n such that a 1 appears in the nth place in this expansion. Every set of positive integers is associated to some real number (the sum of 10^{-n} for all n in the set), and so an enumeration of the real numbers would immediately give rise to an enumeration of the sets of positive integers, which cannot exist, by the preceding theorem.

Problems

2.1 Show that the set of all subsets of an infinite enumerable set is nonenumerable.

2.2 Show that if for some or all of the finite strings from a given finite or enumerable alphabet we associate to the string a total or partial function from positive integers to positive integers, then there is some total function on positive integers taking only the values 1 and 2 that is not associated with any string.

2.3 In mathematics, the real numbers are often identified with the points on a line. Show that the set of real numbers, or equivalently, the set of points on the line, is equinumerous with the set of points on the semicircle indicated in Figure 2-3.

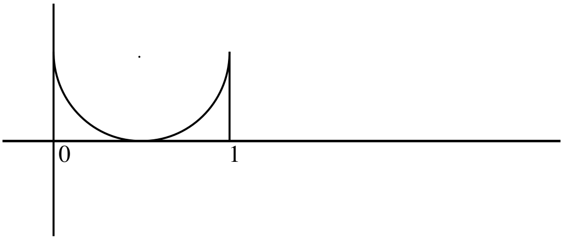


Figure 2-3. Interval, semicircle, and line.

2.4 Show that the set of real numbers ξ with $0 < \xi < 1$, or equivalently, the set of points on the interval shown in Figure 2-3, is equinumerous with the set of points on the semicircle.

2.5 Show that the set of real numbers ξ with $0 < \xi < 1$ is equinumerous with the set of *all* real numbers.

2.6 A real number x is called *algebraic* if it is a solution to some equation of the form

$$c_d x^d + c_{d-1} x^{d-1} + c_{d-2} x^{d-2} + \cdots + c_2 x^2 + c_1 x + c_0 = 0$$

where the c_i are rational numbers and $c_d \neq 0$. For instance, for any rational number r, the number r itself is algebraic, since it is the solution to $x - r = 0$; and the square root $\sqrt{r}$ of r is algebraic, since it is a solution to $x^2 - r = 0$.

(a) Use the fact from algebra that an equation like the one displayed has at most d solutions to show that every algebraic number can be described by a finite string of symbols from an ordinary keyboard.

(b) A real number that is not algebraic is called *transcendental.* Prove that transcendental numbers exist.

2.7 Each real number ξ with $0 < \xi < 1$ has a binary representation $0 \cdot x_1 x_2 x_3 \ldots$ where each x_i is a digit 0 or 1, and the successive places represent halves, quarters, eighths, and so on. Show that the set of real numbers, ξ with $0 < \xi < 1$ and ξ not a rational number with denominator a power of two, is equinumerous with the set of those sets of positive integers that are neither finite nor cofinite.

2.8 Show that if A is equinumerous with C and B is equinumerous with D, and the intersections $A \cap B$ and $C \cap D$ are empty, then the unions $A \cup B$ and $C \cup D$ are equinumerous.

2.9 Show that the set of real numbers ξ with $0 < \xi < 1$ (and hence by an earlier problem the set of *all* real numbers) is equinumerous with the set of all sets of positive integers.

2.10 Show that the following sets are equinumerous:

(a) the set of all pairs of sets of positive integers

(b) the set of all sets of pairs of positive integers

(c) the set of all sets of positive integers.

2.11 Show that the set of points on a line is equinumerous with the set of points on a plane.

2.12 Show that the set of points on a line is equinumerous with the set of points in space.

2.13 (*Richard's paradox*) What (if anything) is wrong with the following argument?

> The set of all finite strings of symbols from the alphabet, including the space, capital letters, and punctuation marks, is enumerable; and for definiteness let us use the specific enumeration of finite strings based on prime decomposition. Some strings amount to definitions in English of sets of positive integers and others do not. Strike out the ones that do not, and we are left with an enumeration of all definitions in English of sets of positive integers, or, replacing each definition by the set it defines, an enumeration of all sets of positive integers that have definitions in English. Since some sets have more than one definition, there will be redundancies in this enumeration

of sets. Strike them out to obtain an irredundant enumeration of all sets of positive integers that have definitions in English. Now consider the set of positive integers defined by the condition that a positive integer n is to belong to the set if and only if it does *not* belong to the nth set in the irredundant enumeration just described.

This set does *not* appear in that enumeration. For it cannot appear at the nth place for any n, since there is a positive integer, namely n itself, that belongs to this set if and only if it does *not* belong to the nth set in the enumeration. Since this set does not appear in our enumeration, it cannot have a definition in English. And yet it does have a definition in English, and in fact we have just given such a definition in the preceding paragraph.

3

Turing Computability

A function is effectively computable *if there are definite, explicit rules by following which one could in principle compute its value for any given arguments. This notion will be further explained below, but even after further explanation it remains an intuitive notion. In this chapter we pursue the analysis of computability by introducing a rigorously defined notion of a* Turing-computable *function. It will be obvious from the definition that Turing-computable functions are effectively computable. The hypothesis that, conversely, every effectively computable function is Turing computable is known as* Turing's thesis. *This thesis is not obvious, nor can it be rigorously proved (since the notion of effective computability is an intuitive and not a rigorously defined one), but an enormous amount of evidence has been accumulated for it. A small part of that evidence will be presented in this chapter, with more in chapters to come. We first introduce the notion of Turing machine, give examples, and then present the official definition of what it is for a function to be computable by a Turing machine, or Turing computable.*

A superhuman being, like Zeus of the preceding chapter, could perhaps write out the whole table of values of a one-place function on positive integers, by writing each entry twice as fast as the one before; but for a human being, completing an infinite process of this kind is impossible in principle. Fortunately, for human purposes we generally do not need the whole table of values of a function f, but only need the values one at a time, so to speak: given some argument n, we need the value $f(n)$. If it is possible to produce the value $f(n)$ of the function f for argument n whenever such a value is needed, then that is almost as good as having the whole table of values written out in advance.

A function f from positive integers to positive integers is called *effectively computable* if a list of instructions can be given that in principle make it possible to determine the value $f(n)$ for any argument n. (This notion extends in an obvious way to two-place and many-place functions.) The instructions must be completely definite and explicit. They should tell you at each step what to do, not tell you to go ask someone else what to do, or to figure out for yourself what to do: the instructions should require no external sources of information, and should require no ingenuity to execute, so that one might hope to automate the process of applying the rules, and have it performed by some mechanical device.

There remains the fact that for all but a finite number of values of n, it will be infeasible in practice for any human being, or any mechanical device, actually to carry out the computation: in principle it could be completed in a finite amount of time if we stayed in good health so long, or the machine stayed in working order so long; but in practice we will die, or the machine will collapse, long before the process is complete. (There is also a worry about finding enough space to store the intermediate results of the computation, and even a worry about finding enough matter to use in writing down those results: there's only a finite amount of paper in the world, so you'd have to writer smaller and smaller without limit; to get an infinite number of symbols down on paper, eventually you'd be trying to write on molecules, on atoms, on electrons.) But our present study will ignore these practical limitations, and work with an idealized notion of computability that goes beyond what actual people or actual machines can be sure of doing. Our eventual goal will be to prove that certain functions are *not* computable, *even if* practical limitations on time, speed, and amount of material could somehow be overcome, and for this purpose the essential requirement is that our notion of computability not be too *narrow*.

So far we have been sliding over a significant point. When we are given as argument a number n or pair of numbers (m, n), what we in fact are directly given is a *numeral* for n or an ordered pair of *numerals* for m and n. Likewise, if the value of the function we are trying to compute is a number, what our computations in fact end with is a *numeral* for that number. Now in the course of human history a great many systems of numeration have been developed, from the primitive *monadic* or *tally* notation, in which the number n is represented by a sequence of n strokes, through systems like Roman numerals, in which bunches of five, ten, fifty, one-hundred, and so forth strokes are abbreviated by special symbols, to the *Hindu–Arabic* or *decimal* notation in common use today. Does it make a difference in a definition of computability which of these many systems we adopt?

Certainly computations can be *harder in practice* with some notations than with others. For instance, multiplying numbers given in decimal numerals (expressing the product in the same form) is easier in practice than multiplying numbers given in something like Roman numerals. Suppose we are given two numbers, expressed in Roman numerals, say XXXIX and XLVIII, and are asked to obtain the product, also expressed in Roman numerals. Probably for most us the easiest way to do this would be first to translate from Roman to Hindu–Arabic—the rules for doing this are, or at least used to be, taught in primary school, and in any case can be looked up in reference works—obtaining 39 and 48. Next one would carry out the multiplication in our own more convenient numeral system, obtaining 1872. Finally, one would translate the result back into the inconvenient system, obtaining MDCCCLXXII. Doing all this is, of course, harder than simply performing a multiplication on numbers given by decimal numerals to begin with.

But the example shows that when a computation can be done in one notation, it is *possible in principle* to do in any other notation, simply by translating the data from the difficult notation into an easier one, performing the operation using the easier notation, and then translating the result back from the easier to the difficult notation. If a function is effectively computable when numbers are represented in one system of numerals, it will also be so when numbers are represented in any other system of numerals, provided only that translation between the systems can itself be carried out according to explicit rules, which is the case for any historical system of numeration that we have been able to decipher. (To say we have been able to decipher it amounts to saying that there are rules for translating back and forth between it and the system now in common use.) For purposes of framing a rigorously defined notion of computability, it is convenient to use monadic or tally notation.

A *Turing machine* is a specific kind of idealized machine for carrying out computations, especially computations on positive integers represented in monadic notation. We suppose that the computation takes place on a tape, marked into squares, which is unending in both directions—either because it is actually infinite or because there is someone stationed at each end to add extra blank squares as needed. Each square either is *blank*, or has a *stroke* printed on it. (We represent the blank by S_0 or 0 or most often B, and the stroke by S_1 or | or most often 1, depending on the context.) And with at most a finite number of exceptions, all squares are blank, both initially and at each subsequent stage of the computation.

At each stage of the computation, the computer (that is, the human or mechanical agent doing the computation) is *scanning* some one square of the tape. The computer is capable of erasing a stroke in the scanned square if there is one there, or of printing a stroke if the scanned square is blank. And he, she, or it is capable of movement: one square to the right or one square to the left at a time. If you like, think of the machine quite crudely, as a box on wheels which, at any stage of the computation, is over some square of the tape. The tape is like a railroad track; the ties mark the boundaries of the squares; and the machine is like a very short car, capable of moving along the track in either direction, as in Figure 3-1.

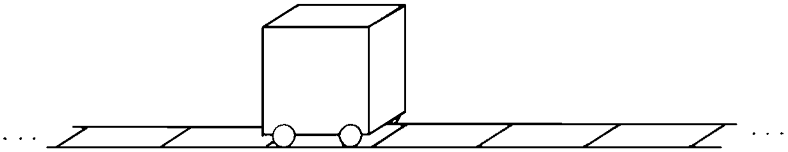


Figure 3-1. A Turing machine.

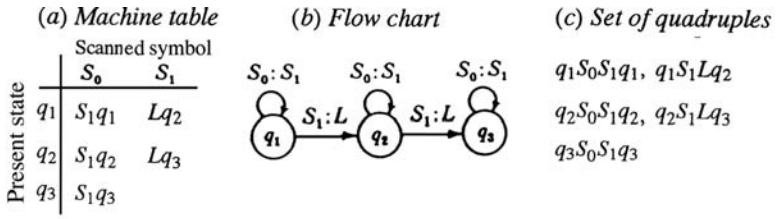
At the bottom of the car there is a device that can read what's written between the ties, and erase or print a stroke. The machine is designed in such a way that at each stage of the computation it is in one of a finite number of internal *states*, $q_1, \ldots, q_m$. Being in one state or another might be a matter of having one or another cog of a certain gear uppermost, or of having the voltage at a certain terminal inside the machine at one or another of m different levels, or what have you: we are not concerned with the mechanics or the electronics of the matter. Perhaps the simplest way to picture the thing is quite crudely: inside the box there is a little man, who does all the reading and writing and erasing and moving. (The box has no bottom: the poor mug just walks along between the ties, pulling the box along.) This operator inside the machine has a list of m instructions written down on a piece of paper and *is in state q_i when carrying out instruction number i.*

Each of the instructions has conditional form: it tells what to do, depending on whether the symbol being scanned (the symbol in the scanned square) is the blank or stroke, S_0 or S_1. Namely, there are five things that can be done:

(1) Erase: write S_0 in place of whatever is in the scanned square.
(2) Print: write S_1 in place of whatever is in the scanned square.
(3) Move one square to the right.
(4) Move one square to the left.
(5) Halt the computation.

[In case the square is already blank, (1) amounts to doing nothing; in case the square already has a stroke in it, (2) amounts to doing nothing.] So depending on what instruction is being carried out (= what state the machine, or its operator, is in) and on what symbol is being scanned, the machine or its operator will perform one or another of these five overt acts. Unless the computation has halted (overt act number 5), the machine or its operator will perform also a covert act, in the privacy of box, namely, the act of determining what the *next* instruction (*next* state) is to be. Thus the *present* state and the presently *scanned symbol* determine what overt *act* is to be performed, and what the *next state* is to be.

The overall *program* of instructions can be specified in various ways, for example, by a *machine table*, or by a *flow chart* (also called a *flow graph*), or by a *set of quadruples*. For the case of a machine that writes three symbols S_1 on a blank tape and then halts, scanning the leftmost of the three, the three sorts of description are illustrated in Figure 3-2.



(*a*) *Machine table*

Present state	Scanned symbol S_0	S_1
q_1	S_1q_1	Lq_2
q_2	S_1q_2	Lq_3
q_3	S_1q_3	

(*b*) *Flow chart*

(*c*) *Set of quadruples*

$q_1S_0S_1q_1$, $q_1S_1Lq_2$
$q_2S_0S_1q_2$, $q_2S_1Lq_3$
$q_3S_0S_1q_3$

Figure 3-2. A Turing machine program.

3.1 Example (Writing a specified number of strokes). We indicate in Figure 3-2 a machine that will write the symbol S_1 three times. A similar construction works for any specified symbol and any specified number of times. The machine will write an S_1 on the square it's initially scanning, move left one square, write an S_1 there, move left one more square, write an S_1 there, and halt. (It halts when it has no further instructions.) There will be three states—one for each of the symbols S_1 that are to be written. In Figure 3-2, the entries in the top row of the machine table (under the horizontal line) tell the machine or its operator, when following instruction q_1, that (1) an S_1 is to be written and instruction q_1 is to be repeated, if the scanned symbol is S_0, but that (2) the machine is to move left and follow instruction q_2 next, if the scanned symbol is S_1. The same information is given in the flow chart by the two arrows that emerge from the node marked q_1; and the same information is also given by the first two quadruples. The significance in general of a table entry, of an arrow in a flow chart, and of a quadruple is shown in Figure 3-3.

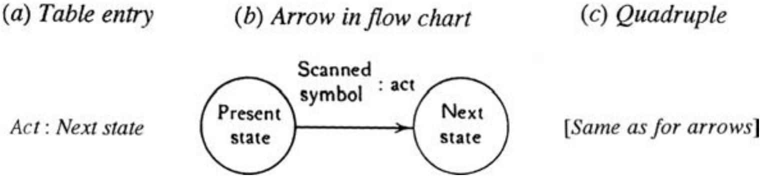


Figure 3-3. A Turing machine instruction.

Unless otherwise stated, it is to be understood that a machine starts in its lowest-numbered state. The machine we have been considering *halts* when it is in state q_3 scanning S_1, for there is no table entry or arrow or quadruple telling it what to do in such a case. A virtue of the flow chart as a way of representing the machine program is that if the starting state is indicated somehow (for example, if it is understood that the leftmost node represents the starting state unless there is an indication to the contrary), then we can dispense with the names of the states: It doesn't matter what you call them. Then the flow chart could be redrawn as in Figure 3-4.

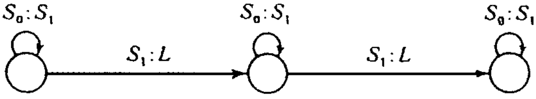


Figure 3-4. Writing three strokes.

We can indicate how such a Turing machine operates by writing down its sequence of *configurations*. There is one configuration for each stage of the computation, showing what's on the tape at that stage, what state the machine is in at that stage, and which square is being scanned. We can show this by writing out what's on the tape and writing the name of the present state under the symbol in the scanned square; for instance,

$$\begin{array}{l}1100111\\ 2\end{array}$$

shows a string or block of two strokes followed by two blanks followed by a string or block of three strokes, with the machine scanning the leftmost stroke and in state 2. Here we have written the symbols S_0 and S_1 simply as 0 and 1, and similarly the state q_2 simply as 2, to save needless fuss. A slightly more compact representation writes the state number as a subscript on the symbol scanned: $1_2100111$.

This same configuration could be written $01_2100111$ or $1_21001110$ or $01_21001110$ or $001_2100111$ or . . . —a block of 0s can be written at the beginning or end of the tape, and can be shorted or lengthened *ad lib.* without changing the significance: the tape is understood to have as many blanks as you please at each end.

We can begin to get a sense of the power of Turing machines by considering some more complex examples.

3.2 Example (Doubling the number of strokes). The machine starts off scanning the leftmost of a block of strokes on an otherwise blank tape, and winds up scanning the leftmost of a block of twice that many strokes on an otherwise blank tape. The flow chart is shown in Figure 3-5.

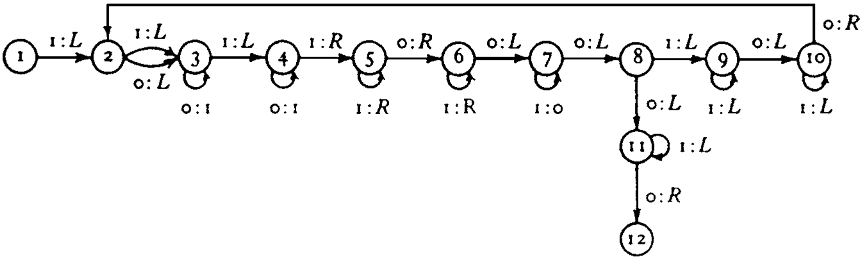


Figure 3-5. Doubling the number of strokes.

How does it work? In general, by writing double strokes at the left and erasing single strokes at the right. In particular, suppose the initial configuration is $1_{1}11$, so that we start in state 1, scanning the leftmost of a block of three strokes on an otherwise blank tape. The next few configurations are as follows:

$$0_{2}111 \qquad 0_{3}0111 \qquad 1_{3}0111 \qquad 0_{4}10111 \qquad 1_{4}10111.$$

So we have written our first double stroke at the left—separated from the original block 111 by a blank. Next we go right, past the blank to the right-hand end of the original block, and erase the rightmost stroke. Here is how that works, in two phases. Phase 1:

$$11_{5}0111 \qquad 110_{5}111 \qquad 1101_{6}11 \qquad 11011_{6}1 \qquad 110111_{6} \qquad 1101110_{6}.$$

Now we know that we have passed the last of the original block of strokes, so (phase 2) we back up, erase one of them, and move one more square left:

$$110111_{7} \qquad 110110_{7} \qquad 11011_{8}0.$$

Now we hop back left, over what is left of the original block of strokes, over the blank separating the original block from the additional strokes we have printed, and over those additional strokes, until we find the blank beyond the leftmost stroke:

$$1101_{9}1 \qquad 110_{9}11 \qquad 11_{10}011 \qquad 1_{10}1011 \qquad 0_{10}11011.$$

Now we will print another two new strokes, much as before:

$$01_{2}1011 \qquad 0_{3}11011 \qquad 1_{3}11011 \qquad 0_{4}111011 \qquad 1_{4}111011.$$

We are now back on the leftmost of the block of newly printed strokes, and the process that led to finding and erasing the rightmost stroke will be repeated, until we arrive at the following:

$$1111011_{7} \qquad 1111010_{7} \qquad 111101_{8}0.$$

Another round of this will lead first to writing another pair of strokes:

$$1_{4}1111101.$$

It will then lead to erasing the last of the original block of strokes:

$$11111101_7 \qquad 11111100_7 \qquad 1111110_8 0.$$

And now the endgame begins, for we have what we want on the tape, and need only move back to halt on the leftmost stroke:

$$111111_{11} \qquad 11111_{11}1 \qquad 1111_{11}11 \qquad 111_{11}111 \qquad 11_{11}1111 \qquad 1_{11}11111$$
$$0_{11}111111 \qquad 1_{12}11111.$$

Now we are in state 12, scanning a stroke. Since there is no arrow from that node telling us what to do in such a case, we halt. The machine performs as advertised.

(*Note*: The fact that the machine doubles the number of strokes when the original number is three is not a proof that the machine performs as advertised. But our examination of the special case in which there are three strokes initially made no essential use of the fact that the initial number was three: it is readily converted into a proof that the machine doubles the number of strokes no matter how long the original block may be.)

Readers may wish, in the remaining examples, to try to design their own machines before reading our designs; and for this reason we give the statements of all the examples first, and collect all the proofs afterward.

3.3 Example (Determining the parity of the length of a block of strokes). There is a Turing machine that, started scanning the leftmost of an unbroken block of strokes on an otherwise blank tape, eventually halts, scanning a square on an otherwise blank tape, where the square contains a blank or a stroke depending on whether there were an even or an odd number of strokes in the original block.

3.4 Example (Adding in monadic (tally) notation). There is a Turing machine that does the following. Initially, the tape is blank except for two solid blocks of strokes, say a left block of p strokes and a right block of q strokes, separated by a single blank. Started on the leftmost blank of the left block, the machine eventually halts, scanning the leftmost stroke in a solid block of $p + q$ stokes on an otherwise blank tape.

3.5 Example (Multiplying in monadic (tally) notation). There is a Turing machine that does the same thing as the one in the preceding example, but with $p \cdot q$ in place of $p + q$.

Proofs

Example 3.3. A flow chart for such a machine is shown in Figure 3-6.

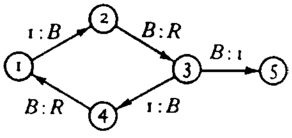


Figure 3-6. Parity machine.

If there were 0 or 2 or 4 or . . . strokes to begin with, this machine halts in state 1, scanning a blank on a blank tape; if there were 1 or 3 or 5 or . . . , it halts in state 5, scanning a stroke on an otherwise blank tape.

Example 3.4. The object is to erase the leftmost stroke, fill the gap between the two blocks of strokes, and halt scanning the leftmost stroke that remains on the tape. Here is one way of doing it, in quadruple notation: $q_1S_1S_0q_1$; $q_1S_0Rq_2$; $q_2S_1Rq_2$; $q_2S_0S_1q_3$; $q_3S_1Lq_3$; $q_3S_0Rq_4$.

Example 3.5. A flow chart for a machine is shown in Figure 3-7.

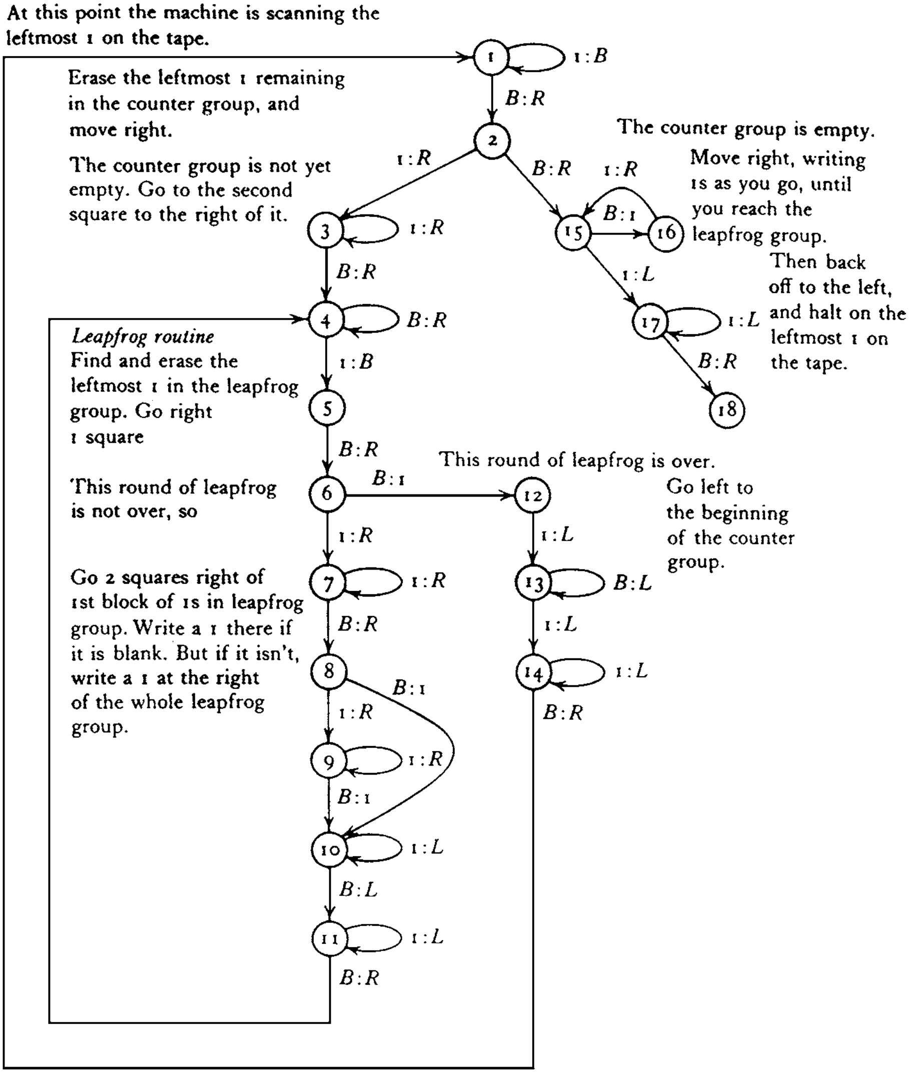

At this point the machine is scanning the leftmost 1 on the tape.
1
1:B
B:R
2
Erase the leftmost 1 remaining in the counter group, and move right.
The counter group is not yet empty. Go to the second square to the right of it.
1:R
3
1:R
B:R
The counter group is empty. Move right, writing 1s as you go, until you reach the leapfrog group. Then back off to the left, and halt on the leftmost 1 on the tape.
B:R
1:R
B:1
15
16
1:L
17
1:L
B:R
18
4
B:R
Leapfrog routine
Find and erase the leftmost 1 in the leapfrog group. Go right 1 square
1:B
5
B:R
This round of leapfrog is not over, so
6
B:1
This round of leapfrog is over. Go left to the beginning of the counter group.
12
1:R
1:L
Go 2 squares right of 1st block of 1s in leapfrog group. Write a 1 there if it is blank. But if it isn't, write a 1 at the right of the whole leapfrog group.
7
1:R
13
B:L
B:R
1:L
8
B:1
14
1:L
1:R
B:R
9
1:R
B:1
10
1:L
B:L
11
1:L
B:R

Figure 3-7. Multiplication machine.

Here is how the machine works. The first block, of p strokes, is used as a *counter*, to keep track of how many times the machine has added q strokes to the group at the right. To start, the machine erases the leftmost of the p strokes and sees if there are any strokes left in the counter group. If not, $pq = q$, and all the machine has to do is position itself over the leftmost stroke on the tape, and halt.

But if there are any strokes left in the counter, the machine goes into a *leapfrog routine*: in effect, it moves the block of q strokes (the *leapfrog group*) q places to the right along the tape. For example, with $p = 2$ and $q = 3$ the tape looks like this initially:

$$11B111$$

and looks like this after going through the leapfrog routine:

$$B1BBBB111.$$

The machine will then note that there is only one 1 left in the counter, and will finish up by erasing that 1, moving right two squares, and changing all Bs to strokes until it comes to a stroke, at which point it continues to the leftmost 1 and halts.

The general picture of how the leapfrog routine works is shown in Figure 3-8.

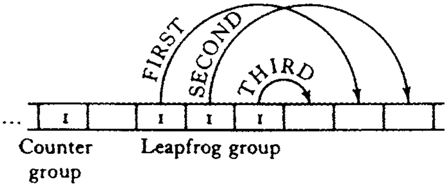


Figure 3-8. Leapfrog.

In general, the leapfrog group consists of a block of 0 or 1 or . . . or q strokes, followed by a blank, followed by the remainder of the q strokes. The blank is there to tell the machine when the leapfrog game is over: without it the group of q strokes would keep moving right along the tape forever. (In playing leapfrog, the portion of the q strokes to the left of the blank in the leapfrog group functions as a counter: it controls the process of adding strokes to the portion of the leapfrog group to the right of the blank. That is why there are two big loops in the flow chart: one for each counter-controlled subroutine.)

We have not yet given an official definition of what it is for a numerical function to be computable by a Turing machine, specifying how inputs or arguments are to be represented on the machine, and how outputs or values represented. Our specifications for a k-place function from positive integers to positive integers are as follows:

(a) The arguments $m_1, \ldots, m_k$ of the function will be represented in monadic notation by blocks of those numbers of strokes, each block separated from the next by a single blank, on an otherwise blank tape. Thus, at the beginning of the computation of, say, $3 + 2$, the tape will look like this: $111B11$.

(b) Initially, the machine will be scanning the leftmost 1 on the tape, and will be in its initial state, state 1. Thus in the computation of $3 + 2$, the initial configuration will be $1_1 11B11$. A configuration as described by (a) and (b) is called a *standard initial configuration* (or *position*).

(c) If the function that is to be computed assigns a value n to the arguments that are represented initially on the tape, then the machine will eventually halt on a tape containing a block of that number of strokes, and otherwise blank. Thus in the computation of $3 + 2$, the tape will look like this: 11111.

(d) In this case, the machine will halt scanning the leftmost 1 on the tape. Thus in the computation of $3 + 2$, the final configuration will be $1_n 1111$, where nth state is one for which there is no instruction what to do if scanning a stroke, so that in this configuration the machine will be halted. A configuration as described by (c) and (d) is called a *standard final configuration* (or *position*).

(e) If the function that is to be computed assigns no value to the arguments that are represented initially on the tape, then the machine either will never halt, or will halt in some nonstandard configuration such as $B_n 11111$ or $B11_n 111$ or $B11111_n$.

The restriction above to the standard position (scanning the leftmost 1) for starting and halting is inessential, but *some* specifications or other have to be made about initial and final positions of the machine, and the above assumptions seem especially simple.

With these specifications, any Turing machine can be seen to compute a function of one argument, a function of two arguments, and, in general, a function of k arguments for each positive integer k. Thus consider the machine specified by the single quadruple $q_1 11 q_2$. Started in a standard initial configuration, it immediately halts, leaving the tape unaltered. If there was only a single block of strokes on the tape initially, its final configuration will be standard, and thus this machine computes the identity function id of one argument: $\text{id}(m) = m$ for each positive integer m. Thus the machine computes a certain total function of one argument. But if there were two or more blocks of strokes on the tape initially, the final configuration will not be standard. Accordingly, the machine computes the extreme partial function of two arguments that is undefined for all pairs of arguments: the empty function e_2 of two arguments. And in general, for k arguments, this machine computes the empty function e_k of k arguments.

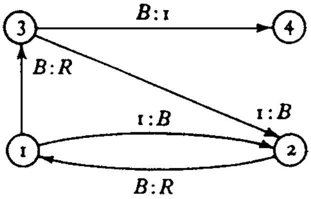


Figure 3-9. A machine computing the value 1 for all arguments.

By contrast, consider the machine whose flow chart is shown in Figure 3-9. This machine computes for each k the total function that assigns the same value, namely 1, to each k-tuple. Started in initial state 1 in a standard initial configuration, this machine erases the first block of strokes (cycling between states 1 and 2 to do so) and goes to state 3, scanning the second square to the right of the first block. If it sees a blank there, it knows it has erased the whole tape, and so prints a single 1 and halts in state 4, in a standard configuration. If it sees a stroke there, it re-enters the cycle between states 1 and 2, erasing the second block of strokes and inquiring again, in state 3, whether the whole tape is blank, or whether there are still more blocks to be dealt with.

A numerical function of k arguments is *Turing computable* if there is some Turing machine that computes it in the sense we have just been specifying. Now computation in the Turing-machine sense is certainly *one* kind of computation in the intuitive sense, so all Turing-computable functions are effectively computable. *Turing's thesis* is that, conversely, any effectively computable function is Turing computable, so that computation in the precise technical sense we have been developing coincides with effective computability in the intuitive sense.

It is easy to imagine liberalizations of the notion of the Turing machine. One could allow machines using more symbols than just the blank and the stroke. One could allow machines operating on a rectangular grid, able to move up or down a square as well as left or right. Turing's thesis implies that no liberalization of the notion of Turing machine will enlarge the class of functions computable, because all functions that are effectively computable in any way at all are already computable by a Turing machine of the restricted kind we have been considering. Turing's thesis is thus a bold claim.

It *is* possible to give a heuristic argument for it. After all, effective computation consists of moving around and writing and perhaps erasing symbols, according to definite, explicit rules; and surely writing and erasing symbols can be done stroke by stroke, and moving from one place to another can be done step by step. But the main argument will be the accumulation of examples of effectively computable functions that we succeed in showing are Turing computable. So far, however, we have had just a few examples of Turing machines computing numerical functions, that is, of effectively computable functions that we have proved to be Turing computable: addition and multiplication in the preceding section, and just now the identity function, the empty function, and the function with constant value 1.

Now addition and multiplication are just the first two of a series of arithmetic operations all of which are effectively computable. The next item in the series is exponentiation. Just as multiplication is repeated addition, so exponentiation is repeated multiplication. (Then repeated exponentiation gives a kind of super-exponentiation, and so on. We will investigate this general process of defining new functions from old in a later chapter.) If Turing's thesis is correct, there must be a Turing machine for each of these functions, computing it. Designing a multiplier was already difficult enough to suggest that designing an exponentiator would be quite a challenge, and in any case, the direct approach of designing a machine for each operation would take us forever, since there are infinitely many operations in the series. Moreover, there are many other effectively computable numerical functions besides the ones in this series. When we return, in the chapter after next, to the task of showing various effectively computable numerical functions to be Turing computable, and thus accumulating evidence for Turing's thesis, a less direct approach will be adopted, and all the operations in the series that begins with addition and multiplication will be shown to be Turing computable in one go.

For the moment, we set aside the positive task of showing functions to be Turing computable and instead turn to examples of numerical functions of one argument that are Turing *un*computable (and so, if Turing's thesis is correct, effectively uncomputable).

Problems

3.1 Consider a tape containing a block of n strokes, followed by a space, followed by a block of m strokes, followed by a space, followed by a block of k strokes, and otherwise blank. Design a Turing machine that when started on the leftmost stroke will eventually halt, having neither printed nor erased anything . . .

(a) . . . on the leftmost stroke of the second block.

(b) . . . on the leftmost stroke of the third block.

3.2 Continuing the preceding problem, design a Turing machine that when started on the leftmost stroke will eventually halt, having neither printed nor erased anything . . .

(a) . . . on the rightmost stroke of the second block.

(b) . . . on the rightmost stroke of the third block.

3.3 Design a Turing machine that, starting with the tape as in the preceding problems, will eventually halt on the leftmost stroke on the tape, which is now to contain a block of n strokes, followed by a blank, followed by a block of $m + 1$ strokes, followed by a blank, followed by a block of k strokes.

3.4 Design a Turing machine that, starting with the tape as in the preceding problems, will eventually halt on the leftmost stroke on the tape, which is now to contain a block of n strokes, followed by a blank, followed by a block of $m - 1$ strokes, followed by a blank, followed by a block of k strokes.

3.5 Design a Turing machine to compute the function $\min(x, y)$ = the smaller of x and y.

3.6 Design a Turing machine to compute the function $\max(x, y)$ = the larger of x and y.

4

Uncomputability

In the preceding chapter we introduced the notion of Turing computability. *In the present short chapter we give examples of Turing-uncomputable functions: the* halting function *in section 4.1, and the* productivity function *in the optional section 4.2. If Turing's thesis is correct, these are actually examples of effectively uncomputable functions.*

4.1 The Halting Problem

There are too many functions from positive integers to positive integers for them all to be Turing computable. For on the one hand, as we have seen in problem 2.2, the set of all such functions is nonenumerable. And on the other hand, the set of Turing machines, and therefore of Turing-computable functions, is enumerable, since the representation of a Turing machine in the form of quadruples amounts to a representation of it by a finite string of symbols from a finite alphabet; and we have seen in Chapter 1 that the set of such strings is enumerable. These considerations show us that there must exist functions that are not Turing computable, but they do not provide an explicit example of such a function. To provide explicit examples is the task of this chapter. We begin simply by examining the argument just given in slow motion, with careful attention to details, so as to extract a specific example of a Turing-uncomputable function from it.

To begin with, we have suggested that we can enumerate the Turing-computable functions of one argument by enumerating the Turing machines, and that we can enumerate the Turing machines using their quadruple representations. As we turn to details, it will be convenient to modify the quadruple representation used so far somewhat. To indicate the nature of the modifications, consider the machine in Figure 3-9 in the preceding chapter. Its quadruple representation would be

$$q_1 S_0 R q_3,\ q_1 S_1 S_0 q_2,\ q_2 S_0 R q_1,\ q_3 S_0 S_1 q_4,\ q_3 S_1 S_0 q_2.$$

We have already been taking the lowest-numbered state q_1 to be the initial state. We now want to assume that the highest-numbered state is a halted state, for which there are no instructions and no quadruples. This is already the case in our example, and if it were not already so in some other example, we could make it so by adding one additional state.

We now also want to assume that for *every* state q_i except this highest-numbered halted state, and for *each* of the two symbols S_j we are allowing ourselves to use, namely $S_0 = B$ and $S_1 = 1$, there is a quadruple beginning $q_i S_j$. This is not so in our example as it stands, where there is no instruction for $q_2 S_1$. We have been interpreting the absence of an instruction for $q_i S_j$ as an instruction to halt, but the same effect could be achieved by giving an explicit instruction to keep the same symbol and then go to the highest-numbered state. When we modify the representation by adding this instruction, the representation becomes

$$q_1 S_0 R q_3, q_1 S_1 S_0 q_2, q_2 S_0 R q_1, q_2 S_1 S_1 q_4, q_3 S_0 S_1 q_4, q_3 S_1 S_0 q_2.$$

Now taking the quadruples beginning $q_1 S_0$, $q_1 S_1$, $q_2 S_0, \ldots$ in that order, as we have done, the first two symbols of each quadruple are predictable and therefore do not need to be written. So we may simply write

$$R q_3, S_0 q_2, R q_1, S_1 q_4, S_1 q_4, S_0 q_2.$$

Representing q_i by i, and S_j by $j + 1$ (so as to avoid 0), and L and R by 3 and 4, we can write still more simply

$$4, 3, 1, 2, 4, 1, 2, 4, 2, 4, 1, 2.$$

Thus the Turing machine can be completely represented by a finite sequence of positive integers—and even, if desired, by a single positive integer, say using the method of coding based on prime decomposition:

$$2^4 \cdot 3^3 \cdot 5 \cdot 7^2 \cdot 11^4 \cdot 13 \cdot 17^2 \cdot 19^4 \cdot 23^2 \cdot 29^4 \cdot 31 \cdot 37^2.$$

Not every positive integer will represent a Turing machine: whether a given positive integer does so or not depends on what the sequence of exponents in its prime decomposition is, and not every finite sequence represents a Turing machine. Those that do must have length some multiple $4n$ of 4, and have among their odd-numbered entries only numbers 1 to 4 (representing B, 1, L, R) and among their even-numbered entries only numbers 1 to $n + 1$ (representing the initial state q_1, various other states q_i, and the halted state q_{n+1}). But no matter: from the above representation we at least get a gappy listing of all Turing machines, in which each Turing machine is listed at least once, and on filling in the gaps we get a gapless list of all Turing machines, $M_1, M_2, M_3, \ldots$, and from this a similar list of all Turing-computable functions of one argument, $f_1, f_2, f_3, \ldots$, where f_i is the total or partial function computed by M_i.

To give a trivial example, consider the machine represented by (1, 1, 1, 1), or $2 \cdot 3 \cdot 5 \cdot 7 = 210$. Started scanning a stroke, it erases it, then leaves the resulting blank alone and remains in the same initial state, never going to the halted state, which would be state 2. Or consider the machine represented by (2, 1, 1, 1) or $2^2 \cdot 3 \cdot 5 \cdot 7 = 420$. Started scanning a stroke, it erases it, then prints it back again, then erases it, then prints it back again, and so on, again never halting. Or consider the machine represented by (1, 2, 1, 1), or $2 \cdot 3^2 \cdot 5 \cdot 7 = 630$. Started scanning a stroke, it erases it, then goes to the halted state 2 when it scans the resulting blank, which means halting in a nonstandard final configuration. A little thought shows that 210, 420, 630 are the smallest numbers that represent Turing machines, so the three machines just described will be M_1, M_2, M_3, and we have $f_1 = f_2 = f_3 =$ the empty function.

We have now indicated an explicit enumeration of the Turing-computable functions of one argument, obtained by enumerating the machines that compute them. The fact that such an enumeration is possible shows, as we remarked at the outset, that there must exist Turing-uncomputable functions of a single argument. The point of actually specifying one such enumeration is to be able to exhibit a particular such function. To do so, we define a *diagonal function d* as follows:

$$(1) \qquad d(n) = \begin{cases} 2 & \text{if } f_n(n) \text{ is defined and } = 1 \\ 1 & \text{otherwise.} \end{cases}$$

Now d is a perfectly genuine total function of one argument, but it is not Turing computable, that is, d is neither f_1 nor f_2 nor f_3, and so on. *Proof*: Suppose that d is one of the Turing computable functions—the mth, let us say. Then for each positive integer n, either $d(n)$ and $f_m(n)$ are both defined and equal, or neither of them is defined. But consider the case $n = m$:

$$(2) \qquad f_m(m) = d(m) = \begin{cases} 2 & \text{if } f_m(m) \text{ is defined and } = 1 \\ 1 & \text{otherwise.} \end{cases}$$

Then whether $f_m(m)$ is or is not defined, we have a contradiction: Either $f_m(m)$ is undefined, in which case (2) tells us that it is defined and has value 1; or $f_m(m)$ is defined and has a value $\neq 1$, in which case (2) tells us it has value 1; or $f_m(m)$ is defined and has value 1, in which case (2) tells us it has value 2. Since we have derived a contradiction from the assumption that d appears somewhere in the list $f_1, f_2, \ldots, f_m, \ldots$, we may conclude that the supposition is false. We have proved:

4.1 Theorem. The diagonal function d is not Turing computable.

According to Turing's thesis, since d is not Turing computable, d cannot be effectively computable. Why not? After all, although no Turing machine computes the function d, *we* were able compute at least its first few values. For since, as we have noted, $f_1 = f_2 = f_3 =$ the empty function we have $d(1) = d(2) = d(3) = 1$. And it may seem that we can actually compute $d(n)$ for any positive integer n—if we don't run out of time.

Certainly it is straightforward to discover which quadruples determine M_n for $n = 1, 2, 3$, and so on. (This is straightforward in principle, though eventually humanly infeasible in practice because the duration of the trivial calculations, for large n, exceeds the lifetime of a human being and, in all probability, the lifetime of the human race. But in our idealized notion of computability, we ignore the fact that human life is limited.)

And certainly it is perfectly routine to follow the operations of M_n, once the initial configuration has been specified; and if M_n does eventually halt, we must eventually get that information by following its operations. Thus if we start $M_n$with input n and it does halt with that input, then by following its operations until it halts, we can see whether it halts in nonstandard position, leaving $f_n(n)$ undefined, or halts in standard position with output $f_n(n) = 1$, or halts in standard position with output $f_n(n) \neq 1$. In the first or last cases, $d(n) = 1$, and in the middle case, $d(n) = 2$.

But there is yet another case where $d(n) = 1$; namely, the case where M_n never halts at all. If M_n is destined never to halt, given the initial configuration, can we find *that* out in a finite amount of time? This is the essential question: determining whether machine M_n, started scanning the leftmost of an unbroken block of n strokes on an otherwise blank tape, does or does not eventually halt.

Is *this* perfectly routine? Must there be some point in the routine process of following its operations at which it becomes clear that it will never halt? In simple cases this is so, as we saw in the cases of M_1, M_2, and M_3 above. But for the function d to be effectively computable, there would have to be a *uniform* mechanical procedure, applicable not just in these simple cases but also in more complicated cases, for discovering whether or not a given machine, started in a given configuration, will ever halt.

Thus consider the multiplier in Example 3.5. Its sequential representation would be a sequence of 68 numbers, each ≤ 18. It is routine to verify that it represents a Turing machine, and one can easily enough derive from it a flow chart like the one shown in Figure 3-7, but *without the annotations, and of course without the accompanying text*. Suppose one came upon such a sequence. It would be routine to check whether it represented a Turing machine and, if so, again to derive a flow chart *without annotations and accompanying text*. But is there a uniform method or mechanical routine that, in this and much more complicated cases, allows one to determine from inspecting the flow chart, *without any annotations or accompanying text*, whether the machine eventually halts, once the initial configuration has been specified?

If there is such a routine, Turing's thesis is erroneous: if Turing's thesis is correct, there can be no such routine. At present, several generations after the problem was first posed, no one has yet succeeded in describing any such routine—a fact that must be considered some kind of evidence in favor of the thesis.

Let us put the matter another way. A function closely related to d is the *halting function h* of two arguments. Here $h(m, n) = 1$ or 2 according as machine m, started with input n, eventually halts or not. If h were effectively computable, d would be effectively computable. For given n, we could first compute $h(n, n)$. If we got $h(n, n) = 2$, we would know that $d(n) = 1$. If we got $h(n, n) = 1$, we would know that we could safely start machine M_n in stardard initial configuration for input n, and that it would eventually halt. If it halted in nonstandard configuration, we would again have $d(n) = 1$. If it halted in standard final configuration giving an output $f_n(n)$, it would have $d(n) = 1$ or 2 according as $f_n(n) \neq 1$ or $= 1$.

This is an informal argument showing that if h were effectively computable, then d would be effectively computable. Since we have shown that d is not Turing computable, assuming Turing's thesis it follows that d is not effectively computable, and hence that h is not effectively computable, and so not Turing computable. It is also possible to prove rigorously, though we do not at this point have the apparatus needed to do so, that if h were Turing computable, then d would be Turing computable, and since we have shown that d is *not* Turing computable, this would show that h is not

Turing computable. Finally, it is possible to prove rigorously in another way, not involving d, that h is not Turing computable, and this we now do.

4.2 Theorem. The halting function h is not Turing computable.

Proof: By way of background we need two special Turing machines. The first is a *copying machine* C, which works as follows. Given a tape containing a block of n strokes, and otherwise blank, if the machine is started scanning the leftmost stroke on the tape, it will eventually halt with the tape containing two blocks of n strokes separated by a blank, and otherwise blank, with the machine scanning the leftmost stroke on the tape. Thus if the machine is started with

$$\ldots BBB1111BBB\ldots$$

it will halt with

$$\ldots BBB1111B1111BBB\ldots$$

eventually. We ask you to design such a machine in the problems at the end of this chapter (and give you a pretty broad hint how to do it at the end of the book).

The second is a *dithering machine D*. Started on the leftmost of a block of n strokes on an otherwise blank tape, D eventually halts if $n > 1$, but never halts if $n = 1$. Such a machine is described by the sequence

$$1, 3, 4, 2, 3, 1, 3, 3.$$

Started on a stroke in state 1, it moves right and goes into state 2. If it finds itself on a stroke, it moves back left and halts, but if it finds itself on a blank, it moves back left and goes into state 1, starting an endless back-and-forth cycle.

Now suppose we had a machine H that computed the function h. We could *combine* the machines C and H as follows: if the states of C are numbered 1 through p, and the states of H are numbered 1 through q, renumber the latter states $p + 1$ through $r = p + q$, and write these renumbered instructions after the instructions for C. Originally, C tells us to halt by telling us to go into state $p + 1$, but in the new combined instructions, going into state $p + 1$ means not halting, but beginning the operations of machine H. So the new combined instructions will have us first go through the operations of C, and then, when C would have halted, go through the operations of H. The result is thus a machine G that computes the function $g(n) = h(n, n)$.

We now combine *this* machine G with the dithering machine D, renumbering the states of the latter as $r + 1$ and $r + 2$, and writing its instructions after those for G. The result will be a machine M that goes through the operations of G and then the operations of D. Thus if machine number n halts when started on its own number, that is, if $h(n, n) = g(n) = 1$, then the machine M does *not* halt when started on that number n, whereas if machine number n does *not* halt when started on its own number, that is, if $h(n, n) = g(n) = 2$, then machine M *does* halt when started on n.

But of course there can be no such machine as M. For what would it do if started with input its own number m? It would halt if and only if machine number m, which is to say itself, does *not* halt when started with input the number m. This contradiction shows there can be no such machine as H.

The *halting problem* is to find an effective procedure that, given any Turing machine M, say represented by its number m, and given any number n, will enable us to determine whether or not that machine, given that number as input, ever halts. For the problem to be solvable by a Turing machine would require there to be a Turing machine that, given m and n as inputs, produces as its output the answer to the question whether machine number m with input n ever halts. Of course, a Turing machine of the kind we have been considering could not produce the output by printing the word ‘yes’ or ‘no’ on its tape, since we are considering machines that operate with just two symbols, the blank and the stroke. Rather, we take the affirmative answer to be presented by an output of 1 and the negative by an output of 2. With this understanding, the question whether the halting problem can be solved by a Turing machine amounts to the question whether the halting function h is Turing computable, and we have just seen in Theorem 4.2 that it is not. That theorem, accordingly, is often quoted in the form: ‘The halting problem is not solvable by a Turing machine.’ Assuming Turing’s thesis, it follows that it is not solvable at all.

Thus far we have two examples of functions that are not Turing computable—or problems that are not solvable by any Turing machine—and if Turing’s thesis is correct, these functions are not effectively computable. A further example is given in the next section. Though working through the example will provide increased familiarity with the potential of Turing machines that will be desirable when we come to the next chapter, and in any case the example is a beautiful one, still none of the material connected with this example is strictly speaking indispensable for any of our further work; and therefore we have starred the section in which it appears as optional.

4.2* The Productivity Function

Consider a k-state Turing machine, that is, a machine with k states (not counting the halted state). Start it with input k, that is, start it in its initial state on the leftmost of a block of k strokes on an otherwise blank tape. If the machine never halts, or halts in nonstandard position, give it a score of zero. If it halts in standard position with output n, that is, on the leftmost of a block of n strokes on an otherwise blank tape, give it a score of n. Now define $s(k) =$ the highest score achieved by any k-state Turing machine. This function can be shown to be Turing uncomputable.

We first show that if the function s were Turing computable, then so would be the function t given by $t(k) = s(k) + 1$. For supposing we have a machine that computes s, we can modify it as follows to get a machine, having one more state than the original machine, that computes t. Where the instructions for the original machine would have it halt, the instructions for the new machine will have it go into the new, additional state. In this new state, if the machine is scanning a stroke, it is to move one square to the left, remaining in the new state; while if it is scanning a blank, it is to print a stroke and halt. A little thought shows that a computation of the new machine will go through all the same steps as the old machine, except that, when the old machine would halt on the leftmost of a block of n strokes, the new machine will go through two more steps of computation (moving left and printing a stroke), leaving it halted on the leftmost of a block of $n + 1$ strokes. Thus its output will be one more than the output of the original machine, and if the original machine, for a given argument, computes the value of s, the new machine will compute the value of t.

Thus, to show that no Turing machine can compute s, it will now be enough to show that no Turing machine can compute t. And this is not hard to do. For suppose there were a machine computing t. It would have some number k of states (not counting the halted state). Started on the leftmost of a block of k strokes on an otherwise blank tape, it would halt on the leftmost of a block of $t(k)$ strokes on an otherwise blank tape. But then $t(k)$ would be the score of this particular k-state machine, and that is impossible, since $t(k) > s(k) =$ the highest score achieved by any k-state machine. Thus we have proved:

4.3 Proposition. The scoring function s is not Turing computable.

Let us have another look at the function s in the light of Turing's thesis. According to Turing's thesis, since s is not Turing computable, s cannot be effectively computable. Why not? After all there are (ignoring labelling) only finitely many quadruple representations or flow charts of k-place Turing machines for a given k. We could in principle start them all going in state 1 with input k and await developments. Some machines will halt at once, with score 0. As time passes, one or another of the other machines may halt; then we can check whether or not it has halted in standard position. If not, its score is 0; if so, its score can be determined simply by counting the number of strokes in a row on the tape. If this number is less than or equal to the score of some k-state machine that stopped earlier, we can ignore it. If it is greater than the score of any such machine, then it is the new record-holder. Some machines will run on forever, but since there are only finitely many machines, there will come a time when any machine that is ever going to halt has halted, and the record-holding machine at that time is a k-state machine of maximum score, and its score is equal to $s(k)$. Why doesn't this amount to an effective way of computing $s(k)$?

It would, *if* we had some method of effectively determining which machines are eventually going to halt. Without such a method, we cannot determine which of the machines that haven't halted yet at a given time are destined to halt at some later time, and which are destined never to halt at all, and so we cannot determine whether or not we have reached a time when all machines that are ever going to halt have halted. The procedure outlined in the preceding paragraph gives us a solution to the scoring problem, the problem of computing $s(n)$, only if we already have a solution to the *halting problem*, the problem of determining whether or not a given machine will, for given input, eventually halt. This is the flaw in the procedure.

There is a related Turing-uncomputable function that is even simpler to describe than s, called the *Rado* or *busy-beaver* function, which may be defined as follows. Consider a Turing machine started with the tape *blank* (rather than with input equal to the number of states of the machine, as in the scoring-function example). If the machine eventually halts, scanning the leftmost of an unbroken block of strokes on an otherwise blank tape, its *productivity* is said to be the length of that block. But if the machine never halts, or halts in some other configuration, its productivity is said to be 0. Now define $p(n)$ = the productivity of the most productive Turing machine having no more than n states (not counting the halted state).

This function also can be shown to be Turing uncomputable.

The facts needed about the function p can be conveniently set down in a series of examples. We state all the examples first, and then give our proofs, in case the reader wishes to look for a proof before consulting ours.

4.4 Example. $p(1) = 1$

4.5 Example. $p(n + 1) > p(n)$ for all n

4.6 Example. There is an i such that $p(n + i) \geq 2p(n)$ for all n

Proofs

Example 4.4. There are just 25 Turing machines with a single state q_1. Each may be represented by a flow chart in which there is just one node, and 0 or 1 or 2 arrows (from that node back to itself). Let us enumerate these flow charts.

Consider first the flow chart with no arrows at all. (There is just one.) The corresponding machine halts immediately with the tape still blank, and thus has productivity 0.

Consider next flow charts with two arrows, labelled 'B:—' and '1 : . . . ,' where each of '—' and '. . .' may be filled in with R or L or B or 1. There are $4 \cdot 4 = 16$ such flow charts, corresponding to the 4 ways of filling in '—' and the 4 ways of filling in '. . .'. Each such flow chart corresponds to a machine that never halts, and thus has productivity 0. The machine never halts because no matter what symbol it is scanning, there is always an instruction for it to follow, even if it is an instruction like 'print a blank on the (already blank) square you are scanning, and stay in the state you are in'.

Consider flow charts with one arrow. There are four of them where the arrow is labelled '1: . . . '. These all halt immediately, since the machine is started on a blank, and there is no instruction what to do when scanning a blank. So again the productivity is 0.

Finally, consider flow charts with one arrow labelled 'B:—'. Again there are four of them. Three of them have productivity 0: the one 'B:B', which stays put, and the two labelled 'B:R' and 'B:L', which move endlessly down the tape in one direction or the other (*touring* machines). The one labelled 'B:1' prints a stroke and then halts, and thus has productivity 1. Since there is thus a 1-state machine whose productivity is 1, and every other 1-state machine has productivity 0, the *most* productive 1-state machine has productivity 1.

Example 4.5. Choose any of the most productive n-state machines, and add one more state, as in Figure 4-1.

The result is an $(n + 1)$-state machine of productivity $n + 1$. There may be $(n + 1)$-state machines of even greater productivity than this, but we have established that

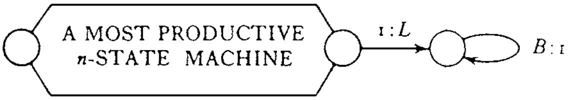


Figure 4-1. Increasing productivity by 1.

the productivity of the most productive $(n+1)$-state machines is *at least* greater by 1 than the productivity of the most productive n-state machine.

Example 4.6. We can take $i = 11$. To see this, plug together an n-state machine for writing a block of n strokes (Example 3.1) with a 12-state machine for doubling the length of a row of strokes (Example 3.2). Here 'plugging together' means superimposing the starting node of one machine on the halting node of the other: *identifying* the two nodes. [Number the states of the first machine 1 through n, and those of the second machine $(n-1)+1$ through $(n-1)+12$, which is to say n through $n+11$. This is the same process we described in terms of lists of instructions rather than flow charts in our proof of Theorem 4.2.] The result is shown in Figure 4-2.

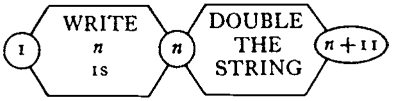


Figure 4-2. Doubling productivity.

The result is an $(n+11)$-state machine with productivity $2n$. Since there may well be $(n+11)$-state machines with even greater productivity, we are not entitled to conclude that the most productive $(n+11)$-state machine has productivity exactly $2n$, but we are entitled to conclude that the most productive $(n+11)$-state machine has productivity *at least* $2n$.

So much for the pieces. Now let us put them together into a proof that the function p is not Turing computable. The proof will be by *reductio ad absurdum*: we deduce an absurd conclusion from the supposition that there is a Turing machine computing p.

The first thing we note is that if there is such a machine, call it *BB*, and the number of its states is j, then we have

$$(1) \qquad p(n+2j) \geq p(p(n))$$

for any n. For given a j-state machine *BB* computing p, we can plug together an n-state machine writing a row of n strokes with two replicas of *BB* as in Figure 4-3.

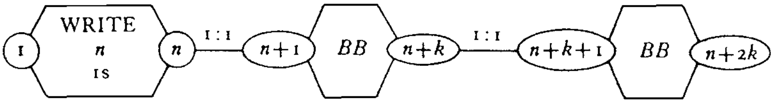


Figure 4-3. Boosting productivity using the hypothetical machine *BB*.

The result is an $(n+2j)$-state machine of productivity $p(p(n))$. Now from Example 4.5 above it follows that if $a < b$, then $p(a) < p(b)$. Turning this around, if $p(b) \leq p(a)$, we must have $b \leq a$. Applying this observation to (1), we have

$$n + 2j \geq p(n) \tag{2}$$

for any n. Letting i be as in Example 4.6 above, we have

$$p(m + i) \geq 2m \tag{3}$$

for any m. But applying (2) with $n = m + i$, we have

$$m + i + 2j \geq p(m + i) \tag{4}$$

for any m. Combining (3) and (4), we have

$$m + i + 2j \geq 2m \tag{5}$$

for any m. Setting $k = i + 2j$, we have

$$m + k \geq 2m \tag{6}$$

for any m. But this is absurd, since clearly (6) fails for any $m > k$. We have proved:

4.7 Theorem. The productivity function p is Turing uncomputable.

Problems

4.1 Is there a Turing machine that, started anywhere on the tape, will eventually halt if and only if the tape originally was *not* completely blank? If so, sketch the design of such a machine; if not, briefly explain why not.

4.2 Is there a Turing machine that, started anywhere on the tape, will eventually halt if and only if the tape originally was completely blank? If so, sketch the design of such a machine; if not, briefly explain why not.

4.3 Design a copying machine of the kind described at the beginning of the proof of theorem 4.2.

4.4 Show that if a two-place function g is Turing computable, then so is the one-place function f given by $f(x) = g(x, x)$. For instance, since the multiplication function $g(x, y) = xy$ is Turing computable, so is the square function $f(x) = x^2$.

4.5 A *universal* Turing machine is a Turing machine U such that for any other Turing machine M_n and any x, the value of the two-place function computed by U for arguments n and x is the same as the value of the one-place function computed by M_n for argument x. Show that if Turing's thesis is correct, then a universal Turing machine must exist.

5

Abacus Computability

Showing that a function is Turing computable directly, by giving a table or flow chart for a Turing machine computing the function, is rather laborious, and in the preceding chapters we did not get beyond showing that addition and multiplication and a few other functions are Turing computable. In this chapter we provide a less direct way of showing functions to be Turing computable. In section 5.1 we introduce an ostensibly more flexible kind of idealized machine, an abacus machine*, or simply an* abacus*. In section 5.2 we show that despite the ostensible greater flexibility of these machines, in fact anything that can be computed on an abacus can be computed on a Turing machine. In section 5.3 we use the flexibility of these machines to show that a large class of functions, including not only addition and multiplication, but exponentiation and many other functions, are computable on a abacus. In the next chapter functions of this class will be called* recursive*, so what will have been proved by the end of this chapter is that all recursive functions are Turing computable.*

5.1 Abacus Machines

We have shown addition and multiplication to be Turing-computable functions, but not much beyond that. Actually, the situation is even a bit worse. It seemed appropriate, when considering Turing machines, to define Turing computability for functions on positive integers (excluding zero), but in fact it is customary in work on other approaches to computability to consider functions on natural numbers (including zero). If we are to compare the Turing approach with others, we must adapt our definition of Turing computability to apply to natural numbers, as can be accomplished (at the cost of some slight artificiality) by the expedient of letting the number n be represented by a string of $n + 1$ strokes, so that a single stroke now represents zero, two strokes represent one, and so on. But with this change, the adder we presented in the last chapter actually computes $m + n + 1$, rather than $m + n$, and would need to be modified to compute the standard addition function; and similarly for the multiplier. The modifications are not terribly difficult to carry out, but they still leave us with only a very few examples of interesting effectively computable functions that have been shown to be Turing computable. In this chapter we greatly enlarge the number of examples, but we do *not* do so directly, by giving tables or flow charts for the relevant Turing machines. Instead, we do so indirectly, by way of another kind of idealized machine.

Historically, the notion of Turing computability was developed before the age of high-speed digital computers, and in fact, the theory of Turing computability formed a not insignificant part of the theoretical background for the development of such computers. The kinds of computers that are ordinary today are in one respect more flexible than Turing machines in that they have *random-access storage*. A *Lambek* or *abacus machine* or simply *abacus* will be an idealized version of computer with this 'ordinary' feature. In contrast to a Turing machine, which stores information symbol by symbol on squares of a one-dimensional tape along which it can move a single step at a time, a machine of the seemingly more powerful 'ordinary' sort has access to an unlimited number of *registers* R_0, R_1, R_2, . . . , in each of which can be written numbers of arbitrary size. Moreover, this sort of machine can go directly to register R_n without inching its way, square by square, along the tape. That is, each register has its own *address* (for register R_n it might be just the number n) which allows the machine to carry out such instructions as

put the sum of the numbers in registers R_m and R_n into register R_p

which we abbreviate

$$[m] + [n] \to p.$$

In general, $[n]$ is the number in register R_n, and the number at the right of an arrow identifies the register in which the result of the operation at the left of the arrow is to be stored. When working with such machines, it is natural to consider functions on the natural numbers (including zero), and not just the positive integers (excluding zero). Thus, the number $[n]$ in register R_n at a given time may well be zero: the register may be *empty*.

It should be noted that our 'ordinary' sort of computing machine is really quite extraordinary in one respect: although real digital computing machines often have random-access storage, there is always a finite upper limit on the size of the numbers that can be stored; for example, a real machine might have the ability to store any of the numbers 0, 1, . . . , 10 000 000 in each of its registers, but no number greater than ten million. Thus, it is entirely possible that a function that is computable in principle by one of our idealized machines is not computable in practice by any real machine, simply because, for certain arguments, the computation would require more capacious registers than any real machine possesses. (Indeed, addition is a case in point: there is no finite bound on the sizes of the numbers one might think of adding, and hence no finite bound on the size of the registers needed for the arguments and the sum.) But this is in line with our objective of abstracting from technological limitations so as to arrive at a notion of computability that is not too narrow. We seek to show that certain functions are uncomputable in an absolute sense: uncomputable even by our idealized machines, and therefore uncomputable by any past, present, or future real machine.

In order to avoid discussion of electronic or mechanical details, we may imagine the abacus machine in crude, Stone Age terms. Each register may be thought of as a roomy, numbered box capable of holding any number of stones: none or one or two or . . . , so that $[n]$ will be the number of stones in box number n. The 'machine' can be thought of as operated by a little man who is capable of carrying out two sorts of operations: adding a stone to the box of a specified number, and removing a stone from the box of a specified number, if there are any stones there to be removed.

The table for a Turing machine is in effect a list of numbered instructions, where ‘attending to instruction q’ is called ‘being in state q’. The instructions all have the following form:

$$(q)\begin{cases}\text{if you are scanning a blank} & \text{then perform action } a \text{ and go to } r\\ \text{if you are scanning a stroke} & \text{then perform action } b \text{ and go to } s.\end{cases}$$

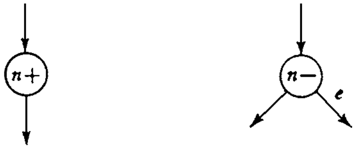
Here each of the actions is one of the following four options: erase (put a blank in the scanned square), print (put a stroke in the scanned square), move left, move right. It is permitted that one or both of r or s should be q, so ‘go to r’ or ‘go to s’ amounts to ‘remain with q’.

Turing machines can also be represented by flow charts, in which the states or instructions do not have to be numbered. An abacus machine program could also be represented in a table of numbered instructions. These would each be of one or the other of the following two forms:

$$(q)\quad \text{add one to box } m \text{ and go to } r$$

$$(q)\begin{cases}\text{if box } m \text{ is not empty} & \text{then subtract one from box } m \text{ and go to } r\\ \text{if box } m \text{ is empty} & \text{then go to } s.\end{cases}$$

But in practice we are going to be working throughout with a flow-chart representation. In this representation, the elementary operations will be symbolized as in Figure 5-1.



Add a stone to the pile in box n

If box n is empty, come out on the arrow marked ‘e’, but if not, remove a stone and come out on the other arrow

Figure 5-1. Elementary operations in abacus machines.

Flow charts can be built up as in the following examples.

5.1 Example (Emptying box n). Emptying the box of a specified number n can be accomplished with a single instruction as follows:

$$(1)\qquad\begin{cases}\text{if box } n \text{ is not empty} & \text{then subtract 1 from box } n \text{ and stay with 1}\\ \text{if box } n \text{ is empty} & \text{then halt.}\end{cases}$$

The corresponding flow chart is indicated in Figure 5-2.

In the figure, halting is indicated by an arrow leading nowhere. The block diagram also shown in Figure 5-2 summarizes what the program shown in the flow chart accomplishes,

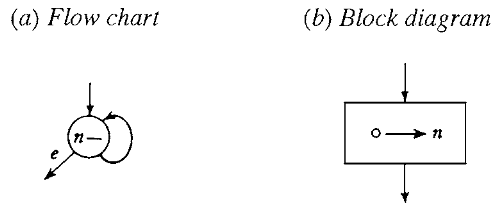

(*a*) *Flow chart* (*b*) *Block diagram*

Figure 5-2. Emptying a box.

without indicating *how* it is accomplished. Such summaries are useful in showing how more complicated programs can be put together out of simpler ones.

5.2 Example (Emptying box m into box n). The program is indicated in Figure 5-3.

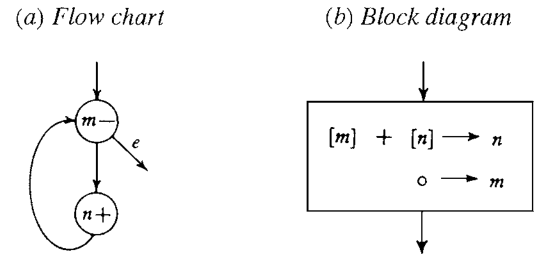

(*a*) *Flow chart* (*b*) *Block diagram*

Figure 5-3. Emptying one box into another.

The figure is intended for the case $m \neq n$. (If $m = n$, the program halts—exits on the e arrow—either at once or never, according as the box is empty or not originally.) In future we assume, unless the contrary possibility is explicitly allowed, that when we write of boxes m, n, p, and so on, distinct letters represent distinct boxes.

When as intended $m \neq n$, the *effect* of the program is the same as that of carrying stones from box m to box n until box m is empty, but there is no way of instructing the machine or its operator to do exactly that. What the operator *can* do is ($m-$) take stones out of box m, one at a time, and throw them on the ground (or take them to wherever unused stones are stored), and then ($n+$) pick stones up off the ground (or take them from wherever unused stones are stored) and put them, one at a time, into box n. There is no assurance that the stones put into box n are the very same stones that were taken out of box m, but we need no such assurance in order to be assured of the desired *effect* as described in the block diagram, namely,

$[m] + [n] \to n$: the number of stones in box n after this move equals
the sum of the numbers in m and in n before the move

and then

$0 \to m$: the number of stones in box m after this move is 0.

5.3 Example (Adding box m to box n, without loss from m). To accomplish this we must make use of an *auxiliary* register p, which must be empty to begin with (and will be empty again at the end as well). Then the program is as indicated in Figure 5-4.

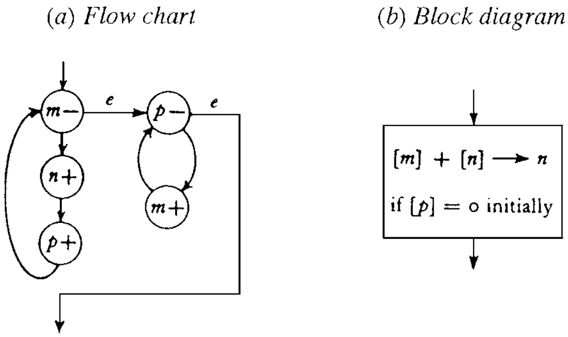

(*a*) *Flow chart* (*b*) *Block diagram*

Figure 5-4. Addition.

In case no assumption is made about the contents of register p at the beginning, the operation accomplished by this program is the following:

$$\begin{aligned}[m]+[n] &\to n\\ [m]+[p] &\to m\\ 0 &\to p.\end{aligned}$$

Here, as always, the vertical order represents a *sequence of moves*, from top to bottom. Thus, p is emptied *after* the other two moves are made. (The order of the first two moves is arbitrary: The effect would be the same if their order were reversed.)

5.4 Example (Multiplication). The numbers to be multiplied are in distinct boxes m_1 and m_2; two other boxes, n and p, are empty to begin with. The product appears in box n. The program is indicated in Figure 5-5.

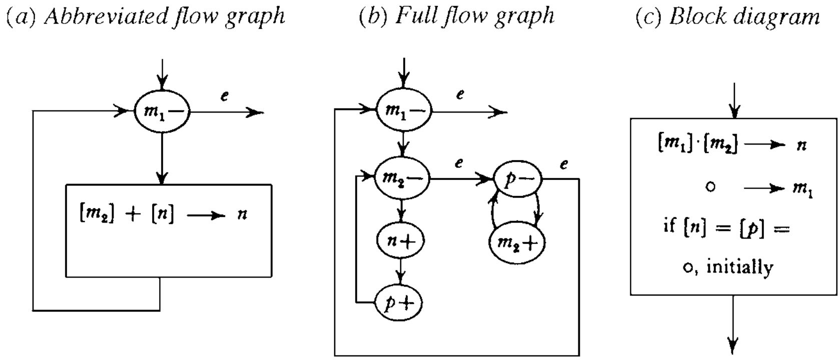

(*a*) *Abbreviated flow graph* (*b*) *Full flow graph* (*c*) *Block diagram*

Figure 5-5. Multiplication.

Instead of constructing a flow chart *de novo*, we use the block diagram of the preceding example a shorthand for the flow chart of that example. It is then straightforward to draw the full flow chart, as in Figure 5-5(b), where the m of the preceding example is changed to m_2. The procedure is to dump $[m_2]$ stones repeatedly into box n, using box m_1 as a counter:

We remove a stone from box m_1 before each dumping operation, so that when box m_1 is empty we have

$$[m_2] + [m_2] + \cdots + [m_2] \qquad ([m_1]\text{ summands})$$

stones in box n.

5.5 Example (Exponentiation). Just as multiplication is repeated addition, so exponentiation is repeated multiplication. The program is perfectly straightforward, once we arrange the multiplication program of the preceding example to have $[m_2] \cdot [n] \rightarrow n$. How that is to be accomplished is shown in Figure 5-6.

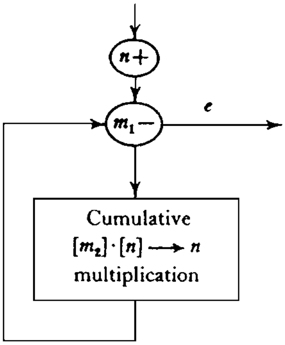


Figure 5-6. Exponentiation.

The cumulative multiplication indicated in this abbreviated flow chart is carried out in two steps. First, use a program like Example 5.4 with a new auxiliary:

$$\begin{aligned}[n] \cdot [m_2] &\rightarrow q\\ 0 &\rightarrow n.\end{aligned}$$

Second, use a program like Example 5.2:

$$\begin{aligned}[q] + [n] &\rightarrow n\\ 0 &\rightarrow q.\end{aligned}$$

The result gives $[n] \cdot [m_2] \rightarrow n$. Provided the boxes n, p, and q are empty initially, the program for exponentiation has the effect

$$\begin{aligned}[m_2]^{[m_1]} &\rightarrow n\\ 0 &\rightarrow m_1\end{aligned}$$

in strict analogy to the program for multiplication. (Compare the diagrams in the preceding example and in this one.)

Structurally, the abbreviated flow charts for multiplication and exponentiation differ only in that for exponentiation we need to put a single stone in box n at the beginning. If $[m_1] = 0$ we have $n = 1$ when the program terminates (as it will at once, without going through the multiplication routine). This corresponds to the convention that $x^0 = 1$ for any natural number x. But if $[m_1]$ is positive, $[n]$ will finally be a product of $[m_1]$ factors $[m_2]$, corresponding to repeated application of the rule $x^{y+1} = x \cdot x^y$, which is implemented by means of cumulative multiplication, using box m_1 as a counter.

It should now be clear that the initial restriction to two elementary sorts of acts, $n+$ and $n-$, does not prevent us from computing fairly complex functions, including all the functions in the series that begins *sum*, *product*, *power*, . . . , and where the $n + 1$st member is obtained by iterating the nth member. This is considerably further than we got with Turing machines in the preceding chapters.

5.2 Simulating Abacus Machines by Turing Machines

We now show that, despite the ostensible greater flexibility of abacus machines, all abacus-computable functions are Turing computable. Before we can describe a method for transforming abacus flow charts into equivalent Turing-machine flow charts, we need to standardize certain features of abacus computations, as we did earlier for Turing computations with our official definition of Turing computability. We must know where to place the arguments, initially, and where to look, finally, for values. The following conventions will do as well as any, for a function f of r arguments $x_1, \ldots, x_r$:

(a) Initially, the arguments are the numbers of stones in the first r boxes, and all other boxes to be used in the computation are empty. Thus, $x_1 = [1], \ldots, x_r = [r]$, $0 = [r+1] = [r+2] = \cdots$.

(b) Finally, the value of the function is the number of stones is some previously specified box n (which may but need not be one of the first r). Thus, $f(x_1, \ldots, x_r) = [n]$ when the computation halts, that is, when we come to an arrow in the flow chart that terminates in no node.

(c) If the computation never halts, $f(x_1, \ldots, x_r)$ is undefined.

The computation routines for addition, multiplication, and exponentiation in the preceding section were essentially in this form, with $r = 2$ in each case. They were formulated in a general way, so as to leave open the question of just which boxes are to contain the arguments and value. For example, in the adder we only specified that the arguments are to be stored in distinct boxes numbered m and n, that the sum will be found in box x, and that a third box, numbered p and initially empty, will be used as an auxiliary in the course of the computation. But now we must specify m, n, and p subject to the restriction that m and n must be 1 and 2, and p must be some number greater than 2. Then we might settle on $n = 1$, $m = 2$, $p = 3$, to get a particular program for addition in the standard format, as in Figure 5-7.

The standard format associates a definite function from natural numbers to natural numbers with each abacus, once we specify the number r of arguments and the number n of the box in which the values will appear. Similarly, the standard format for Turing-machine computations associates a definite function from natural numbers to natural numbers (originally, from positive integers to positive integers, but we have modified that above) with each Turing machine, once we specify the number r of arguments. Observe that once we have specified the chart of an abacus

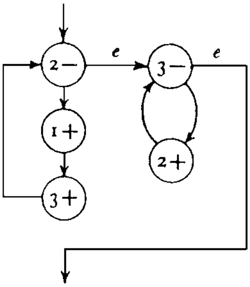


Figure 5-7. Addition in standard format.

machine A in standard form, then for each register n that we might specify as holding the result of the computation there are infinitely many functions A_n^r that we have specified as computed by the abacus: one function for each possible number r of arguments. Thus if A is determined by the simplest chart for addition, as in Example 5.2, with $n = 1$ and $m = 2$, we have

$$A_1^2(x, y) = x + y$$

for all natural numbers x and y, but we also have the identity function $A_1^1(x) = x$ of one argument, and for three or more arguments we have $A_r^1(x_1, \ldots, x_r) = x_1 + x_2$. Indeed, for $r = 0$ arguments we may think of A as computing a 'function' of a sort, namely, the number $A_1^0 = 0$ of strokes in box 1 when the computation halts, having been started with all boxes ('except the first r') empty. Of course, the case is entirely parallel for Turing machines, each of which computes a function of r arguments in standard format for each $r = 0, 1, 2, \ldots$, the value for 0 being what we called the productivity of the machine in the preceding chapter.

Having settled on standard formats for the two kinds of computation, we can turn to the problem of designing a method for converting the flow chart of an abacus A_n, with n designated as the box in which the values will appear, into the chart of a Turing machine that computes the same functions: for each r, the Turing machine will compute the same function A_n^r of r arguments that the abacus computes. Our method will specify a Turing-machine flow chart that is to replace each node of type $n+$ with its exiting arrow (as on the left in Figure 5-1, but without the entering arrow) in the abacus flow chart; a Turing-machine flow chart that is to replace each node of type $n-$ with its two exiting arrows (as on the right in Figure 5-1, again without the entering arrow); and a *mop-up* Turing-machine flow chart that, at the end, makes the machine erase all but the nth block of strokes on the tape and halt, scanning the leftmost of the remaining strokes.

It is important to be clear about the relationship between boxes of the abacus and corresponding parts of the Turing machine's tape. For example, in computing $A_n^4(0, 2, 1, 0)$, the initial tape and box configurations would be as shown in Figure 5-8. Boxes containing one or two or . . . stones are represented by blocks of two or three or . . . strokes on the tape. Single blanks separate portions of the tape corresponding to successive boxes. Empty boxes are always represented by single squares, which may be blank (as with R_5, R_6, R_7, . . . in the figure) or contain a single 1 (as with R_1 and

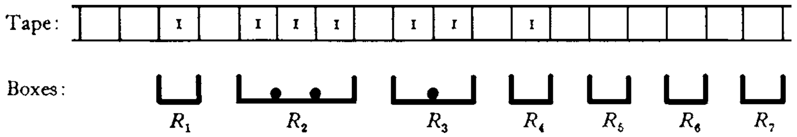
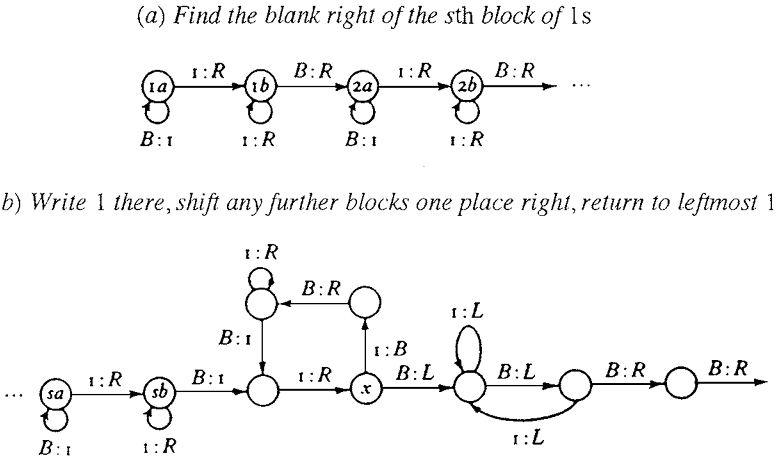


Figure 5-8. Correspondence between boxes and tape.

R_4 in the figure). The 1 is mandatory if there are any strokes further to the right on the tape, and is mandatory initially for empty argument boxes. The blank is mandatory initially for R_{r+1}, R_{r+2}, ... Then at any stage of the computation we can be sure that when in moving to the right or left we encounter two successive blanks, there are no further strokes to be found anywhere to the right or left (as the case may be) on the tape. The exact portion of the tape that represents a box will wax and wane with the contents of that box as the execution of the program progresses, and will shift to the right or left on the tape as stones are added to or removed from lower-numbered boxes.

The *first step* in our method for converting abacus flow charts into equivalent Turing-machine flow charts can now be specified: replace each $s+$ node (consisting of a node marked $s+$ and the arrow leading from it) by a copy of the $s+$ flow chart shown in Figure 5-9.


(*a*) *Find the blank right of the s*th *block of* 1s

(*b*) *Write* 1 *there, shift any further blocks one place right, return to leftmost* 1

Figure 5-9. The $s+$ flow chart.

The first $2(s-1)$ nodes of the $s+$ chart simply take the Turing machine across the first $s-1$ blocks of strokes. In the course of seeking the sth block, the machine substitutes the 1-representation for the B-representation of any empty boxes encountered along the way.

When it enters the node *sa*, the Turing machine has arrived at the sth block. Then again substitutes the 1-representation for the B-representation of that box, if that box is empty. On leaving node *sb*, the machine writes a stroke, moves 1 square right, and does one thing or another (node x) depending on whether it is then scanning a blank or a stroke.

If it is scanning a blank, there can be no more strokes to the right, and it therefore returns to standard position. But if it is scanning a stroke at that point, it has more work to do before returning to standard position, for there are more blocks of strokes to be dealt with, to the right on the tape. These must be shifted one square rightwards, by erasing the first 1 in each block and filling the blank to the block's right with a stroke—continuing this routine until it finds a blank to the right of the last blank it has replaced by a stroke. At that point there can be no further strokes to the right, and the machine returns to standard position.

Note that node 1*a* is needed in case the number *r* of arguments is 0: in case the 'function' that the abacus computes is a number A^0_n. Note, too, that the first $s - 1$ pairs of nodes (with their efferent arrows) are identical, while the last pair is different only in that the arrow from node *sb* to the right is labelled *B*:1 instead of *B*:*R*. What the general *s*+ flow chart looks like in the case $s = 1$ is shown in Figure 5-10.

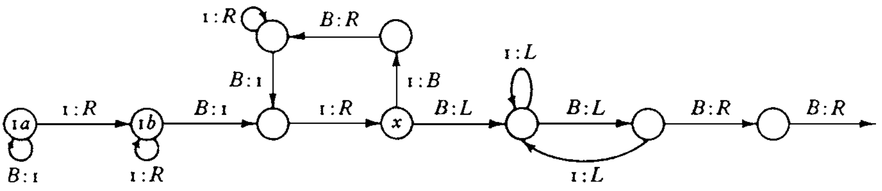


Figure 5-10. The special case $s = 1$.

The *second step* in our method of converting abacus flow charts into equivalent Turing machine flow charts can now be specified: replace each *s*− node (with the two arrows leading from it) by a copy of an *s*− flow chart having the general pattern shown in Figure 5-11.

Readers may wish to try to fill in the details of the design for themselves, as an exercise. (Our design will be given later.) When the first and second steps of the method have been carried out, the abacus flow chart will have been converted into something that is not quite the flow chart of a Turing machine that computes the same function that the abacus does. The chart will (probably) fall short in two respects, one major and one minor. The minor respect is that if the abacus ever halts, there must be one or more 'loose' arrows in the chart: arrows that terminate in no node. This is simply because that is how halting is represented in abacus flow charts: by an arrow leading nowhere. But in Turing-machine flow charts, halting is represented in a different way, by a node with no arrows leading from it. The major respect is that in computing $A^r_n(x_1, \ldots, x_r)$ the Turing machine would halt scanning the leftmost 1 on the tape, *but the value of the function would be represented by the nth block of* strokes *on the tape*. Even if $n = 1$, we cannot depend on there being no strokes on the tape after the first block, so our method requires one more step.

The *third step*: after completing the first two steps, redraw all loose arrows so they terminate in the input node of a *mop-up chart*, which makes the machine (which will be scanning the leftmost 1 on the tape at the beginning of this routine) erase all but the first block of strokes if $n = 1$, and halt scanning the leftmost of the remaining strokes. But if $n \neq 1$, it erases everything on the tape except for both the leftmost

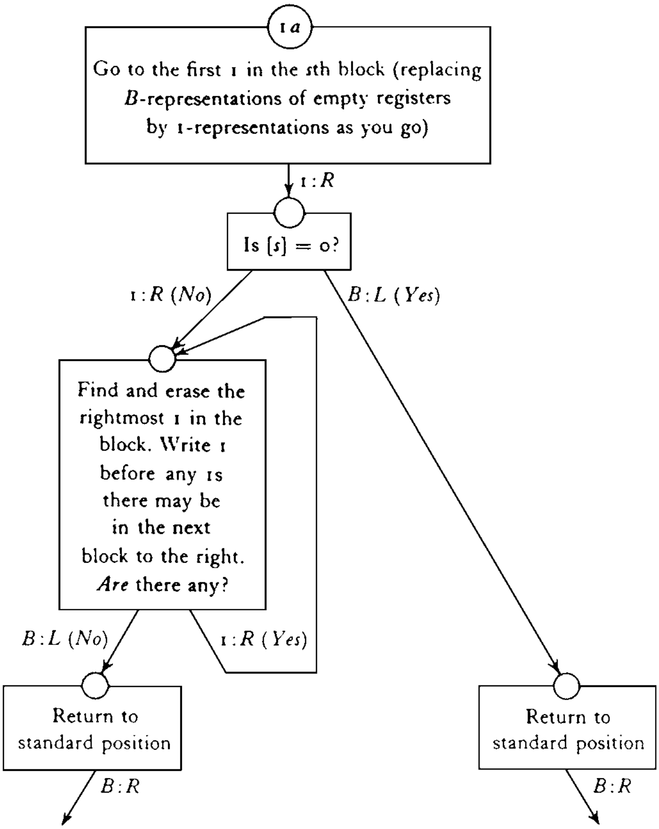

1a
Go to the first 1 in the sth block (replacing B-representations of empty registers by 1-representations as you go)
1:R
Is [s] = 0?
1:R (No)
B:L (Yes)
Find and erase the rightmost 1 in the block. Write 1 before any 1s there may be in the next block to the right. Are there any?
B:L (No)
1:R (Yes)
Return to standard position
B:R
Return to standard position
B:R

Figure 5-11. Abbreviated *s*− flow chart.

1 on the tape and the *n*th block, repositions all strokes but the rightmost in the *n*th block immediately to the right of the leftmost 1, erases the rightmost 1, and then halts scanning the leftmost 1. In both cases, the effect is to place the leftmost 1 in the block representing the value just where the leftmost 1 was initially. Again readers may wish to try to fill in the details of the design for themselves, as an exercise. (Our design will be given shortly.)

The proof that all abacus-computable functions are Turing computable is now finished, except for the two steps that we have invited readers to try as exercises. For the sake of completeness, we now present our own solutions to these exercises: our own designs for the second and third stages of the construction reducing an abacus computation to a Turing computation.

For the second stage, we describe what goes into the boxes in Figure 5-11. The top block of the diagram contains a chart identical with the material from node 1*a* to *sa* (inclusive) of the *s*+ flow chart. The arrow labelled 1:*R* from the bottom of this block corresponds to the one that goes right from node *sa* in the *s*+ flow chart.

The 'Is [*s*] = 0?' box contains nothing but the shafts of the two emergent arrows: They originate in the node shown at the top of that block.

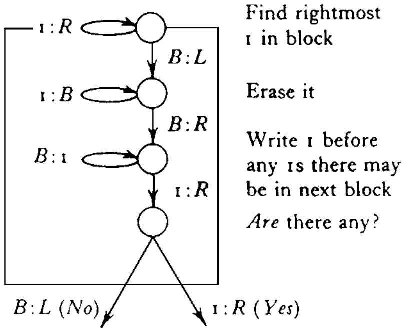


Figure 5-12. Detail of the $s-$ flow chart.

The 'Return to standard position' blocks contain replicas of the material to the right of node x in the $s+$ chart: The B:L arrows entering those boxes correspond to the B:L arrow from node x.

The only novelty is in the remaining block: 'Find and erase the . . . ' That block contains the chart shown in Figure 5-12.

For the third stage, the mop-up chart, for $n \neq 1$, is shown in Figure 5-13.

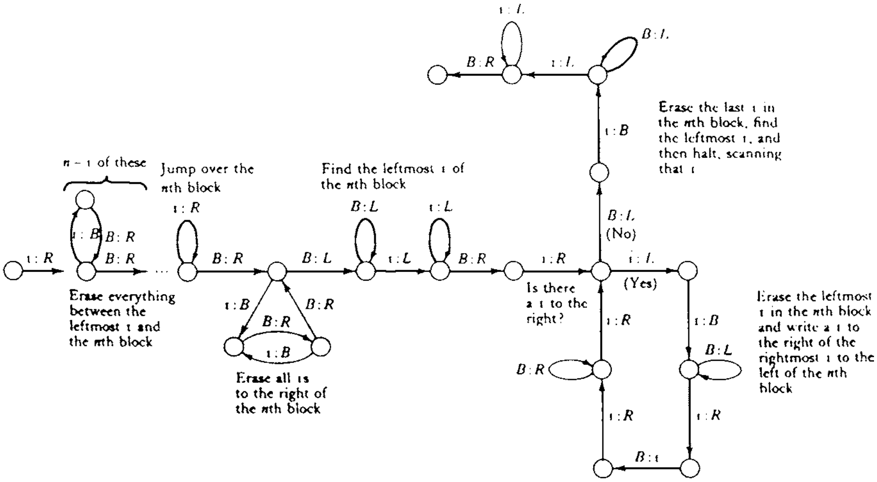

1 : L
B : R
1 : L
B : L
Erase the last 1 in the nth block, find the leftmost 1, and then halt, scanning that 1
1 : B
n − 1 of these
Jump over the nth block
Find the leftmost 1 of the nth block
1 : R
B : L
1 : L
1 : B
B : R
1 : R
B : R
B : R
B : L
1 : L
B : R
1 : R
B : L (No)
1 : L
(Yes)
Erase everything between the leftmost 1 and the nth block
1 : B
B : R
B : R
1 : B
Erase all 1s to the right of the nth block
Is there a 1 to the right?
1 : R
B : R
1 : R
1 : B
B : L
1 : R
B : 1
Erase the leftmost 1 in the nth block and write a 1 to the right of the rightmost 1 to the left of the nth block

Figure 5-13. Mop-up chart.

We have proved:

5.6 Theorem. Every abacus-computable function is Turing computable.

We know from the preceding chapter some examples of functions that are *not* Turing computable. By the foregoing theorem, these functions are also not abacus computable. It is also possible to prove directly the existence of functions that are not abacus computable, by arguments parallel to those used for Turing computability in the preceding chapter.

5.3 The Scope of Abacus Computability

We now turn from showing that particular functions are abacus computable to showing that certain processes for defining new functions from old, when applied to old abacus-computable functions, produce new abacus-computable functions. (These processes will be explained and examined in more detail in the next chapter, and readers may wish to defer reading this section until after that chapter.)

Now we initially indicated that to compute a function of r arguments on an abacus, we must specify r registers or boxes in which the arguments are to be stored initially (represented by piles of rocks) and we must specify a register or box in which the value of the function is to appear (represented by a pile of rocks) at the end of the computation. To facilitate comparison with computations by Turing machines in standard form, we then insisted that the input or arguments were to be placed in the first r registers, but left it open in which register n the output or value would appear: it was not necessary to be more specific, because the simulation of the operations of an abacus by a Turing machine could be carried out wherever we let the output appear. For the purposes of this section, we are therefore free now to insist that the output register n, which we have heretofore left unspecified, be specifically register $r + 1$. We also wish to insist that at the end of the computation the original arguments should be back in registers 1 through r. In the examples considered earlier this last condition was not met, but those examples are easily modified to meet it. We give some further, trivial examples here, where all our specifications are exactly met.

5.7 Example (Zero, successor, identity). First consider the zero function z, the one-place function that takes the value 0 for all arguments. It is computed by the vacuous program: box 2 is empty anyway.

Next consider the successor function s, the one-place function that takes any natural number x to the next larger natural number $x + 1$. It is computed by modifying the program in Example 5.3, as shown in Figure 5-14.

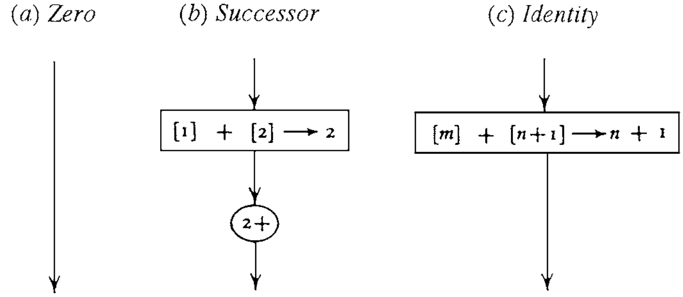


Figure 5-14. Three basic functions.

Initially and finally, $[1] = x$; initially $[2] = 0$; finally, $[2] = s(x)$. Finally consider identity function id^m_n, the n-place function whose value for n arguments $x_1, \ldots, x_n$ is the mth one among them, x_m. It is computed by the program of the same Example 5.3. Initially and finally, $[1] = x_1, \ldots, [n] = x_n$; initially, $[n + 1] = 0$; finally $[n + 1] = x_m$.

Three different processes for defining new functions from old can be used to expand our initial list of examples. A first process is *composition*, also called *substitution*. Suppose we have two 3-place functions g_1 and g_2, and a 2-place function f. The function h obtained from them by composition is the 3-place function given by

$$h(x_1, x_2, x_3) = f(g_1(x_1, x_2, x_3), g_2(x_1, x_2, x_3)).$$

Suppose g_1 and g_2 and f are abacus computable according to our specifications, and we are given programs for them.

$$\begin{array}{c}\downarrow\\ \boxed{f([1],[2]) \to 3}\\ \downarrow\end{array} \qquad \begin{array}{c}\downarrow\\ \boxed{g_1([1],[2],[3]) \to 4}\\ \downarrow\end{array} \qquad \begin{array}{c}\downarrow\\ \boxed{g_2([1],[2],[3]) \to 4}\\ \downarrow\end{array}$$

We want to find a program for h, to show it is abacus computable:

$$\begin{array}{c}\downarrow\\ \boxed{h([1],[2],[3]) \to 4}\\ \downarrow\end{array}.$$

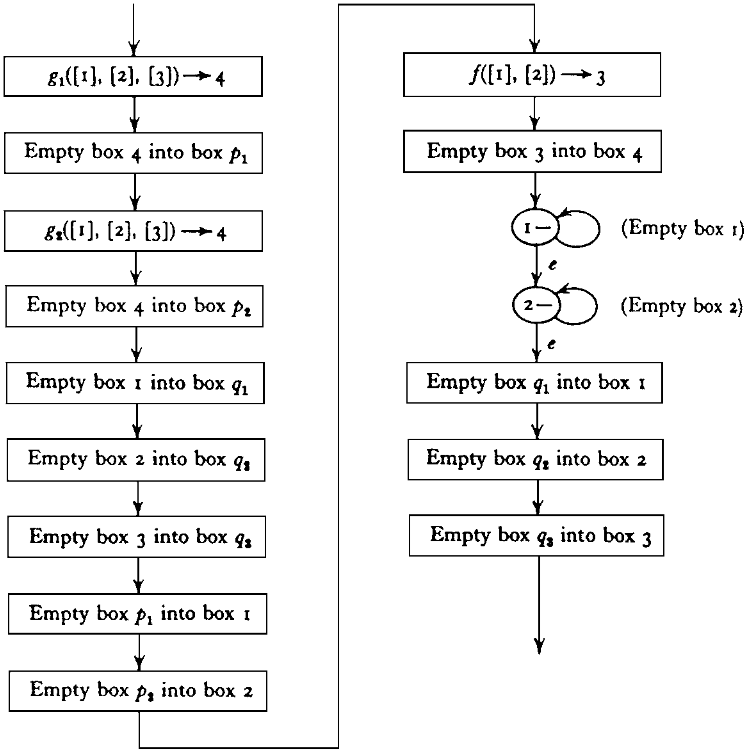
The thing is perfectly straightforward: It is a matter of shuttling the results of subcomputations around so as to be in the right boxes at the right times.

First, we identify five registers, none of which are used in any of the given programs. Let us call these registers p_1, p_2, q_1, q_2, and q_3. They will be used for temporary storage. In the single program which we want to construct, the 3 arguments are stored initially in boxes 1, 2, and 3; all other boxes are empty initially; and at the end, we want the n arguments back in boxes 1, 2, 3, and want the value $f(g_1([1], [2], [3]), g_2([1], [2], [3]))$ in box number 4. To arrange that, all we need are the three given programs, plus the program of Example 5.2 for emptying one box into another.

We simply compute $g_1([1], [2], [3])$ and store the result in box p_1 (which figures in none of the given programs, remember); then compute $g_2([1], [2], [3])$ and store the result in box p_2; then store the arguments in boxes 1, 2, and 3 in boxes q_1, q_2, and q_3, emptying boxes 1 through 4; then get the results of the computations of g_1 and g_2 out of boxes p_1 and p_2 where they have been stored, emptying them into boxes 1 and 2; then compute $f([1], [2]) = f[g_1(\text{original arguments}), g_2(\text{original arguments})]$, getting the result in box 3; and finally, tidy up, moving the overall result of the computation from box 3 to box 4, emptying box 3 in the process, and refilling boxes 1 through 3 with the original arguments of the overall computation, which were stored in boxes q_1, q_2, and q_3. Now everything is as it should be. The structure of the flow chart is shown in Figure 5-15.

Another process, called (*primitive*) *recursion*, is what is involved in defining multiplication as repeated addition, exponentiation as repeated multiplication, and so on. Suppose we have a 1-place functions f and a 3-place function g. The function h obtained from them by (primitive) recursion is the 2-place function h given by

$$\begin{aligned} h(x, 0) &= f(x) \\ h(x, y + 1) &= g(x, y, h(x, y)). \end{aligned}$$



$g_1([1], [2], [3]) \to 4$
Empty box 4 into box p_1
$g_2([1], [2], [3]) \to 4$
Empty box 4 into box p_2
Empty box 1 into box q_1
Empty box 2 into box q_2
Empty box 3 into box q_3
Empty box p_1 into box 1
Empty box p_2 into box 2
$f([1], [2]) \to 3$
Empty box 3 into box 4
1−
(Empty box 1)
e
2−
(Empty box 2)
e
Empty box q_1 into box 1
Empty box q_2 into box 2
Empty box q_3 into box 3

Figure 5-15. Composition.

For instance, if $f(x) = x$ and $g(x, y, z) = z + 1$, then

$$\begin{aligned}
h(x, 0) &= f(x) &&= x &&= x + 0\\
h(x, 1) &= g(x, 0, x) && &&= x + 1\\
h(x, 2) &= g(x, 1, x + 1) &&= (x + 1) + 1 &&= x + 2
\end{aligned}$$

and in general $h(x, y) = x + y$. Suppose f and g are abacus computable according to our specifications, and we are given programs for them:

$$\begin{array}{c} \downarrow \\ \boxed{f([1]) \to 2} \\ \downarrow \end{array} \qquad \begin{array}{c} \downarrow \\ \boxed{g_1([1], [2], [3]) \to 4} \\ \downarrow \end{array}.$$

We want to find a program for h, to show it is abacus computable

$$\begin{array}{c} \downarrow \\ \boxed{h([1], [2]) \to 3} \\ \downarrow \end{array}.$$

The thing is easily done, as in Figure 5-16.

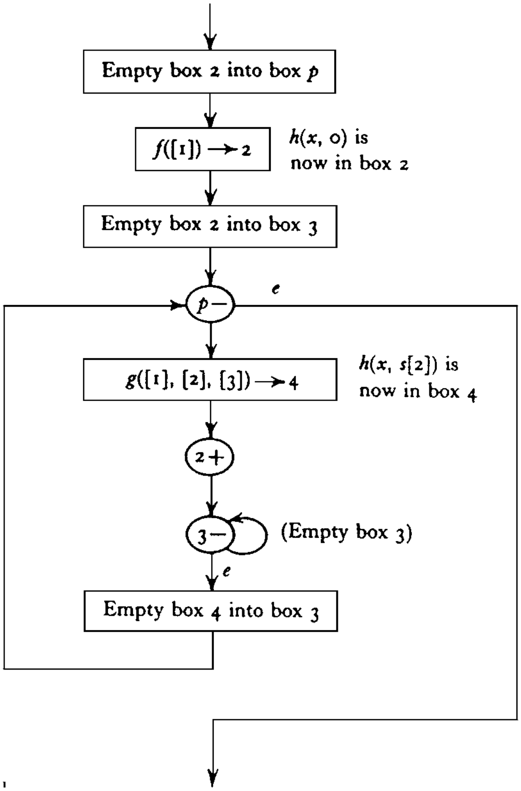


Figure 5-16. Recursion.

Initially, $[1] = x$, $[2] = y$, and $[3] = [4] = \cdots = 0$. We use a register number p that is not used in the f and g programs as a counter. We put y into it at the beginning, and after each stage of the computation we see whether $[p] = 0$. If so, the computation is essentially finished; if not, we subtract 1 from $[p]$ and go through another stage. In the first three steps we calculate $f(x)$ and see whether entry y was 0. If so, the first of the pair of equations for h is operative: $h(x, y) = h(x, 0) = f(x)$, and the computation is finished, with the result in box 3, as required. If not, the second of the pair of equations for h is operative, and we successively compute $h(x, 1)$, $h(x, 2), \ldots$ (see the cyle in Figure 5-16) until the counter (box p) is empty. At that point the computation is finished, with $h(x, y)$ in box 3, as required.

A final process is *minimization*. Suppose we have a 2-place function f; then we can define a 1-place function h as follows. If $f(x, 0), \ldots, f(x, i - 1)$ are all defined and $\neq 0$, and $f(x, i) = 0$, then $h(x) = i$. If there is no i with these properties, either because for some i the values $f(x, 0), \ldots, f(x, j - 1)$ are all defined and $\neq 0$ but $f(x, j)$ is not defined, or because for all i the value $f(x, i)$ is defined and $\neq 0$, then $h(x)$ is undefined. The function h is called the function obtained from f by minimization. If f is abacus computable, so is h, with a flow chart as in Figure 5-17.

Initially, box 2 is empty, so that if $f(x, 1) = 0$, the program will halt with the correct answer, $h(x) = 0$, in box 2. (Box 3 will be empty.) Otherwise, box 3 will be

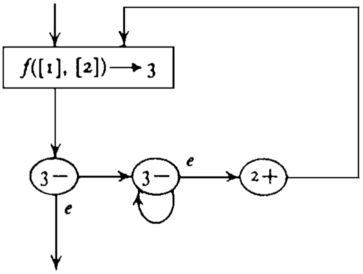


Figure 5-17. Minimization.

emptied and a single rock place in box 2, preparatory to the computation of $f(x, 1)$. If this value is 0, the program halts, with the correct value, $h(x) = 1$, in box 2. Otherwise, another rock is placed in box 2, and the procedure continues until such time (if any) as we have a number y of rocks in box 2 that is enough to make $f(x, y) = 0$.

The extensive class of functions obtainable from the trivial functions considered in the example at the beginning of this section by the kinds of processes considered in the rest of this section will be studied in the next chapter, where they will be given the name *recursive* functions. At this point we know the following:

5.8 Theorem. All recursive functions are abacus computable (and hence Turing computable).

So as we produce more examples of such functions, we are going to be producing more evidence for Turing's thesis.

Problems

5.1 Design an abacus machine for computing the difference function $\dot{-}$ defined by letting $x \dot{-} y = x - y$ if $y < x$, and $= 0$ otherwise.

5.2 The signum function sg is defined by letting $\mathrm{sg}(x) = 1$ if $x > 0$, and $= 0$ otherwise. Give a direct proof that sg is abacus computable by designing an abacus machine to compute it.

5.3 Give an indirect proof that sg is abacus computable by showing that sg is obtainable by composition from functions known to be abacus computable.

5.4 Show (directly by designing an appropriate abacus machine, or indirectly) that the function f defined by letting $f(x, y) = 1$ if $x < y$, and $= 0$ otherwise, is abacus computable.

5.5 The *quotient* and the *remainder* when the positive integer x is divided by the positive integer y are the unique natural numbers q and r such that $x = qy + r$ and $0 \le r < y$. Let the functions quo and rem be defined as follows: $\mathrm{rem}(x, y) =$ the remainder on dividing x by y if $y \neq 0$, and $= x$ if $y = 0$; $\mathrm{quo}(x, y) =$ the quotient on dividing x by y if $y \neq 0$, and $= 0$ if $y = 0$. Design an abacus machine for computing the remainder function rem.

5.6 Write an abacus-machine flow chart for computing the quotient function quo of the preceding problem.

5.7 Show that for any k there is a Turing machine that, when started on the leftmost 1 on a tape containing k blocks of 1s separated by single blanks, halts on the leftmost 1 on a tape that is exactly the same as the starting tape, except that everything has been moved one square to the right, without the machine in the course of its operations ever having moved left of the square on which it was started.

5.8 Review the operations of a Turing machine simulating some give abacus machine according to the method of this chapter. What is the furthest to the left of the square on which it is started that such a machine can ever go in the course of its operations?

5.9 Show that any abacus-computable function is computable by a Turing machine that never moves left of the square on which it is started.

5.10 Describe a reasonable way of coding abacus machines by natural numbers.

5.11 Given a reasonable way of coding abacus machines by natural numbers, let $d(x) = 1$ if the one-place function computed by abacus number x is defined and has value 0 for argument x, and $d(x) = 0$ otherwise. Show that this function is not abacus computable.

6

Recursive Functions

The intuitive notion of an effectively computable *function is the notion of a function for which there are definite, explicit rules, following which one could in principle compute its value for any given arguments. This chapter studies an extensive class of effectively computable functions, the* recursively computable, *or simply* recursive, *functions. According to* Church's thesis, *these are in fact* all *the effectively computable functions. Evidence for Church's thesis will be developed in this chapter by accumulating examples of effectively computable functions that turn out to be recursive. The subclass of* primitive recursive *functions is introduced in section 6.1, and the full class of* recursive *functions in section 6.2. The next chapter contains further examples. The discussion of recursive computability in this chapter and the next is entirely independent of the discussion of Turing and abacus computability in the preceding three chapters, but in the chapter after next the three notions of computability will be proved equivalent to each other.*

6.1 Primitive Recursive Functions

Intuitively, the notion of an *effectively computable* function f from natural numbers to natural numbers is the notion of a function for which there is a finite list of instructions that in principle make it possible to determine the value $f(x_1, \ldots, x_n)$ for any arguments $x_1, \ldots, x_n$. The instructions must be so definite and explicit that they require no external sources of information and no ingenuity to execute. But the determination of the value given the arguments need only be possible in principle, disregarding practical considerations of time, expense, and the like: the notion of effective computability is an idealized one.

For purposes of computation, the natural numbers that are the arguments and values of the function must be presented in some system of numerals or other, though the class of functions that is effectively computable will not be affected by the choice of system of numerals. (This is because conversion from one system of numerals to another is itself an effective process that can be carried out according to definite, explicit rules.) Of course, in practice some systems of numerals are easier to work with than others, but that is irrelevant to the idealized notion of effective computability.

For present purposes we adopt a variant of the primeval monadic or tally notation, in which a positive integer n is represented by n strokes. The variation is needed because we want to consider not just positive integers (excluding zero) but the natural numbers (including zero). We adopt the system in which the number zero is represented by the cipher 0, and a natural number $n > 0$ is represented by the cipher 0 followed by a sequence of n little raised strokes or *accents*. Thus the numeral for one is $0'$, the numeral for two is $0''$, and so on.

Two functions that are extremely easy to compute in this notation are the *zero* function, whose value $z(x)$ is the same, namely zero, for any argument x, and the *successor* function $s(x)$, whose value for any number x is the next larger number. In our special notation we write:

$$\begin{array}{llll} z(0) = 0 & z(0') = 0 & z(0'') = 0 & \cdots \\ s(0) = 0' & s(0') = 0'' & s(0'') = 0''' & \cdots. \end{array}$$

To compute the zero function, given any any argument, we simply ignore the argument and write down the symbol 0. To compute the successor function in our special notation, given a number written in that notation, we just add one more accent at the right.

Some other functions it is easy to compute (in *any* notation) are the *identity functions*. We have earlier encountered also the identity function of one argument, id or more fully id_1^1, which assigns to each natural number as argument that same number as value:

$$\mathrm{id}_1^1(x) = x.$$

There are two identity functions of two arguments: id_1^2 and id_2^2. For any pair of natural numbers as arguments, these pick out the first and second, respectively, as values:

$$\mathrm{id}_1^2(x, y) = x \qquad \mathrm{id}_2^2(x, y) = y.$$

In general, for each positive integer n, there are n identity functions of n arguments, which pick out the first, second, . . . , and nth of the arguments:

$$\mathrm{id}_i^n(x_1, \ldots, x_i, \ldots, x_n) = x_i.$$

Identity functions are also called *projection functions*. [In terms of analytic geometry, $\mathrm{id}_1^2(x, y)$ and $\mathrm{id}_2^2(x, y)$ are the projections x and y of the point (x, y) to the X-axis and to the Y-axis respectively.]

The foregoing functions—zero, successor, and the various identity functions—are together called the *basic* functions. They can be, so to speak, computed in one step, at least on one way of counting steps.

The stock of effectively computable functions can be enlarged by applying certain processes for defining new functions from old. A first sort of operation, composition, is familiar and straightforward. If f is a function of m arguments and each of $g_1, \ldots, g_m$ is a function of n arguments, then the function obtained by *composition* from $f, g_1, \ldots, g_m$ is the function h where we have

$$\boxed{h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))} \quad \text{(Cn)}$$

One might indicate this in shorthand:

$$h = \mathrm{Cn}[f, g_1, \ldots, g_m].$$

Composition is also called *substitution*.

Clearly, if the functions g_i are all effectively computable and the function f is effectively computable, then so is the function h. The number of steps needed to compute $h(x_1, \ldots, x_n)$ will be the sum of the number of steps needed to compute $y_1 = g_1(x_1, \ldots, x_n)$, the number needed to compute $y_2 = g_2(x_1, \ldots, x_n)$, and so on, plus at the end the number of steps needed to compute $f(y_1, \ldots, y_m)$.

6.1 Example (Constant functions). For any natural number n, let the *constant* function const_n be defined by $\mathrm{const}_n(x) = n$ for all x. Then for each n, const_n can be obtained from the basic functions by finitely many applications of composition. For, const_0 is just the zero function z, and $\mathrm{Cn}[s, z]$ is the function h with $h(x) = s(z(x)) = s(0) = 0' = 1 = \mathrm{const}_1(x)$ for all x, so $\mathrm{const}_1 = \mathrm{Cn}[s, z]$. (Actually, such notations as $\mathrm{Cn}[s, z]$ are genuine function symbols, belonging to the same grammatical category as h, and we could have simply written $\mathrm{Cn}[s, z](x) = s(z(x))$ here rather than the more longwinded ‘if $h = \mathrm{Cn}[s, z]$, then $h(x) = z(x)'$’.) Similarly $\mathrm{const}_2 = \mathrm{Cn}[s, \mathrm{const}_1]$, and generally $\mathrm{const}_{n+1} = \mathrm{Cn}[s, \mathrm{const}_n]$.

The examples of effectively computable functions we have had so far are admittedly not very exciting. More interesting examples are obtainable using a different process for defining new functions from old, a process that can be used to define addition in terms of successor, multiplication in terms of addition, exponentiation in terms of multiplication, and so on. By way of introduction, consider addition. The rules for computing this function in our special notation can be stated very concisely in two equations as follows:

$$x + 0 = x \qquad x + y' = (x + y)'.$$

To see how these equations enable us to compute sums consider adding $2 = 0''$ and $3 = 0'''$. The equations tell us:

$$\begin{array}{lll}
0'' + 0''' = (0'' + 0'')' & \text{by 2nd equation} & \text{with } x = 0'' \text{ and } y = 0'' \\
0'' + 0'' \;= (0'' + 0')' & \text{by 2nd equation} & \text{with } x = 0'' \text{ and } y = 0' \\
0'' + 0' \;\;= (0'' + 0)' & \text{by 2nd equation} & \text{with } x = 0'' \text{ and } y = 0 \\
0'' + 0 \;\;\;= 0'' & \text{by 1st equation} & \text{with } x = 0''.
\end{array}$$

Combining, we have the following:

$$\begin{aligned}
0'' + 0''' &= (0'' + 0'')' \\
&= (0'' + 0')'' \\
&= (0'' + 0)''' \\
&= 0'''''.
\end{aligned}$$

So the sum is $0''''' = 5$. Thus we use the second equation to reduce the problem of computing $x + y$ to that of computing $x + z$ for smaller and smaller z, until we arrive at $z = 0$, when the first equation tells us directly how to compute $x + 0$.

Similarly, for multiplication we have the rules or equations

$$x \cdot 0 = 0 \qquad x \cdot y' = x + (x \cdot y)$$

which enable us to reduce the computation of a product to the computation of sums, which we know how to compute:

$$\begin{aligned} 0'' \cdot 0''' &= 0'' + (0'' \cdot 0'') \\ &= 0'' + (0'' + (0'' \cdot 0')) \\ &= 0'' + (0'' + (0'' + (0'' \cdot 0))) \\ &= 0'' + (0'' + (0'' + 0)) \\ &= 0'' + (0'' + 0'') \end{aligned}$$

after which we would carry out the computation of the sum in the last line in the way indicated above, and obtain $0''''''$.

Now addition and multiplication are just the first two of a series of arithmetic operations, all of which are effectively computable. The next item in the series is exponentiation. Just as multiplication is repeated addition, so exponentiation is repeated multiplication. To compute x^y, that is, to raise x to the power y, multiply together y xs as follows:

$$x \cdot x \cdot x \cdot \cdots \cdot x \qquad \text{(a row of } y\ x\text{s).}$$

Conventionally, a product of *no* factors is taken to be 1, so we have the equation

$$x^0 = 0'.$$

For higher powers we have

$$\begin{aligned} x^1 &= x \\ x^2 &= x \cdot x \\ &\;\vdots \\ x^y &= x \cdot x \cdot \cdots \cdot x && \text{(a row of } y\ x\text{s)} \\ x^{y+1} &= x \cdot x \cdot \cdots \cdot x \cdot x = x \cdot x^y && \text{(a row of } y + 1\ x\text{s).} \end{aligned}$$

So we have the equation

$$x^{y'} = x \cdot x^y.$$

Again we have two equations, and these enable us to reduce the computation of a power to the computation of products, which we know how to do.

Evidently the next item in the series, *super-exponentiation*, would be defined as follows:

$$x^{x^{x^{x^{\cdot^{\cdot^{\cdot}}}}}} \qquad \text{(a stack of } y\ x\text{s).}$$

The alternative notation $x \uparrow y$ may be used for exponentiation to avoid piling up of superscripts. In this notation the definition would be written as follows:

$$x \uparrow x \uparrow x \uparrow \ldots \uparrow x \qquad \text{(a row of } y\ x\text{s).}$$

Actually, we need to indicate the grouping here. It is to the right, like this:

$$x \uparrow (x \uparrow (x \uparrow \ldots \uparrow x \ldots))$$

and not to the left, like this:

$$(\ldots((x \uparrow x) \uparrow x) \uparrow \ldots) \uparrow x.$$

For it makes a difference: $3 \uparrow (3 \uparrow 3) = 3 \uparrow (27) = 7\,625\,597\,484\,987$; while $(3 \uparrow 3) \uparrow 3 = 27 \uparrow 3 = 19\,683$. Writing $x \Uparrow y$ for the super-exponential, the equations would be

$$x \Uparrow 0 = 0' \qquad x \Uparrow y' = x \uparrow (x \Uparrow y).$$

The next item in the series, *super-duper-exponentiation*, is analogously defined, and so on.

The process for defining new functions from old at work in these cases is called (*primitive*) *recursion*. As our official format for this process we take the following:

$$\boxed{h(x, 0) = f(x), \;\; h(x, y') = g(x, y, h(x, y))} \text{ (Pr).}$$

Where the boxed equations—called the *recursion equations* for the function h—hold, h is said to be definable by (primitive) recursion from the functions f and g. In shorthand,

$$h = \mathrm{Pr}[f, g].$$

Functions obtainable from the basic functions by composition and recursion are called *primitive recursive.*

All such functions are effectively computable. For if f and g are effectively computable functions, then h is an effectively computable function. The number of steps needed to compute $h(x, y)$ will be the sum of the number of steps needed to compute $z_0 = f(x) = h(x, 0)$, the number needed to compute $z_1 = g(x, 0, z_0) = h(x, 1)$, the number needed to compute $z_2 = g(x, 1, z_1) = h(x, 2)$, and so on up to $z_y = g(x, y - 1, z_{y-1}) = h(x, y)$.

The definitions of sum, product, and power we gave above are approximately in our official boxed format. [The main difference is that the boxed format allows one, in computing $h(x, y')$, to apply a function taking x, y, and $h(x, y)$ as arguments. In the examples of sum, product, and power, we never needed to use y as an argument.] By fussing over the definitions we gave above, we can put them exactly into the format (Pr), thus showing addition and multiplication to be primitive recursive.

6.2 Example (The addition or sum function). We start with the definition given by the equations we had above,

$$x + 0 = x \qquad x + y' = (x + y)'.$$

As a step toward reducing this to the boxed format (Pr) for recursion, we replace the ordinary plus sign, written between the arguments, by a sign written out front:

$$\mathrm{sum}(x, 0) = x \qquad \mathrm{sum}(x, y') = \mathrm{sum}(x, y)'.$$

To put these equations in the boxed format (Pr), we must find functions f and g for which we have

$$f(x) = x \qquad g(x, y, —) = s(—)$$

for all natural numbers x, y, and —. Such functions lie ready to hand: $f = \mathrm{id}_1^1$, $g = \mathrm{Cn}[s, \mathrm{id}_3^3]$. In the boxed format we have

$$\mathrm{sum}(x, 0) = \mathrm{id}_1^1(x) \qquad \mathrm{sum}(x, s(y)) = \mathrm{Cn}\big[s, \mathrm{id}_3^3\big]\,(x, y, \mathrm{sum}(x, y))$$

and in shorthand we have

$$\mathrm{sum} = \mathrm{Pr}\big[\mathrm{id}_1^1, \mathrm{Cn}\big[s, \mathrm{id}_3^3\big]\big].$$

6.3 Example (The multiplication or product function). We claim prod = $\mathrm{Pr}[z, \mathrm{Cn}[\mathrm{sum}, \mathrm{id}_1^3, \mathrm{id}_3^3]]$. To verify this claim we relate it to the boxed formats (Cn) and (Pr). In terms of (Pr) the claim is that the equations

$$\mathrm{prod}(x, 0) = z(x) \qquad \mathrm{prod}(x, s(y)) = g(x, y, \mathrm{prod}(x, y))$$

hold for all natural numbers x and y, where [setting $h = g$, $f = \mathrm{sum}$, $g_1 = \mathrm{id}_1^3$, $g_2 = \mathrm{id}_3^3$ in the boxed (Cn) format] we have

$$\begin{aligned} g(x_1, x_2, x_3) &= \mathrm{Cn}\big[\mathrm{sum}, \mathrm{id}_1^3, \mathrm{id}_3^3\big](x_1, x_2, x_3) \\ &= \mathrm{sum}\big(\mathrm{id}_1^3(x_1, x_2, x_3), \mathrm{id}_3^3(x_1, x_2, x_3)\big) \\ &= x_1 + x_3 \end{aligned}$$

for all natural numbers x_1, x_2, x_3. Overall, then, the claim is that the equations

$$\mathrm{prod}(x, 0) = z(x) \qquad \mathrm{prod}(x, s(y)) = x + \mathrm{prod}(x, y)$$

hold for all natural numbers x and y, which is true:

$$x \cdot 0 = 0 \qquad x \cdot y' = x + x \cdot y.$$

Our rigid format for recursion serves for functions of two arguments such as sum and product, but we are sometimes going to wish to use such a scheme to define functions of a single argument, and functions of more than two arguments. Where there are three or more arguments $x_1, \ldots, x_n, y$ instead of just the two x, y that appear in (Pr), the modification is achieved by viewing each of the five occurrences of x in the boxed format as shorthand for $x_1, \ldots, x_n$. Thus with $n = 2$ the format is

$$\begin{aligned} h(x_1, x_2, 0) &= f(x_1, x_2) \\ h(x_1, x_2, s(y)) &= g(x_1, x_2, y, h(x_1, x_2, y)). \end{aligned}$$

6.4 Example (The factorial function). The factorial $x!$ for positive x is the product $1 \cdot 2 \cdot 3 \cdot \cdots \cdot x$ of all the positive integers up to and including x, and by convention $0! = 1$. Thus we have

$$\begin{aligned} 0! &= 1 \\ y'! &= y! \cdot y'. \end{aligned}$$

To show this function is recursive we would seem to need a version of the format for recursion with $n = 0$. Actually, however, we can simply define a two-argument function with a *dummy* argument, and then get rid of the dummy argument afterwards by composing with an identity function. For example, in the case of the factorial function we can define

$$\begin{aligned} \text{dummyfac}(x, 0) &= \text{const}_1(x) \\ \text{dummyfac}(x, y') &= \text{dummyfac}(x, y) \cdot y' \end{aligned}$$

so that $\text{dummyfac}(x, y) = y!$ regardless of the value of x, and then define $\text{fac}(y) = \text{dummyfac}$ (y, y). More formally,

$$\text{fac} = \text{Cn}\big[\text{Pr}\big[\text{const}_1, \text{Cn}\big[\text{prod}, \text{id}_3^3, \text{Cn}\big[\text{s}, \text{id}_2^3\big]\big]\big], \text{id}, \text{id}\big].$$

(We leave to the reader the verification of this fact, as well as the conversions of informal-style definitions into formal-style definitions in subsequent examples.)

The example of the factorial function can be generalized.

6.5 Proposition. Let f be a primitive recursive function. Then the functions

$$g(x, y) = f(x, 0) + f(x, 1) + \cdots + f(x, y) = \sum_{i=0}^{y} f(x, i)$$

$$h(x, y) = f(x, 0) \cdot f(x, 1) \cdot \cdots \cdot f(x, y) = \prod_{i=0}^{y} f(x, i)$$

are primitive recursive.

Proof: We have for the g the recursion equations

$$\begin{aligned} g(x, 0) &= f(x, 0) \\ g(x, y') &= g(x, y) + f(x, y') \end{aligned}$$

and similarly for h.

Readers may wish, in the further examples to follow, to try to find definitions of their own before reading ours; and for this reason we give the description of the functions first, and our definitions of them (in informal style) afterwards.

6.6 Example. The exponential or power function.

6.7 Example (The (modified) predecessor function). Define pred (x) to be the predecessor $x - 1$ of x for $x > 0$, and let $\text{pred}(0) = 0$ by convention. Then the function pred is primitive recursive.

6.8 Example (The (modified) difference function). Define $x \dot{-} y$ to be the difference $x - y$ if $x \geq y$, and let $x \dot{-} y = 0$ by convention otherwise. Then the function $\dot{-}$ is primitive recursive.

6.9 Example (The signum functions). Define $\text{sg}(0) = 0$, and $\text{sg}(x) = 1$ if $x > 0$, and define $\overline{\text{sg}}(0) = 1$ and $\overline{\text{sg}}(x) = 0$ if $x > 0$. Then sg and $\overline{\text{sg}}$ are primitive recursive.

Proofs

Example 6.6. $x \uparrow 0 = 1, x \uparrow s(y) = x \cdot (x \uparrow y)$, or more formally,

$$\exp = \mathrm{Pr}\big[\mathrm{Cn}[s, z], \mathrm{Cn}\big[\mathrm{prod}, \mathrm{id}_1^3, \mathrm{id}_3^3\big]\big].$$

Example 6.7. $\mathrm{pred}(0) = 0, \mathrm{pred}(y') = y$.

Example 6.8. $x \dot{-} 0 = x, x \dot{-} y' = \mathrm{pred}(x \dot{-} y)$.

Example 6.9. $\mathrm{sg}(y) = 1 \dot{-} (1 \dot{-} y), \overline{\mathrm{sg}}(y) = 1 \dot{-} y$.

6.2 Minimization

We now introduce one further process for defining new functions from old, which can take us beyond primitive recursive functions, and indeed can take us beyond total functions to partial functions. Intuitively, we consider a *partial* function f to be *effectively computable* if a list of definite, explicit instructions can be given, following which one will, in the case they are applied to any x in the domain of f, arrive after a finite number of steps at the value $f(x)$, but following which one will, in the case they are applied to any x not in the domain of f, go on forever without arriving at any result. This notion applies also to two- and many-place functions.

Now the new process we want to consider is this. Given a function f of $n + 1$ arguments, the operation of *minimization* yields a total or partial function h of n arguments as follows:

$$\mathrm{Mn}[f](x_1, \ldots, x_n) = \begin{cases} y & \text{if } f(x_1, \ldots, x_n, y) = 0, \text{ and for all } t < y \\ & \quad f(x_1, \ldots, x_n, t) \text{ is defined and } \neq 0 \\ \text{undefined} & \text{if there is no such } y. \end{cases}$$

If $h = \mathrm{Mn}[f]$ and f is an effectively computable total or partial function, then h also will be such a function. For writing x for $x_1, \ldots, x_n$, we compute $h(x)$ by successively computing $f(x, 0)$, $f(x, 1)$, $f(x, 2)$, and so on, stopping if and when we reach a y with $f(x, y) = 0$. If x is in the domain of h, there will be such a y, and the number of steps needed to compute $h(x)$ will be the sum of the number of steps needed to compute $f(x, 0)$, the number of steps needed to compute $f(x, 1)$, and so on, up through the number of steps needed to compute $f(x, y) = 0$. If x is not in the domain of h, this may be for either of two reasons. On the one hand, it may be that all of $f(x, 0)$, $f(x, 1)$, $f(x, 2)$, ... are defined, but they are all nonzero. On the other hand, it may be that for some i, all of $f(x, 0)$, $f(x, 1)$, ..., $f(x, i - 1)$ are defined and nonzero, but $f(x, i)$ is undefined. In either case, the attempt to compute $h(x)$ will involve one in a process that goes on forever without producing a result.

In case f is a total function, we do not have to worry about the second of the two ways in which $\mathrm{Mn}[f]$ may fail to be defined, and the above definition boils down to the following simpler form.

$$\mathrm{Mn}[f](x_1, \ldots, x_n) = \begin{cases} \text{the smallest } y \text{ for which} & \\ \quad f(x_1, \ldots, x_n, y) = 0 & \text{if such a } y \text{ exists} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The total function f is called *regular* if for every $x_1, \ldots, x_n$ there is a y such that $f(x_1, \ldots, x_n, y) = 0$. In case f is a regular function, Mn[f] will be a total function. In fact, if f is a total function, Mn[f] will be total if and only if f is regular.

For example, the product function is regular, since for every x, $x \cdot 0 = 0$; and Mn[prod] is simply the zero function. But the sum function is not regular, since $x + y = 0$ only in case $x = y = 0$; and Mn[sum] is the function that is defined only for 0, for which it takes the value 0, and undefined for all $x > 0$.

The functions that can be obtained from the basic functions z, s, id_i^n by the processes Cn, Pr, and Mn are called the *recursive* (total or partial) *functions*. (In the literature, 'recursive function' is often used to mean more specifically 'recursive *total* function', and 'partial recursive function' is then used to mean 'recursive total or partial function'.) As we have observed along the way, recursive functions are all effectively computable.

The hypothesis that, conversely, all effectively computable total functions are recursive is known as *Church's thesis* (the hypothesis that all effectively computable partial functions are recursive being known as the *extended* version of Church's thesis). The interest of Church's thesis derives largely from the following fact. Later chapters will show that some particular functions of great interest in logic and mathematics are nonrecursive. In order to infer from such a theoretical result the conclusion that such functions are not effectively computable (from which may be inferred the practical advice that logicians and mathematicians would be wasting their time looking for a set of instructions to compute the function), we need assurance that Church's thesis is correct.

At present Church's thesis is, for us, simply an hypothesis. It has been made somewhat plausible to the extent that we have shown a significant number of effectively computable functions to be recursive, but one can hardly on the basis of just these few examples be assured of its correctness. More evidence of the correctness of the thesis will accumulate as we consider more examples in the next two chapters.

Before turning to examples, it may be well to mention that the thesis that every effectively computable total function is *primitive* recursive would simply be erroneous. Examples of recursive total functions that are not primitive recursive are described in the next chapter.

Problems

6.1 Let f be a two-place recursive total function. Show that the following functions are also recursive:

(a) $g(x, y) = f(y, x)$

(b) $h(x) = f(x, x)$

(c) $k_{17}(x) = f(17, x)$ and $k^{17}(x) = f(x, 17)$.

6.2 Let $J_0(a, b)$ be the function coding pairs of positive integers by positive integers that was called J in Example 1.2, and from now on use the name J for the corresponding function coding pairs of natural numbers by natural numbers, so that $J(a, b) = J_0(a + 1, b + 1) - 1$. Show that J is primitive recursive.

6.3 Show that the following functions are primitive recursive:

(a) the *absolute difference* $|x - y|$, defined to be $x - y$ if $y < x$, and $y - x$ otherwise.

(b) the *order characteristic*, $\chi_{\leq}(x, y)$, defined to be 1 if $x \leq y$, and 0 otherwise.

(c) the *maximum* $\max(x, y)$, defined to be the larger of x and y.

6.4 Show that the following functions are primitive recursive:

(a) $c(x, y, z) = 1$ if $yz = x$, and 0 otherwise.

(b) $d(x, y, z) = 1$ if $J(y, z) = x$, and 0 otherwise.

6.5 Define $K(n)$ and $L(n)$ as the first and second entries of the pair coded (under the coding J of the preceding problems) by the number n, so that $J(K(n),\ L(n)) = n$. Show that the functions K and L are primitive recursive.

6.6 An alternative coding of pairs of numbers by numbers was considered in Example 1.2, based on the fact that every natural number n can be written in one and only one way as 1 less than a power of 2 times an odd number, $n = 2^{k(n)}(2l(n) \dot{-} 1) \dot{-} 1$. Show that the functions k and l are primitive recursive.

6.7 Devise some reasonable way of assigning code numbers to recursive functions.

6.8 Given a reasonable way of coding recursive functions by natural numbers, let $d(x) = 1$ if the one-place recursive function with code number x is defined and has value 0 for argument x, and $d(x) = 0$ otherwise. Show that this function is not recursive.

6.9 Let $h(x, y) = 1$ if the one-place recursive function with code number x is defined for argument y, and $h(x, y) = 0$ otherwise. Show that this function is not recursive.

7

Recursive Sets and Relations

In the preceding chapter we introduced the classes of primitive recursive and recursive functions. In this chapter we introduce the related notions of primitive recursive *and* recursive sets *and* relations, *which help provide many more examples of primitive recursive and recursive functions. The basic notions are developed in section 7.1. Section 7.2 introduces the related notion of a* semirecursive *set or relation. The optional section 7.3 presents examples of recursive total functions that are not primitive recursive.*

7.1 Recursive Relations

A set of, say, natural numbers is *effectively decidable* if there is an effective procedure that, applied to a natural number, in a finite amount of time gives the correct answer to the question whether it belongs to the set. Thus, representing the answer 'yes' by 1 and the answer 'no' by 0, a set is effectively decidable if and only if its *characteristic function* is effectively computable, where the characteristic function is the function that takes the value 1 for numbers in the set, and the value 0 for numbers not in the set. A set is called *recursively decidable*, or simply *recursive* for short, if its characteristic function is recursive, and is called *primitive recursive* if its characteristic function is primitive recursive. Since recursive functions are effectively computable, recursive sets are effectively decidable. Church's thesis, according to which all effectively computable functions are recursive, implies that all effectively decidable sets are recursive.

These notions can be generalized to relations. Officially, a two-place *relation* R among natural numbers will be simply a set of ordered pairs of natural numbers, and we write Rxy—or $R(x, y)$ if punctuation seems needed for the sake of readability—interchangeably with $(x, y) \in R$ to indicate that the relation R holds of x and y, which is to say, that the pair (x, y) belongs to R. Similarly, a k-place relation is a set of ordered k-tuples. [In case $k = 1$, a one-place relation on natural numbers ought to be a set of 1-tuples (sequences of length one) of numbers, but we will take it simply to be a set of numbers, not distinguishing in this context between n and (n). We thus write Sx or $S(x)$ interchangeably with $x \in S$.] The *characteristic function* of a k-place relation is the k-argument function that takes the value 1 for a k-tuple if the relation holds of that k-tuple, and the value 0 if it does not; and a relation is *effectively* *decidable* if its characteristic function is effectively computable, and is (*primitive*) *recursive* if its characteristic function is (primitive) recursive.

7.1 Example (Identity and order). The identity relation, which holds if and only if $x = y$, is primitive recursive, since a little thought shows its characteristic function is $1 - (\mathrm{sg}(x \dot{-} y) + \mathrm{sg}(y \dot{-} x))$. The strict less-than order relation, which holds if and only if $x < y$, is primitive recursive, since its characteristic function is $\mathrm{sg}(y \dot{-} x)$.

We are now ready to indicate an important process for obtaining new recursive functions from old. What follows is actually a pair of propositions, one about primitive recursive functions, the other about recursive functions (according as one reads the proposition with or without the bracketed word 'primitive'). The same proof works for both propositions.

7.2 Proposition (Definition by cases). Suppose that f is the function defined in the following form:

$$f(x, y) = \begin{cases} g_1(x, y) & \text{if } C_1(x, y) \\ \vdots & \qquad\vdots \\ g_n(x, y) & \text{if } C_n(x, y) \end{cases}$$

where $C_1, \ldots, C_n$ are (primitive) recursive relations that are mutually exclusive, meaning that for no x, y do more than one of them hold, and collectively exhaustive, meaning that for any x, y at least one of them holds, and where $g_1, \ldots, g_n$ are (primitive) recursive total functions. Then f is (primitive) recursive.

Proof: Let c_i be the characteristic function of C_i. By recursion, define $h_i(x, y, 0) = 0$, $h_i(x, y, z') = g_i(x, y)$. Let $k_i(x, y) = h_i(x, y, c_i(x, y))$, so $k_i(x, y) = 0$ unless $C_i(x, y)$ holds, in which case $k_i(x, y) = g_i(x, y)$. It follows that $f(x, y) = k_1(x, y) + \ldots + k_n(x, y)$, and f is (primitive) recursive since it is obtainable by primitive recursion and composition from the g_i and the c_i, which are (primitive) recursive by assumption, together with the addition (and identity) functions.

7.3 Example (The maximum and minimum functions). As an example of definition by cases, consider $\max(x, y) =$ the larger of the numbers x, y. This can be defined as follows:

$$\max(x, y) = \begin{cases} x & \text{if } x \geq y \\ y & \text{if } x < y \end{cases}$$

or in the official format of the proposition above with $g_1 = \mathrm{id}_1^2$ and $g_2 = \mathrm{id}_2^2$. Similarly, function $\min(x, y) =$ the smaller of x, y is also primitive recursive.

These particular functions, max and min, can also be shown to be primitive recursive in a more direct way (as you were asked to do in the problems at the end of the preceding chapter), but in more complicated examples, definition by cases makes it far easier to establish the (primitive) recursiveness of important functions. This is mainly because there are a variety of processes for defining new relations from old that can be shown to produce new (primitive) recursive relations when applied to (primitive) recursive relations. Let us list the most important of these.

Given a relation $R(y_1, \ldots, y_m)$ and total functions $f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)$, the relation defined by *substitution* of the f_i in R is the relation $R^*(x_1, \ldots, x_n)$ that holds of $x_1, \ldots, x_n$ if and only if R holds of $f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)$, or in symbols,

$$R^*(x_1, \ldots, x_n) \leftrightarrow R(f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)).$$

If the relation R^* is thus obtained by substituting functions f_i in the relation R, then the characteristic function c^* of R^* is obtainable by composition from the f_i and the characteristic function c of R:

$$c^*(x_1, \ldots, x_n) = c(f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n)).$$

Therefore, *the result of substituting recursive total functions in a recursive relation is itself a recursive relation.* (Note that it is important here that the functions be *total*.)

An illustration may make this important notion of substitution clearer. For a given function f, the *graph* relation of f is the relation defined by

$$G(x_1, \ldots, x_n, y) \leftrightarrow f(x_1, \ldots, x_n) = y.$$

Let $f^*(x_1, \ldots, x_n, y) = f(x_1, \ldots, x_n)$. Then f^* is recursive if f is, since

$$f^* = \mathrm{Cn}\big[f, \mathrm{id}_1^{n+1}, \ldots, \mathrm{id}_n^{n+1}\big].$$

Now $f(x_1, \ldots, x_n) = y$ if and only if

$$f^*(x_1, \ldots, x_n, y) = \mathrm{id}_{n+1}^{n+1}(x_1, \ldots, x_n, y).$$

Indeed, the latter condition is essentially just a long-winded way of writing the former condition. But this shows that if f is a recursive total function, then the graph relation $f(x_1, \ldots, x_n) = y$ is obtainable from the identity relation $u = v$ by substituting the recursive total functions f^* and id_{n+1}^{n+1}. Thus *the graph relation of a recursive total function is a recursive relation.* More compactly, if less strictly accurately, we can summarize by saying that the graph relation $f(x) = y$ is obtained by substituting the recursive total function f in the identity relation. (This compact, slightly inaccurate manner of speaking, which will be used in future, suppresses mention of the role of the identity functions in the foregoing argument.)

Besides substitution, there are several *logical* operations for defining new relations from old. To begin with the most basic of these, given a relation R, its *negation* or *denial* is the relation S that holds if and only if R does not:

$$S(x_1, \ldots, x_n) \leftrightarrow {\sim}R(x_1, \ldots, x_n).$$

Given two relations R_1 and R_2, their *conjunction* is the relation S that holds if and only if R_1 holds *and* R_2 holds:

$$S(x_1, \ldots, x_n) \leftrightarrow R_1(x_1, \ldots, x_n) \;\&\; R_2(x_1, \ldots, x_n)$$

while their *disjunction* is the relation S that holds if and only if R_1 holds *or* R_2 holds (or both do):

$$S(x_1, \dots, x_n) \leftrightarrow R_1(x_1, \dots, x_n) \vee R_2(x_1, \dots, x_n).$$

Conjunction and disjunctions of more than two relations are similarly defined. Note that when, in accord with our official definition, relations are considered as sets of k-tuples, the negation is simply the complement, the conjunction the intersection, and the disjunction the union.

Given a relation $R(x_1, \dots, x_n, u)$, by the relation obtained from R through *bounded universal quantification* we mean the relation S that holds of $x_1, \dots, x_n, u$ if and only if for all $v < u$, the relation R holds of $x_1, \dots, x_n, v$. We write

$$S(x_1, \dots, x_n, u) \leftrightarrow \forall v < u \;\; R(x_1, \dots, x_n, v)$$

or more fully:

$$S(x_1, \dots, x_n, u) \leftrightarrow \forall v(v < u \rightarrow R(x_1, \dots, x_n, v)).$$

By the relation obtained from R through *bounded existential quantification* we mean the relation S that holds of $x_1, \dots, x_n, u$ if and only if for some $v < u$, the relation R holds of $x_1, \dots, x_n, v$. We write

$$S(x_1, \dots, x_n, u) \leftrightarrow \exists v < u \; R(x_1, \dots, x_n, v)$$

or more fully:

$$S(x_1, \dots, x_n, u) \leftrightarrow \exists v(v < u \;\&\; R(x_1, \dots, x_n, v)).$$

The bounded quantifiers $\forall v \leq u$ and $\exists v \leq u$ are similarly defined.

The following theorem and its corollary are stated for recursive relations (and recursive total functions), but hold equally for primitive recursive relations (and primitive recursive functions), by the same proofs, though it would be tedious for writers and readers alike to include a bracketed '(primitive)' everywhere in the statement and proof of the result.

7.4 Theorem (Closure properties of recursive relations).

(a) A relation obtained by substituting recursive total functions in a recursive relation is recursive.

(b) The graph relation of any recursive total function is recursive.

(c) If a relation is recursive, so is its negation.

(d) If two relations are recursive, then so is their conjunction.

(e) If two relations are recursive, then so is their disjunction.

(f) If a relation is recursive, then so is the relation obtained from it by bounded universal quantification.

(g) If a relation is recursive, then so is the relation obtained from it by bounded existential quantification.

Proof:

(a), (b): These have already been proved.

(c): In the remaining items, we write simply x for $x_1, \ldots, x_n$. The characteristic function c^* of the negation or complement of R is obtainable from the characteristic function c of R by $c^*(x) = 1 \dot{-} c(x)$.

(d), (e): The characteristic function c^* of the conjunction or intersection of R_1 and R_2 is obtainable from the characteristic functions c_1 and c_2 of R_1 and R_2 by $c^*(x) = \min(c_1(x), c_2(x))$, and the characteristic function $c\dagger$ of the disjunction or union is similarly obtainable using max in place of min.

(f), (g): From the characteristic function $c(x, y)$ of the relation $R(x, y)$ the characteristic functions u and e of the relations $\forall v \leq y\ R(x_1, \ldots, x_n, v)$ and $\exists v \leq y\ R(x_1, \ldots, x_n, v)$ are obtainable as follows:

$$u(x, y) = \prod_{i=0}^{y} c(x, i) \qquad e(x, y) = \mathrm{sg}\left(\sum_{i=0}^{y} c(x, i)\right)$$

where the summation $\left(\sum\right)$ and product $\left(\prod\right)$ notation is as in Proposition 6.5. For the product will be 0 if any factor is 0, and will be 1 if and only if all factors are 1; while the sum will be positive if any summand is positive. For the strict bounds $\forall v < y$ and $\exists v < y$ we need only replace y by $y \dot{-} 1$.

7.5 Example (Primality). Recall that a natural number x is prime if $x > 1$ and there do not exist any u, v both $<x$ such that $x = u \cdot v$. The set P of primes is primitive recursive, since we have

$$P(x) \leftrightarrow 1 < x \,\&\, \forall u < x\ \forall v < x(u \cdot v \neq x).$$

Here the relation $1 < x$ is the result of substituting const_1 and id into the relation $y < x$, which we know to be primitive recursive from Example 7.1, and so this relation is primitive recursive by clause (a) of the theorem. The relation $u \cdot v = x$ is the graph of a primitive recursive function, namely, the product function; hence this relation is primitive recursive by clause (b) of the theorem. So P is obtained by negation, bounded universal quantification, and conjunction from primitive recursive relations, and is primitive recursive by clauses (c), (d), and (f) of the theorem.

7.6 Corollary (Bounded minimization and maximization). Given a (primitive) recursive relation R, let

$$\mathrm{Min}[R](x_1, \ldots, x_n, w) = \begin{cases} \text{the smallest } y \leq w \text{ for which} & \\ \quad R(x_1, \ldots, x_n, y) & \text{if such a } y \text{ exists} \\ w + 1 & \text{otherwise} \end{cases}$$

and

$$\mathrm{Max}[R](x_1, \ldots, x_n, w) = \begin{cases} \text{the largest } y \leq w \text{ for which} & \\ \quad R(x_1, \ldots, x_n, y) & \text{if such a } y \text{ exists} \\ 0 & \text{otherwise.} \end{cases}$$

Then $\mathrm{Min}[R]$ and $\mathrm{Max}[R]$ are (primitive) recursive total functions.

Proof: We give the proof for Min. Write x for $x_1, \dots, x_n$. Consider the (primitive) recursive relation $\forall t \leq y \sim R(x, t)$, and let c be its characteristic function. If there is a smallest $y \leq w$ such that $R(x, y)$, then abbreviating $c(x, i)$ to $c(i)$ we have

$$c(0) = c(1) = \cdots = c(y-1) = 1 \qquad c(y) = c(y+1) = \cdots = c(w) = 0.$$

So c takes the value 1 for the y numbers $i < y$, and the value 0 thereafter. If there is no such y, then

$$c(0) = c(1) = \cdots = c(w) = 1.$$

So c takes the value 1 for all $w + 1$ numbers $i \leq w$. In either case

$$\mathrm{Min}[R](x, w) = \sum_{i=0}^{w} c(x, i)$$

and is therefore (primitive) recursive. The proof for Max is similar, and is left to the reader.

7.7 Example (Quotients and remainders). Given natural numbers x and y with $y > 0$, there are unique natural numbers q and r such that $x = q \cdot y + r$ and $r < y$. They are called the *quotient* and *remainder* on division of x by y. Let quo(x, y) be the quotient on dividing x by y if $y > 0$, and set quo$(x, 0) = 0$ by convention. Let rem(x, y) be the remainder on dividing x by y if $y > 0$, and set rem$(x, 0) = x$ by convention. Then quo is primitive recursive, as an application of bounded maximization, since $q \leq x$ and q is the largest number such that $q \cdot y \leq x$.

$$\mathrm{quo}(x, y) = \begin{cases} \text{the largest } z \leq x \text{ such that } y \cdot z \leq x & \text{if} \quad y \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

We apply the preceding corollary (in its version for primitive recursive functions and relations). If we let $Rxyz$ be the relation $y \cdot z \leq x$, then quo$(x, y) = \mathrm{Max}[R](x, y, x)$, and therefore quo is primitive recursive. Also rem is primitive recursive, since rem$(x, y) = x \dot{-} (\mathrm{quo}(x, y) \cdot y)$. Another notation for rem(x, y) is x mod y.

7.8 Corollary. Suppose that f is a regular primitive function and that there is a primitive recursive function g such that the least y with $f(x_1, \dots, x_n, y) = 0$ is always less than $g(x_1, \dots, x_n)$. Then Mn$[f]$ is not merely recursive but primitive recursive.

Proof: Let $R(x_1, \dots, x_n, y)$ hold if and only if $f(x_1, \dots, x_n, y) = 0$. Then

$$\mathrm{Mn}[f](x_1, \dots, x_n) = \mathrm{Min}[R](x_1, \dots, x_n, g(x_1, \dots, x_n)).$$

7.9 Proposition. Let R be an $(n + 1)$-place recursive relation. Define a total or partial function r by

$$r(x_1, \dots, x_n) = \text{the least } y \text{ such that } R(x_1, \dots, x_n, y).$$

Then r is recursive.

Proof: The function r is just Mn$[c]$, where c is the characteristic function of $\sim R$.

Note that if r is a function and R its graph relation, then $r(x)$ is the *only* y such that $R(x, y)$, and therefore *a fortiori* the *least* such y (as well as the *greatest* such y).

So the foregoing proposition tells us that if the graph relation of a function is recursive, the function is recursive. We have not set this down as a numbered corollary because we are going to be getting a stronger result at the beginning of the next section.

7.10 Example (The next prime). Let $f(x) =$ the least y such that $x < y$ and y is prime. The relation

$$x < y \ \& \ y \text{ is prime}$$

is primitive recursive, using Example 7.5. Hence the function f is recursive by the preceding proposition. There is a theorem in Euclid's *Elements* that tells us that for any given number x there exists a prime $y > x$, from which we know that our function f is total. But actually, the proof in Euclid shows that there is a prime $y > x$ with $y \leq x! + 1$. Since the factorial function is primitive recursive, the Corollary 7.8 applies to show that f is actually *primitive* recursive.

7.11 Example (Logarithms). Subtraction, the inverse operation to addition, can take us beyond the natural numbers to negative integers; but we have seen there is a reasonable modified version $\dot{-}$ that stays within the natural numbers, and that it is primitive recursive. Division, the inverse operation to multiplication, can take us beyond the integers to fractional rational numbers; but again we have seen there is a reasonable modified version quo that is primitive recursive. Because the power or exponential function is not commutative, that is, because in general $x^y \neq y^x$, there are *two* inverse operations: the yth root of x is the z such that $z^y = x$, while the base-x logarithm of y is the z such that $x^z = y$. Both can take us beyond the rational numbers to irrational real numbers or even imaginary and complex numbers. But again there is a reasonable modified version, or several reasonable modified versions. Here is one for the logarithms

$$\mathrm{lo}(x, y) = \begin{cases} \text{the greatest } z \leq x \text{ such that } y^z \text{ divides } x & \text{if } x, y > 1 \\ 0 & \text{otherwise} \end{cases}$$

where 'divides x' means 'divides x without remainder'. Clearly if $x, y > 1$ and y^z divides x, z must be (quite a bit) less than x. So we can agrue as in the proof of 7.7 to show that lo is a primitive recursive function. Here is another reasonable modified logarithm function:

$$\mathrm{lg}(x, y) = \begin{cases} \text{the greatest } z \text{ such that } y^z \leq x & \text{if } x, y > 1 \\ 0 & \text{otherwise.} \end{cases}$$

The proof that lg is primitive recursive is left to the reader.

The next series of examples pertain to the coding of finite sequences of natural numbers by single natural numbers. The coding we adopt is based on the fact that each positive integer can be written in one and only one way as a product of powers of larger and larger primes. Specifically:

$$(a_0, a_1, \ldots, a_{n-1}) \text{ is coded by } 2^n 3^{a_0} 5^{a_1} \cdots \pi(n)^{a_{n-1}}$$

where $\pi(n)$ is the nth prime (counting 2 as the 0th). (When we first broached the topic of coding finite sequences by single numbers in section 1.2, we used a slightly different coding. That was because we were then coding finite sequences of positive integers, but now want to code finite sequences of natural numbers.) We state the examples first and invite the reader to try them before we give our own proofs.

7.12 Example (The nth prime). Let $\pi(n)$ be the nth prime, counting 2 as the 0th, so $\pi(0) = 2$, $\pi(1) = 3$, $\pi(2) = 5$, $\pi(3) = 7$, and so on. This function is primitive recursive.

7.13 Example (Length). There is a primitive recursive function lh such that if s codes a sequence $(a_0, a_1, \ldots, a_{n-1})$, then the value lh($s$) is the *length* of that sequence.

7.14 Example (Entries). There is a primitive recursive function ent such that if s codes a sequence $(a_0, a_1, \ldots, a_{n-1})$, then for each $i < n$ the value of ent(s, i) is the ith *entry* in that sequence (counting a_0 as the 0th).

Proofs

Example 7.12. $\pi(0) = 2$, $\pi(x') = f(\pi(x))$, where f is the next prime function of Example 7.10. The form of the definition is similar to that of the factorial function: see Example 6.4 for how to reduce definitions of this form to the official format for recursion.

Example 7.13. $\mathrm{lh}(s) = \mathrm{lo}(s, 2)$ will do, where lo is as in Example 7.11. Applied to

$$2^n 3^{a_0} 5^{a_1} \cdots \pi(n)^{a_{n-1}}$$

this function yields n.

Example 7.14. $\mathrm{ent}(s, i) = \mathrm{lo}(s, \pi(i + 1))$ will do. Applied to

$$2^n 3^{a_0} 5^{a_1} \cdots \pi(n)^{a_{n-1}}$$

and i, this function yields a_i.

There are some further examples pertaining to coding, but these will not be needed till a much later chapter, and even then only in a section that is optional reading, so we defer them to the optional final section of this chapter. Instead we turn to another auxiliary notion.

7.2 Semirecursive Relations

Intuitively, a set is (*positively*) *effectively semidecidable* if there is an effective procedure that, applied to any number, will if the number is in the set in a finite amount of time give the answer ‘yes’, but will if the number is not in the set never give an answer. For instance, the domain of an effectively computable partial function f is always effectively semidecidable: the procedure for determining whether n is in the domain of f is simply to try to compute the value $f(n)$; if and when we succeed, we know that n is in the domain; but if n is not in the domain, we never succeed.

The notion of effective semidecidability extends in the obvious way to relations. When applying the procedure, after any number t of steps of computation, we can tell whether we have obtained the answer ‘yes’ already, or have so far obtained no answer. Thus if S is a semidecidable set we have

$$S(x) \leftrightarrow \exists t\, R(x, t)$$

where R is the *effectively decidable* relation 'by t steps of computation we obtain the answer "yes"'. Conversely, if R is an effectively decidable relation of any kind, and S is the relation obtained from R by (unbounded) existential quantification, then S is effectively semidecidable: we can attempt to determine whether n is in S by checking whether $R(n, 0)$ holds, and if not, whether $R(n, 1)$ holds, and if not, whether $R(n, 2)$ holds, and so on. If n is in S, we must eventually find a t such that $R(n, t)$, and will thus obtain the answer 'yes'; but if n is not in S, we go on forever without obtaining an answer.

Thus we may characterize the effectively semidecidable sets as those obtained from two-place effectively decidable relations by existential quantification, and more generally, the n-place effectively semidecidable relations as those obtained from $(n + 1)$-place effectively decidable relations by existential quantification. We define an n-place relation S on natural numbers to be (*positively*) *recursively semidecidable*, or simply *semirecursive*, if it is obtainable from an $(n + 1)$-place recursive relation R by existential quantification, thus:

$$S(x_1, \ldots, x_n) \leftrightarrow \exists y\, R(x_1, \ldots, x_n, y).$$

A y such that R holds of the x_i and y may be called a 'witness' to the relation S holding of the x_i (provided we understand that when the witness is a number rather than a person, a witness only testifies to what is true). Semirecursive relations are effectively semidecidable, and Church's thesis would imply that, conversely, effectively semidecidable relations are semirecursive.

These notions should become clearer as we work out their most basic properties, an exercise that provides an opportunity to review the basic properties of recursive relations. The closure properties of recursive relations established in Theorem 7.4 can be used to establish a similar but not identical list of properties of semirecursive relations.

7.15 Corollary (Closure properties of semirecursive relations).

(a) Any recursive relation is semirecursive.

(b) A relation obtained by substituting recursive total functions in a semirecursive relation is semirecursive.

(c) If two relations are semirecursive, then so is their conjunction.

(d) If two relations are semirecursive, then so is their disjunction.

(e) If a relation is semirecursive, then so is the relation obtained from it by bounded universal quantification.

(f) If a relation is semirecursive, then so is the relation obtained from it by existential quantification.

Proof: We write simply x for $x_1, \ldots, x_n$.

(a): If Rx is a recursive relation, then the relation S given by $Sxy \leftrightarrow (Rx \,\&\, y = y)$ is also recursive, and we have $R(x) \leftrightarrow \exists y\, Sxy$.

(b): If Rx is a semirecursive relation, say $Rx \leftrightarrow \exists y Sxy$ where S is recursive, and if $R^*x \leftrightarrow Rf(x)$, where f is a recursive total function, then the relation S^* given by $S^*xy \leftrightarrow Sf(x)y$ is also recursive, and we have $R^*x \leftrightarrow \exists y\ S^*xy$ and R^* is semirecursive.

(c): If R_1x and R_2x are semirecursive relations, say $R_ix \leftrightarrow \exists y\ S_ixy$ where S_1 and S_2 are recursive, then the relation S given by $Sxw \leftrightarrow \exists y_1 < w\ \exists y_2 < w(S_1xy_1\ \&\ S_2xy_2)$ is also recursive, and we have $(R_1x\ \&\ R_2y) \leftrightarrow \exists w\ Sxw$. We are using here the fact that for any two numbers y_1 and y_2, there is a number w greater than both of them.

(d): If R_i and S_i are as in (c), then the relation S given by $Sxy \leftrightarrow (S_1xy \vee S_2xy)$ is also recursive, and we have $(R_1y \vee R_2x) \leftrightarrow \exists y\ Sxy$.

(e): If Rx is a semirecursive relation, say $Rx \leftrightarrow \exists y\ Sxy$ where S is recursive, and if $R^*x \leftrightarrow \forall u < xRu$, then the relation S^* given by $S^*xw \leftrightarrow \forall u < x\ \exists y < w\ Suy$ is also recursive, and we have $R^*x \leftrightarrow \exists w\ S^*xw$. We are using here the fact that for any finite number of numbers $y_0, y_1, \ldots, y_x$ there is a number w greater than all of them.

(f): If Rxy is a semirecursive relation, say $Rxy \leftrightarrow \exists z\ Sxyz$ where S is recursive, and if $R^*x \leftrightarrow \exists y\ Rxy$, then the relation S^* given by $S^*xw \leftrightarrow \exists y < w\ \exists z < w\ Sxyz$ is also recursive, and we have $R^*x \leftrightarrow \exists w\ S^*xw$.

The potential for semirecursive relations to yield new recursive relations and functions is suggested by the following propositions. Intuitively, if we have a procedure that will eventually tell us when a number is in a set (but will tell us nothing if it is not), and *also* have a procedure that will eventually tell us when a number is not in a set (but will tell us nothing if it is), then by combining them we can get a procedure that will tell us *whether or not* a number is in the set: apply both given procedures (say by doing a step of the one, then a step of the other, alternately), and eventually one or the other must give us an answer. In jargon, if a set and its complement are both effectively semidecidable, the set is decidable. The next proposition is the formal counterpart of this observation.

7.16 Proposition (Complementation principle, or Kleene's theorem). If a set and its complement are both semirecursive, then the set (and hence also its complement) is recursive.

Proof: If Rx and $\sim Rx$ are both semirecursive, say $Rx \leftrightarrow \exists y\ S^+xy$ and $\sim Rx \leftrightarrow \exists y\ S^-xy$, then the relation S^* given by $S^*xy \leftrightarrow (S^+xy \vee S^-xy)$ is recursive, and if f is the function defined by letting $f(x)$ be the least y such that S^*xy, then f is a recursive total function. But then we have $Rx \leftrightarrow S^+xf(x)$, showing that R is obtainable by substituting a recursive total function in a recursive relation, and is therefore recursive.

7.17 Proposition (First graph principle). If the graph relation of a total or partial function f is semirecursive, then f is a recursive total or partial function.

Proof: Suppose $f(x) = y \leftrightarrow \exists z\, Sxyz$, where S is recursive. We first introduce two auxiliary functions:

$$g(x) = \begin{cases} \text{the least } w \text{ such that} & \\ \quad \exists y < w\ \exists z < w\ Sxyz & \text{if such a } w \text{ exists} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$h(x, w) = \begin{cases} \text{the least } y < w \text{ such that} & \\ \quad \exists z < w\ Sxyz & \text{if such a } y \text{ exists} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Here the relations involved are *recursive*, and not just semirecursive, since they are obtained from S by *bounded*, not unbounded, existential quantification. So g and h are recursive. And a little thought shows that $f(x) = h(x, g(x))$, so f is recursive also.

The converse of the foregoing proposition is also true—the graph relation of a recursive partial function is semirecursive, and hence a total or partial function is recursive if and only if its graph relation is recursive or semirecursive—but we are not at this point in a position to prove it.

An *unavoidable* appeal to Church's thesis is made whenever one passes from a theorem about what is or isn't recursively computable to a conclusion about what is or isn't effectively computable. On the other hand, an *avoidable* or *lazy* appeal to Church's thesis is made whenever, in the proof of a technical theorem, we skip the verification that certain obviously effectively computable functions are recursively computable. Church's thesis is mentioned in connection with omissions of verifications only when writing for comparatively inexperienced readers, who cannot reasonably be expected to be able to fill in the gap for themselves; when writing for the more experienced reader one simply says "proof left to reader" as in similar cases elsewhere in mathematics. The reader who works through the following optional section and/or the optional Chapter 8 and/or the optional sections of Chapter 15 will be well on the way to becoming "experienced" enough to fill in virtually any such gap.

7.3* Further Examples

The list of recursive functions is capable of indefinite extension using the machinery developed so far. We begin with the examples pertaining to coding that were alluded to earlier.

7.18 Example (First and last). There are primitive recursive functions fst and lst such that if s codes a sequence $(a_0, a_1, \ldots, a_{n-1})$, then fst($s$) and lst($s$) are the first and last entries in that sequence.

7.19 Example (Extension). There is a primitive recursive function ext such that if s codes a sequence $(a_0, a_1, \ldots, a_{n-1})$, then for any b, ext(s, b) codes the *extended* sequence $(a_0, a_1, \ldots, a_{n-1}, b)$.

7.20 Example (Concatenation). There is a primitive recursive function conc such that if s codes a sequence $(a_0, a_1, \ldots, a_{n-1})$ and t codes a sequence $(b_0, b_1, \ldots, b_{m-1})$, then conc (s, t) codes the *concatenation* $(a_0, a_1, \ldots, a_{n-1}, b_0, b_1, \ldots, b_{m-1})$ of the two sequences.

Proofs

Example 7.18. $\mathrm{fst}(s) = \mathrm{ent}(s, 0)$ and $\mathrm{lst}(s) = \mathrm{ent}(s, \mathrm{lh}(s) \dot{-} 1)$ will do.

Example 7.19. $\mathrm{ext}(s, b) = 2 \cdot s \cdot \pi(\mathrm{lh}(s) + 1)^b$ will do. Applied to

$$2^n 3^{a_0} 5^{a_1} \cdots \pi(n)^{a_{n-1}}$$

this function yields

$$2^{n+1} 3^{a_0} 5^{a_1} \cdots \pi(n)^{a_{n-1}} \pi(n+1)^b.$$

Example 7.20. A head-on approach here does not work, and we must proceed a little indirectly, first introducing an auxiliary function such that

$$g(s, t, i) = \text{ the code for } (a_0, a_1, \ldots, a_{n-1}, b_0, b_1, \ldots, b_{i-1}).$$

We can then obtain the function we really want as $\mathrm{conc}(s, t) = g(s, t, \mathrm{lh}(t))$. The auxiliary g is obtained by recursion as follows:

$$\begin{aligned} g(s, t, 0) &= s \\ g(s, t, i') &= \mathrm{ext}(g(s, t, i), \mathrm{ent}(t, i)). \end{aligned}$$

Two more we leave entirely to the reader.

7.21 Example (Truncation). There is a primitive recursive function tr such that if s codes a sequence $(a_0, a_1, \ldots, a_{n-1})$ and $m \leq n$, then $\mathrm{tr}(s, m)$ codes the *truncated* sequence $(a_0, a_1, \ldots, a_{m-1})$.

7.22 Example (Substitution). There is a primitive recursive function sub such that if s codes a sequence $(a_1, \ldots, a_k)$, and c and d are any natural numbers, then $\mathrm{sub}(s, c, d)$ codes the sequence that results upon taking s and substituting for any entry that is equal to c the number d instead.

We now turn to examples, promised in the preceding chapter, of recursive total functions that are not primitive recursive.

7.23 Example (The Ackermann function). Let $\ll 0 \gg$ be the operation of addition, $\ll 1 \gg$ the operation of multiplication, $\ll 2 \gg$ the operation of exponentiation, $\ll 3 \gg$ the operation of super-exponentiation, and so on, and let $\alpha(x, y, z) = x \ll y \gg z$ and $\gamma(x) = \alpha(x, x, x)$. Thus

$$\begin{aligned} \gamma(0) &= 0 + 0 &&= 0 \\ \gamma(1) &= 1 \cdot 1 &&= 1 \\ \gamma(2) &= 2^2 &&= 4 \\ \gamma(3) &= 3^{3^3} &&= 7\,625\,597\,484\,987 \end{aligned}$$

after which the values of $\gamma(x)$ begin to grow very rapidly. A related function δ is determined as follows:

$$\begin{aligned} \beta_0(0) &= 2 & \beta_0(y') &= (\beta_0(y))' \\ \beta_{x'}(0) &= 2 & \beta_{x'}(y') &= \beta_x(\beta_{x'}(y)) \\ \beta(x, y) &= \beta_x(y) \\ \delta(x) &= \beta(x, x). \end{aligned}$$

Clearly each of $\beta_0, \beta_1, \beta_2, \ldots$ is recursive. The proof that β and hence δ are also recursive is outlined in a problem at the end of the chapter. (The proof for α and γ would be similar.) The proof that γ and hence α is not primitive recursive in effect proceeds by showing that one needs to apply recursion at least once to get a function that grows as fast as the addition function, at least twice to get one that grows as fast as the multiplication function, and so on; so that no finite number of applications of recursion (and composition, starting with the zero, successor, and identity functions) can give a function that grows as fast as γ. (The proof for β and δ would be similar.) While it would take us too far afield to give the whole proof here, working through the first couple of cases can give insight into the nature of recursion. We present the first case next and outline the second in the problems at the end of the chapter.

7.24 Proposition. It is impossible to obtain the sum or addition function from the basic functions (zero, successor, and identity) by composition, without using recursion.

Proof: To prove this negative result we claim something positive, that if f belongs to the class of functions that can be obtained from the basic functions using only composition, then there is a positive integer a such that for all $x_1, \ldots, x_n$ we have $f(x_1, \ldots, x_n) < x + a$, where x is the largest of $x_1, \ldots, x_n$. No such a can exist for the addition function, since $(a + 1) + (a + 1) > (a + 1) + a$, so it will follow that the addition function is not in the class in question—provided we can prove our claim. The claim is certainly true for the zero function (with $a = 1$), and for the successor function (with $a = 2$), and for each identity function (with $a = 1$ again). Since every function in the class we are interested in is built up step by step from these functions using composition, it will be enough to show if the claim holds for given functions, it holds for the function obtained from them by composition.

So consider a composition

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)).$$

Suppose we know

$$g_i(x_1, \ldots, x_n) < x + a_j \qquad \text{where } x \text{ is the largest of the } x_j$$

and suppose we know

$$f(y_1, \ldots, y_m) < y + b \qquad \text{where } y \text{ is the largest of the } y_i.$$

We want to show there is a c such that

$$h(x_1, \ldots, x_n) < x + c \qquad \text{where } x \text{ is the largest of the } x_j.$$

Let a be the largest of $a_1, \ldots, a_m$. Then where x is the largest of the x_j, we have

$$g_i(x_1, \ldots, x_n) < x + a$$

so if $y_i = g_i(x_1, \ldots, x_n)$, then where y is the largest of the y_i, we have $y < x + a$. And so

$$h(x_1, \ldots, x_n) = f(y_1, \ldots, y_m) < (x + a) + b = x + (a + b)$$

and we may take $c = a + b$.

Problems

7.1 Let R be a two-place primitive recursive, recursive, or semirecursive relation. Show that the following relations are also primitive recursive, recursive, or semirecursive, accordingly:

(a) the *converse* of R, given by $S(x, y) \leftrightarrow R(y, x)$

(b) the *diagonal* of R, given by $D(x) \leftrightarrow R(x, x)$

(c) for any natural number m, the *vertical and horizontal sections* of R at m, given by

$$R_m(y) \leftrightarrow R(m, y) \quad \text{and} \quad R^m(x) \leftrightarrow R(x, m).$$

7.2 Prove that the function lg of Example 7.11 is, as there asserted, primitive recursive.

7.3 For natural numbers, write $u \mid v$ to mean that u divides v without remainder, that is, there is a w such that $u \cdot w = v$. [Thus $u \mid 0$ holds for all u, but $0 \mid v$ holds only for $v = 0$.] We say z is the *greatest common divisor* of x and y, and write $z = \gcd(x, y)$, if $z \mid x$ and $z \mid y$ and whenever $w \mid x$ and $w \mid y$, then $w \leq z$ [except that, by convention, we let $\gcd(0, 0) = 0$]. We say z is the *least common multiple* of x and y, and write $z = \mathrm{lcm}(x, y)$, if $x \mid z$ and $y \mid z$ and whenever $x \mid w$ and $y \mid w$, then $z \leq w$. Show that the functions gcd and lcm are primitive recursive.

7.4 For natural numbers, we say x and y are *relatively prime* if $\gcd(x, y) = 1$, where gcd is as in the preceding problem. The *Euler ϕ-function* $\phi(n)$ is defined as the number of $m < n$ such that $\gcd(m, n) = 1$. Show that ϕ is primitive recursive. More generally, let Rxy be a (primitive) recursive relation, and let $r(x) =$ the number of $y < x$ such that Rxy. Show that r is (primitive) recursive.

7.5 Let A be an infinite recursive set, and for each n, let $a(n)$ be the nth element of A in increasing order (counting the least element as the 0th). Show that the function a is recursive.

7.6 Let f be a (primitive) recursive total function, and let A be the set of all n such that the value $f(n)$ is 'new' in the sense of being different from $f(m)$ for all $m < n$. Show that A is (primitive) recursive.

7.7 Let f be a recursive total function whose range is infinite. Show that there is a one-to-one recursive total function g whose range is the same as that of f.

7.8 Let us define a real number ξ to be *primitive recursive* if the function $f(x) =$ the digit in the $(x + 1)$st place in the decimal expansion of ξ is primitive recursive. [Thus if $\xi = \sqrt{2} = 1.4142\ldots$, then $f(0) = 4$, $f(1) = 1$, $f(2) = 4$, $f(3) = 2$, and so on.] Show that $\sqrt{2}$ is a primitive recursive real number.

7.9 Let $f(n)$ be the nth entry in the infinite sequence 1, 1, 2, 3, 5, 8, 13, 21, . . . of *Fibonacci numbers.* Then f is determined by the conditions $f(0) = f(1) = 1$, and $f(n+ 2) = f(n) + f(n + 1)$. Show that f is a primitive recursive function.

7.10 Show that the truncation function of Example 7.21 is primitive recursive.

7.11 Show that the substitution function of Example 7.22 is primitive recursive.

The remaining problems pertain to Example 7.23 *in the optional section* 7.3. *If you are not at home with the method of proof by mathematical induction, you should probably defer these problems until after that method has been discussed in a later chapter.*

7.12 If f and g are n- (and $n + 2$)-place primitive recursive functions obtainable from the initial functions (zero, successor, identity) by composition, without use of recursion, we have shown in Proposition 7.24 that there are numbers a and b such that for all $x_1, \ldots, x_n$, y, and z we have

$$f(x_1, \ldots, x_n) < x + a, \qquad \text{where } x \text{ is the largest of } x_1, \ldots, x_n$$

$$g(x_1, \ldots, x_n, y, z) < x + b, \qquad \text{where } x \text{ is the largest of } x_1, \ldots, x_n, y, \text{ and } z.$$

Show now that if $h = \text{Pr}[f, g]$, then there is a number c such that for all $x_1, \ldots,$ x_n and y we have

$$h(x_1, \ldots, x_n, y) < cx + c, \qquad \text{where } x \text{ is the largest of } x_1, \ldots, x_n \text{ and } y.$$

7.13 Show that if f and $g_1, \ldots, g_m$ are functions with the property ascribed to the function h in the preceding problem, and if $j = \text{Cn}[f, g_1, \ldots, g_m]$, then j also has that property.

7.14 Show that the multiplication or product function is not obtainable from the initial functions by composition without using recursion at least twice.

7.15 Let β be the function considered in Example 7.23. Consider a natural number s that codes a sequence $(s_0, \ldots, s_m)$ whose every entry s_i is itself a code for a sequence $(b_{i,0}, \ldots, b_{i,ni})$. Call such an s a β-code if the following conditions are met:

if $i < m$, then $b_{i,0} = 2$
if $j < n_0$, then $b_{0,j+1} = b_{0,j}$
if $i < m$ and $j < n_{i+1}$, then $c = b_{i+1,j} \leq n_i$ and $b_{i+1,j+1} = b_{i,c}$.

Call such an s a β-code *covering* (p, q) if $p \leq m$ and $q \leq n_p$.

(a) Show that if s is a β-code covering (p, q), then $b_{p,q} = \beta(p, q)$.

(b) Show that for every p it is the case that for every q there exists a β-code covering (p, q).

7.16 Continuing the preceding problem, show that the relation *Rspqx*, which we define to hold if and only if s is a β-code covering (p, q) and $b_{p,q} = x$, is a primitive recursive relation.

7.17 Continuing the preceding problem, show that β is a recursive (total) function.

8

Equivalent Definitions of Computability

In the preceding several chapters we have introduced the intuitive notion of effective computability, and studied three rigorously defined technical notions of computability: Turing computability, abacus computability, and recursive computability, noting along the way that any function that is computable in any of these technical senses is computable in the intuitive sense. We have also proved that all recursive functions are abacus computable and that all abacus-computable functions are Turing computable. In this chapter we close the circle by showing that all Turing-computable functions are recursive, so that all three notions of computability are equivalent. It immediately follows that Turing's thesis, claiming that all effectively computable functions are Turing computable, is equivalent to Church's thesis, claiming that all effectively computable functions are recursive. The equivalence of these two theses, originally advanced independently of each other, does not amount to a rigorous proof of either, but is surely important evidence in favor of both. The proof of the recursiveness of Turing-computable functions occupies section 8.1. Some consequences of the proof of equivalence of the three notions of computability are pointed out in section 8.2, the most important being the existence of a universal Turing machine, *a Turing machine capable of simulating the behavior of any other Turing machine desired. The optional section 8.3 rounds out the theory of computability by collecting basic facts about* recursively enumerable *sets, sets of natural numbers that can be enumerated by a recursive function. Perhaps the most basic fact about them is that they coincide with the semirecursive sets introduced in the preceding chapter, and hence, if Church's (or equivalently, Turing's) thesis is correct, coincide with the (positively) effectively semidecidable sets.*

8.1 Coding Turing Computations

At the end of Chapter 5 we proved that all abacus-computable functions are Turing computable, and that all recursive functions are abacus computable. (To be quite accurate, the proofs given for Theorem 5.8 did not consider the three processes in their most general form. For instance, we considered only the composition of a two-place function f with two three-place functions g_1 and g_2. But the methods of proof used were perfectly general, and do suffice to show that any recursive function can be computed by some Turing machine.) Now we wish to close the circle by proving, conversely, that every function that can be computed by a Turing machine is recursive.

We will concentrate on the case of a one-place Turing-computable function, though our argument readily generalizes. Let us suppose, then, that f is a one-place function computed by a Turing machine M. Let x be an arbitrary natural number. At the beginning of its computation of $f(x)$, M's tape will be completely blank except for a block of $x + 1$ strokes, representing the argument or input x. At the outset M is scanning the leftmost stroke in the block. When it halts, it is scanning the leftmost stroke in a block of $f(x) + 1$ strokes on an otherwise completely blank tape, representing the value or output $f(x)$. And throughout the computation there are finitely many strokes to the left of the scanned square, finitely many strokes to the right, and at most one stroke in the scanned square. Thus at any time during the computation, if there is a stroke to the left of the scanned square, there is a leftmost stroke to the left, and similarly for the right. We wish to use numbers to code a description of the contents of the tape. A particularly elegant way to do so is through the *Wang coding*. We use *binary notation* to represent the contents of the tape and the scanned square by means of a *pair of natural numbers*, in the following manner:

If we think of the blanks as zeros and the strokes as ones, then the infinite portion of the tape to the left of the scanned square can be thought of as containing a binary numeral (for example, 1011, or 1, or 0) prefixed by an infinite sequence of superfluous 0s. We call this numeral *the left numeral*, and the number it denotes in binary notation *the left number*. The rest of the tape, consisting of the scanned square and the portion to its right, can be thought of as containing a binary numeral *written backwards*, to which an infinite sequence of superfluous 0s is attached. We call this numeral, which appears backwards on the tape, *the right numeral*, and the number it denotes *the right number*. Thus the scanned square contains the digit in the unit's place of the right numeral. We take the right numeral to be written backwards to insure that changes on the tape will always take place in the vicinity of the unit's place of both numerals. If the tape is completely blank, then the left numeral = the right numeral = 0, and the left number = the right number = 0.

8.1 Example (The Wang coding). Suppose the tape looks as in Figure 8-1. Then the left numeral is 11101, the right numeral is 10111, the left number is 29, and the right number is 23. M now moves left, then the new left numeral is 1110, and the new left number is 14, while the new right numeral is 101111, and the new right number is 47.

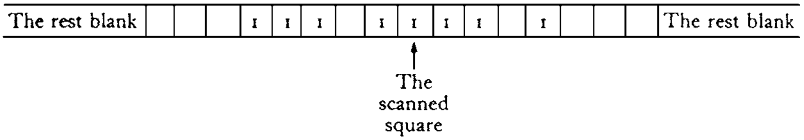


Figure 8-1. A Turing machine tape to be coded.

What are the left and right numbers when M begins the computation? The tape is then completely blank to the left of the scanned square, and so the left numeral is 0 and the left number is 0. The right numeral is 11 ... 1, a block of $x + 1$ digits 1. A sequence of m strokes represents in binary notation

$$2^{m-1} + \cdots + 2^2 + 2 + 1 = 2^m - 1.$$

Thus the right number at the start of M's computation of $f(x)$ will be

$$\mathrm{strt}(x) = 2^{(x+1)} \dot{-} 1.$$

Note that strt is a primitive recursive function.

How do the left and right numbers change when M performs one step in the computation? That depends, of course, on what symbol is being scanned, as well as on what act is performed. How can we determine the symbol scanned? It will be a blank, or 0, if the binary representation of the right number ends in a 0, as is the case when the number is even, and a stroke, or 1, if the binary representation of the right number ends in a 1, as is the case when the number is odd. Thus in either case it will be the remainder on dividing the right number by two, or in other words, if the right number is r, then the symbol scanned will be

$$\mathrm{scan}(r) = \mathrm{rem}(r, 2).$$

Note that scan is a primitive recursive function.

Suppose the act is to *erase*, or put a 0 on, the scanned square. If there was already a 0 present, that is, if $\mathrm{scan}(r) = 0$, there will be no change in the left or right number. If there was a 1 present, that is, if $\mathrm{scan}(r) = 1$, the left number will be unchanged, but the right number will be decreased by 1. Thus if the original left and right numbers were p and r respectively, then the new left and new right numbers will be given by

$$\begin{aligned}\mathrm{newleft}_0(p, r) &= p\\ \mathrm{newrght}_0(p, r) &= r \dot{-} \mathrm{scan}(r).\end{aligned}$$

If instead the act is to *print*, or put a 1 on, the scanned square, there will again be no change in the left number, and there will be no change in the right number either if there was a 1 present. But if there was a 0 present, then the right number will be increased by 1. Thus the new left and new right number will be given by

$$\begin{aligned}\mathrm{newleft}_1(p, r) &= p\\ \mathrm{newrght}_1(p, r) &= r + 1 \dot{-} \mathrm{scan}(r).\end{aligned}$$

Note that all the functions here are primitive recursive.

What happens when M moves left or right? Let p and r be the old (pre-move) left and right numbers, and let p^* and r^* be the new (post-move) left and right numbers. We want to see how p^* and r^* depend upon p, r, and the direction of the move. We consider the case where the machine moves left.

If p is odd, the old numeral ends in a one. If $r = 0$, then the new right numeral is 1, and $r^* = 1 = 2r + 1$. And if $r > 0$, then the new right numeral is obtained from the old by appending a 1 to it at its one's-place end (thus lengthening the numeral); again $r^* = 2r + 1$. As for p^*, if $p = 1$, then the old left numeral is just 1, the new left numeral is 0, and $p^* = 0 = (p \dot{-} 1)/2 = \mathrm{quo}(p, 2)$. And if p is any odd number greater than 1, then the new left numeral is obtained from the old by deleting the 1 in its one's place (thus shortening the numeral), and again $p^* = (p \dot{-} 1)/2 = \mathrm{quo}(p, 2)$. [In Example 8.1, for instance, we had $p = 29$, $p^* = (29 - 1)/2 = 14$,

$r = 23$, $r^* = 2 \cdot 23 + 1 = 47$.] Thus we have established the first of the following two claims:

$$\begin{array}{llll} \text{If } M \text{ moves left} & \text{and } p \text{ is odd} & \text{then } p^* = \text{quo}(p, 2) & \text{and } r^* = 2r + 1 \\ \text{If } M \text{ moves left} & \text{and } p \text{ is even} & \text{then } p^* = \text{quo}(p, 2) & \text{and } r^* = 2r. \end{array}$$

The second claim is established in exactly the same way, and the two claims may be subsumed under the single statement that when M moves left, the new left and right numbers are given by

$$\text{newleft}_2(p, r) = \text{quo}(p, 2)$$
$$\text{newrght}_2(p, r) = 2r + \text{rem}(p, 2).$$

A similar analysis shows that if M moves right, then the new left and right numbers are given by

$$\text{newleft}_3(p, r) = 2p + \text{rem}(r, 2)$$
$$\text{newrght}_3(p, r) = \text{quo}(r, 2).$$

Again all the functions involved are primitive recursive. If we call printing 0, printing 1, moving left, and moving right acts numbers 0, 1, 2, and 3, then the new left number when the old left and right numbers are p and r and the act number is a will be given by

$$\text{newleft}(p, r, a) = \begin{cases} p & \text{if } a = 0 \text{ or } a = 1 \\ \text{quo}(p, 2) & \text{if } a = 2 \\ 2p + \text{rem}(r, 2) & \text{if } a = 3. \end{cases}$$

This again is a primitive recursive function, and there is a similar primitive recursive function newrght(p, r, a) giving the new right number in terms of the old left and right numbers and the act number.

And what are the left and right numbers when M halts? If M halts in *standard position* (or configuration), then the left number must be 0, and the right number must be $r = 2^{f(x)+1} \dot{-} 1$, which is the number denoted in binary notation by a string of $f(x) + 1$ digits 1. Then $f(x)$ will be given by

$$\text{valu}(r) = \text{lg}(r, 2).$$

Here lg is the primitive recursive function of Example 7.11, so valu is also primitive recursive. If we let nstd be the characteristic function of the relation

$$p \neq 0 \vee r \neq 2^{\text{lg}(r, 2)+1} \dot{-} 1$$

then the machine will be in standard position if and only if nstd$(p, r) = 0$. Again, since the relation indicated is primitive recursive, so is the function nstd.

So much, for the moment, for the topic of coding the contents of a Turing tape. Let us turn to the coding of Turing machines and their operations. We discussed the coding of Turing machines in section 4.1, but there we were working with positive integers and here we are working with natural numbers, so a couple of changes will be in order. One of these has already been indicated: we now number the acts 0 through 3 (rather than 1 through 4). The other is equally simple: let us now use

0 for the halted state. A Turing machine will then be coded by a finite sequence whose length is a multiple of four, namely $4k$, where k is the number of states of the machine (not counting the halted state), and with the even-numbered entries (starting with the initial entry, which we count as entry number 0) being numbers ≤ 3 to represent possible acts, while the odd-numbered entries are numbers $\leq k$, representing possible states. Or rather, a machine will be coded by a number coding such a finite sequence.

The instruction as to what *act* to perform when in state q and scanning symbol i will be given by entry number $4(q \dot{-} 1) + 2i$, and the instruction as to what *state* to go into will be given by entry number $4(q \dot{-} 1) + 2i + 1$. For example, the 0th entry tells what act to perform if in the initial state 1 and scanning a blank 0, and the 1st entry what state then to go into; while the 2nd entry tells what act to perform if in initial state 1 and scanning a stroke 1, and the 3rd entry what state then to go into. If the machine with code number m is in state q and the right number is r, so that the symbol being scanned is, as we have seen, given by $\text{scan}(r)$, then the next action to be performed and new state to go into will be given by

$$\begin{aligned}\text{actn}(m, q, r) &= \text{ent}(m, 4(q \dot{-} 1) + 2 \cdot \text{scan}(r))\\ \text{newstat}(m, q, r) &= \text{ent}(m, (4(q \dot{-} 1) + 2 \cdot \text{scan}(r)) + 1).\end{aligned}$$

These are primitive recursive functions.

We have discussed representing the tape contents at a given stage of computation by two numbers p and r. To represent the *configuration* at a given stage of computation, we need also to mention the state q the machine is in. The configuration is then represented by a triple (p, q, r), or by a single number coding such a triple. For definiteness let us use the coding

$$\text{trpl}(p, q, r) = 2^p 3^q 5^r.$$

Then given a code c for the configuration of the machine, we can recover the left, state, and right numbers by

$$\text{left}(c) = \text{lo}(c, 2) \qquad \text{stat}(c) = \text{lo}(c, 3) \qquad \text{rght}(c) = \text{lo}(c, 5)$$

where lo is the primitive recursive function of Example 7.11. Again all the functions here are primitive recursive.

Our next main goal will be to define a primitive recursive function $\text{conf}(m, x, t)$ that will give the code for the configuration after t stages of computation when the machine with code number m is started with input x, that is, is started in its initial state 1 on the leftmost of a block of $x + 1$ strokes on an otherwise blank tape. It should be clear already what the code for the configuration will be at the beginning, that is, after 0 stages of computation. It will be given by

$$\text{inpt}(m, x) = \text{trpl}(0, 1, \text{strt}(x)).$$

What we need to analyse is how to get from a code for the configuration at time t to the configuration at time $t' = t + 1$.

Given the code number m for a machine and the code number c for the configuration at time t, to obtain the code number $c*$ for the configuration at time $t + 1$, we may proceed as follows. First, apply left, stat, and rght to c to obtain the left number, state number, and right number p, q, and r. Then apply actn and newstat to m and r to obtain the number a of the action to be performed, and the number q^* of the state then to enter. Then apply newleft and newrght to p, r, and a to obtain the new left and right numbers p^* and r^*. Finally, apply trpl to p^*, q^*, and r^* to obtain the desired c^*, which is thus given by

$$c^* = \mathrm{newconf}(m, c)$$

where newconf is a composition of the functions left, stat, rght, actn, newstat, newleft, newrght, and trpl, and is therefore a primitive recursive function.

The function conf(m, x, t), giving the code for the configuration after t stages of computation, can then be defined by primitive recursion as follows:

$$\begin{aligned}\mathrm{conf}\,(m, x, 0) &= \mathrm{inpt}(m, x)\\ \mathrm{conf}\,(m, x, t') &= \mathrm{newconf}\,(m, \mathrm{conf}(m, x, t)).\end{aligned}$$

It follows that conf is itself a primitive recursive function.

The machine will be halted when stat(conf(m, x, t)) $= 0$, and will then be halted in standard position if and only if nstd(conf(m, x, t)) $= 0$. Thus the machine will be halted in standard position if and only if stdh$(m, x, t) = 0$, where

$$\mathrm{stdh}(m, x, t) = \mathrm{stat}(\mathrm{conf}(m, x, t)) + \mathrm{nstd}(\mathrm{conf}(m, x, t)).$$

If the machine halts in standard configuration at time t, then the output of the machine will be given by

$$\mathrm{otpt}(m, x, t) = \mathrm{valu}(\mathrm{rght}(\mathrm{conf}(m, x, t))).$$

Note that stdh and otpt are both primitive recursive functions.

The time (if any) when the machine halts in standard configuration will be given by

$$\mathrm{halt}(m, x) = \begin{cases}\text{the least } t \text{ such that } \mathrm{stdh}(m, x, t) = 0 & \text{if such a } t \text{ exists}\\ \text{undefined} & \text{otherwise.}\end{cases}$$

This function, being obtained by minimization from a primitive recursive function, is a recursive partial or total function.

Putting everything together, let $F(m, x) = \mathrm{otpt}(m, x, \mathrm{halt}(m, x))$, a recursive function. Then $F(m, x)$ will be the value of the function computed by the Turing machine with code number m for argument x, if that function is defined for that argument, and will be undefined otherwise. If f is a Turing-computable function, then for some m—namely, for the code number of any Turing machine computing f—we have $f(x) = F(m, x)$ for all x. Since F is recursive, it follows that f is recursive. We have proved:

8.2 Theorem. A function is recursive if and only if it is Turing computable.

The circle is closed.

8.2 Universal Turing Machines

The connection we have established between Turing computability and recursiveness enables us to establish properties of each notion that it would have been more difficult to establish working with that notion in isolation. We begin with one example of this phenomenon pertaining to Turing machines, and one to recursive functions.

8.3 Theorem. The same class of functions are Turing computable whether one defines Turing machines to have a tape infinite in both directions or infinite in only one direction, and whether one requires Turing machines to operate with only one symbol in addition to the blank, or allows them to operate with any finite number.

> *Proof*: Suppose we have a Turing machine M of the kind we have been working with, with a two-way infinite tape. In this chapter we have seen that the total or partial function f computed by M is recursive. In earlier chapters we have seen how a recursive function f can be computed by an abacus machine and hence by a Turing machine simulating an abacus machine. But the Turing machines simulating abacus machines are rather special: according to the problems at the end of Chapter 5, any abacus-computable function can be computed by a Turing machine *that never moves left of the square on which it is started*. Thus we have now shown that for any Turing machine there is another Turing machine computing the same function that uses only the right half of its tape. In other words, if we had begun with a more restrictive notion of Turing machine, where the tape is infinite in one direction only, we would have obtained the same class of Turing-computable functions as with our official, more liberal definition.
>
> Inversely, suppose we allowed Turing machines to operate not only with the blank 0 and the stroke 1, but also with another symbol 2. Then in the proof of the preceding sections we would need to work with *ternary* rather than binary numerals, to code Turing machines by sequences of length a multiple of *six* rather than of four, and make similar minor changes. But with such changes, the proof would still go through, and show that any function computable by a Turing machine of this liberalized kind is still recursive—and therefore was computable by a Turing machine of the original kind already. The result generalizes to more than two symbols in an obvious way: for n symbols counting the blank, we need n-ary numerals and sequences of length a multiple of $2n$.

Similar, somewhat more complicated arguments show that allowing a Turing machine to work on a two-dimensional grid rather than a one-dimensional tape would not enlarge the class of functions that are computable. Likewise the class of functions computable would not be changed if we allowed the use of blank, 0, and 1, and redefined computations so that inputs and outputs are to be given in binary rather than stroke notation. That class is, as is said, *stable under perturbations of definition*, one mark of a *natural* class of objects.

8.4 Theorem (Kleene normal form theorem). Every recursive total or partial function can be obtained from the basic functions (zero, successor, identity) by composition, primitive recursion, and minimization, *using this last process no more than once*.

Proof: Suppose we have a recursive function f. We have seen in earlier chapters that f is computable by an abacus machine and hence by some Turing machine M. We have seen in this chapter that if m is the code number of M, then $f(x) = F(m, x)$ for all x, from which it follows that f can be obtained by composition from the constant function const_m, the identity function id, and the function F [namely, $f(x) = F(\text{const}_m(x), \text{id}(x))$, and therefore $f = \text{Cn}[F, \text{const}_m, \text{id}]$.] Now const_m and id are primitive recursive, and so obtainable from basic functions by composition and primitive recursion, without use of minimization. As for F, reviewing its definition, we see that minimization was used just once (namely, in defining $\text{halt}(m, x)$). Thus *any* recursive function f can be obtained using minimization only once.

An $(n + 1)$-place recursive function F with the property that for every n-place recursive function f there is an m such that

$$f(x_1, \ldots, x_n) = F(m, x_1, \ldots, x_n)$$

is called a *universal* function. We have proved the existence of a two-place universal function, and remarked at the outset that our arguments would apply also to functions with more places. A significant property of our two-place universal function, shared by the analogous many-place universal functions, is that its graph is a semirecursive relation. For $F(m, x) = y$ if and only if the machine with code number m, given input x, eventually halts in standard position, giving output y, which is to say, if and only if

$$\exists t(\text{stdh}(m, x, t) = 0 \,\&\, \text{otpt}(m, x, t) = y).$$

Since what follows the existential quantifier here is a primitive recursive relation, the graph relation $F(m, x) = y$ is obtainable by existential quantification from a primitive recursive relation, and therefore is semirecursive, as asserted. Thus we have the following.

8.5 Theorem. For every k there exists a universal k-place recursive function (whose graph relation is semirecursive).

This theorem has several substantial corollaries in the theory of recursive functions, but as these will not be essential in our later work, we have relegated them to an optional final section—in effect, an appendix—to this chapter. In the closing paragraphs of the present section, we wish to point out the implications of Theorem 8.5 for the theory of Turing machines. Of course, in the definition of universal function and the statement of the foregoing theorem we could have said ‘Turing-computable function’ in place of ‘recursive function’, since we now know these come to the same thing.

A Turing machine for computing a universal function is called a *universal Turing machine*. If U is such a machine (for, say, $k = 1$), then for any Turing machine M we like, the value computed by M for a given argument x will also be computed by U given a code m for M as a further argument in addition to x. Historically, as we have already mentioned, the theory of Turing computability (including the proof of the existence of a universal Turing machine) was established before (indeed, a decade or more before) the age of general-purpose, programmable computers, and in fact formed a significant part of the theoretical background for the development of such computers. We can now say more specifically that the theorem that there exists a universal Turing machine, together with Turing's thesis that all effectively computable functions are Turing computable, heralded the arrival of the computer age by giving the first theoretical assurance that in principle *a general-purpose computer could be designed that could be made to mimic* any *special-purpose computer desired, simply by giving it coded instructions as to what machine it is to mimic as an additional input along with the arguments of the function we want computed.*

8.3* Recursively Enumerable Sets

An immediate consequence of Theorem 8.5 is the following converse to Proposition 7.17.

8.6 Corollary (Second graph principle). The graph relation of a recursive function is semirecursive.

Proof: If f is a recursive (total or partial) function, then there is an m such that $f(x) = F(m, x)$, where F is the universal function of the preceding section. For the graph relation of f we have

$$f(x) = y \leftrightarrow F(m, x) = y.$$

Hence, the graph relation of f is a section, in the sense of Problem 7.1, of the graph relation of F, which is semirecursive, and is therefore itself semirecursive.

At the beginning of this book we defined a set to be enumerable if it is the range of a total or partial function on the positive integers; and clearly we could have said 'natural numbers' in place of 'positive integers'. We now define a set of natural numbers to be *recursively enumerable* if it is the range of a total or partial *recursive* function on natural numbers. It turns out that we could say 'domain' here instead of 'range' without changing the class of sets involved, and that this class is one we have already met with under another name: the semirecursive sets. In the literature the name 'recursively enumerable' or 'r.e.' is more often used than 'semirecursive', though the two come to the same thing.

8.7 Corollary. Let A be a set of natural numbers. Then the following conditions are equivalent:

(a) A is the range of some recursive total or partial function.
(b) A is the domain of some recursive total or partial function.
(c) A is semirecursive.

Proof: First suppose A is semirecursive. Then the relation

$$Rxy \leftrightarrow Ax \,\&\, x = y$$

is semirecursive, since A is semirecursive, the identity relation is semirecursive, and semirecursive relations are closed under conjunction. But the relation R is the graph relation of the restriction of the identity function to A, that is, of the function

$$\mathrm{id}_A(x) = \begin{cases} x & \text{if } Ax \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Since the graph relation is semirecursive, the function is recursive by Proposition 7.17. And A is both the range and the domain of id_A. Hence A is both the range of a recursive partial function and the domain of such a function.

Now suppose f is a recursive partial or total function. Then by Corollary 8.6 the graph relation $f(x) = y$ is semirecursive. Since semirecursive relations are closed under existential quantification, the following sets are also semirecursive:

$$Ry \leftrightarrow \exists x(f(x) = y)$$
$$Dx \leftrightarrow \exists y(f(x) = y).$$

But these sets are precisely the range and the domain of f. Thus the range and domain of any recursive function are semirecursive.

We have said quite a bit about recursively enumerable (or equivalently, semirecursive) sets without giving any examples of such sets. Of course, in a sense we *have* given many examples, since every recursive set is recursively enumerable. But are there any other examples? We are at last in a position to prove that there are.

8.8 Corollary. There exists a recursively enumerable set that is not recursive.

Proof: Let F be the universal function of Theorem 8.5, and let A be the set of x such that $F(x, x) = 0$. Since the graph relation of F is semirecursive, this set is also semirecursive (or equivalently, recursively enumerable). If it were recursive, its complement would also be recursive, which is to say, the characteristic function c of its complement would be a recursive function. But then, since F is a universal function, there would be an m such that $c(x) = F(m, x)$ for all x, and in particular, $c(m) = F(m, m)$. But since c is the characteristic function of the complement of A, we have $c(m) = 0$ if and only if m is *not* in A, which, by the definition of A, means if and only if $F(m, m)$ is *not* $= 0$ (is either undefined, or defined and > 0). This is a contradiction, showing that A cannot be recursive.

When we come to apply computability theory to logic, we are going to find that there are many more natural examples than this of recursively enumerable sets that are not recursive.

Problems

8.1 We proved Theorem 8.2 for one-place functions. For two-place (or many-place) functions, the only difference in the proof would occur right at the beginning, in defining the function strt. What is the right number at the beginning of a computation with arguments x_1 and x_2?

8.2 Suppose we liberalized our definition of Turing machine to allow the machine to operate on a two-dimensional grid, like graph paper, with vertical up and down actions as well as horizontal left and right actions. Describe some reasonable way of coding a configuration of such a machine.

The remaining problems pertain to the optional section 8.3.

8.3 The (*positive*) *semicharacteristic function* of a set A is the function c such that $c(a) = 1$ if a is in A, and $c(a)$ is undefined otherwise. Show that a set A is recursively enumerable if and only if its semicharacteristic function is recursive.

8.4 A two-place relation S is called *recursively enumerable* if there are two recursive total or partial functions f and g with the same domain such that for all x and y we have $Sxy \leftrightarrow \exists t(f(t) = x \,\&\, g(t) = y)$. Show that S is recursively enumerable if and only if the set of all $J(x, y)$ such that Sxy is recursively enumerable, where J is the usual primitive recursive pairing function.

8.5 Show that any recursively enumerable set A can be defined in the form $Ay \leftrightarrow \exists w\, Ryw$ for some *primitive* recursive relation R.

8.6 Show that any nonempty recursively enumerable set A is the range of some *primitive* recursive function.

8.7 Show that any infinite recursively enumerable set A is the range of some *one-to-one* recursive total function.

8.8 A one-place total function f on the natural numbers is *monotone* if and only if whenever $x < y$ we have $f(x) < f(y)$. Show that if A is the range of a monotone recursive function, then A is recursive.

8.9 A pair of recursively enumerable sets A and B are called *recursively inseparable* if they are disjoint, but there is no recursive set C that contains A and is disjoint from B. Show that a recursively inseparable pair of recursively enumerable sets exists.

8.10 Give an example of a recursive partial function f such that f cannot be extended to a recursive total function, or in other words, such that there is no recursive total function g such that $g(x) = f(x)$ for all x in the domain of f.

8.11 Let R be a recursive relation, and A the recursively enumerable set given by $Ax \leftrightarrow \exists w\ Rxw$. Show that if A is not recursive, then for any recursive total function f there is an x in A such that the least ‘witness’ that x is in A (that is, the least w such that Rxw) is greater than $f(x)$.

8.12 Show that if f is a recursive *total* function, then there is a sequence of functions $f_1, \ldots, f_n$ with last item $f_n = f$, such that each either is a basic function (zero, successor, identity) or is obtainable from earlier functions in the sequence by composition, primitive recursion, or minimization, *and all functions in the sequence are total.*

9

A Précis of First-Order Logic: Syntax

This chapter and the next contain a summary of material, mainly definitions, needed for later chapters, of a kind that can be found expounded more fully and at a more relaxed pace in introductory-level logic textbooks. Section 9.1 gives an overview of the two groups of notions from logical theory that will be of most concern: notions pertaining to formulas *and* sentences*, and notions pertaining to* truth under an interpretation. *The former group of notions, called* syntactic, *will be further studied in section 9.2, and the latter group, called* semantic*, in the next chapter.*

9.1 First-Order Logic

Logic has traditionally been concerned with relations among statements, and with properties of statements, that hold by virtue of 'form' alone, regardless of 'content'. For instance, consider the following argument:

(1) A mother or father of a person is an ancestor of that person.
(2) An ancestor of an ancestor of a person is an ancestor of that person.
(3) Sarah is the mother of Isaac, and Isaac is the father of Jacob.
(4) Therefore, Sarah is an ancestor of Jacob.

Logic teaches that the premisses (1)–(3) (*logically*) *imply* or have as a (*logical*) *consequence* the conclusion (4), because in any argument of the same form, if the premisses are true, then the conclusion is true. An example of another argument of the same form would be the following:

(5) A square or cube of a number is a power of that number.
(6) A power of a power of a number is a power of that number.
(7) Sixty-four is the cube of four and four is the square of two.
(8) Therefore, sixty-four is a power of two.

Modern logic represents the forms of statements by certain algebraic-looking symbolic expressions called *formulas*, involving special signs. The special signs we are going to be using are shown in Table 9-1.

Table 9-1. *Logical symbols*

$\sim$	Negation	'not . . . '
&	Conjunction	'. . . and . . . '
$\vee$	Disjunction	'. . . or . . . '
$\rightarrow$	Conditional	'if . . . then . . . '
$\leftrightarrow$	Biconditional	'. . . if and only if . . . '
$\forall x, \forall y, \forall z, \ldots$	Universal quantification	'for every x', 'for every y', 'for every z', . . .
$\exists x, \exists y, \exists z, \ldots$	Existential quantification	'for some x', 'for some y', 'for some z', . . .

In this symbolism, the form shared by the arguments (1)–(4) and (5)–(8) above might be represented as follows:

(9) $\forall x \forall y((\mathbf{P}yx \vee \mathbf{Q}yx) \rightarrow \mathbf{R}yx)$

(10) $\forall x \forall y(\exists z(\mathbf{R}yz \,\&\, \mathbf{R}zx) \rightarrow \mathbf{R}yx)$

(11) **Pab** & **Qbc**

(12) **Rac**

Content is put back into the forms by providing an *interpretation*. Specifying an interpretation involves specifying what sorts of things the xs and ys and zs are supposed to stand for, which of these things **a** and **b** and **c** are supposed to stand for, and which relations among these things **P** and **Q** and **R** are supposed to stand for. One interpretation would let the xs and ys and zs stand for (human) persons, **a** and **b** and **c** for the persons Sarah and Isaac and Jacob, and **P** and **Q** and **R** for the relations among persons of mother to child, father to child, and ancestor to descendent, respectively. With this interpretation, (9) and (10) would amount to the following more stilted versions of (1) and (2):

(13) For any person x and any person y, if either y is the mother of x or y is the father of x, then y is an ancestor of x.

(14) For any person x and any person y, if there is a person z such that y is an ancestor of z and z is an ancestor of x, then y is an ancestor of x.

(11) and (12) would amount to (3) and (4).

A different interpretation would let the xs and ys and zs stand for (natural) numbers, **a** and **b** and **c** for the numbers sixty-four and four and two, and **P** and **Q** and **R** for the relations of the cube or the square or a power of a number to that number, respectively. With this interpretation, (9)–(12) would amount to (5)–(8). We say that (9)–(11) imply (12) because in *any* interpretation in which (9)–(11) come out true, (12) comes out true.

Our goal in this chapter will be to make the notions of formula and interpretation rigorous and precise. In seeking the degree of clarity and explicitness that will be needed for our later work, the first notion we need is a division of the symbols that may occur in formulas into two sorts: *logical* and *nonlogical*. The logical symbols are the logical operators we listed above, the *connective symbols* (the tilde $\sim$, the ampersand &, the wedge $\vee$, the arrow $\rightarrow$, the double arrow $\leftrightarrow$), the *quantifier symbols* (the inverted ay $\forall$, the reversed ee $\exists$), plus the *variables* $x, y, z, \ldots$ that go with the quantifiers, plus left and right parentheses and commas for punctuation.

The nonlogical symbols are to begin with of two sorts: *constants* or *individual symbols*, and *predicates* or *relation symbols*. Each predicate comes with a fixed positive number of *places*. (It is possible to consider zero-place predicates, called *sentence letters*, but we have no need for them here.) As we were using them above, **a** and **b** and **c** were constants, and **P** and **Q** and **R** were two-place predicates.

Especially though not exclusively when dealing with mathematical material, some further apparatus is often necessary or useful. Hence we often include one more logical symbol, a special two-place predicate, the *identity symbol* or equals sign $=$, for '... is (the very same thing as) ...'. To repeat, the equals sign, though a two-place predicate, is counted as a logical symbol, but it is the only exception: all other predicates count as nonlogical symbols. Also, we often include one more category of nonlogical symbols, called *function symbols*. Each function symbol comes with a fixed number of *places*. (Occasionally, constants are regarded as zero-place function symbols, though usually we don't so regard them.)

We conscript the word 'language' to mean an enumerable set of nonlogical symbols. A special case is the *empty language* $L_\varnothing$, which is just the empty set under another name, with no nonlogical symbols. Here is another important case.

9.1 Example (The language of arithmetic). One language that will be of especial interest to us in later chapters is called the *language of arithmetic*, L^*. Its nonlogical symbols are the constant zero $\mathbf{0}$, the two-place predicate less-than $<$, the one-place function symbol successor $'$, and the two-place function symbols addition $+$ and multiplication $\cdot$.

Intuitively, *formulas* are just the sequences of symbols that correspond to grammatically well-formed sentences of English. Those that, like (9)–(12) above, correspond to English sentences that make a complete statement capable of being true or false are called *closed* formulas. Those that, like $(\mathbf{P}yz \vee \mathbf{Q}yx)$, correspond to English sentences involving unidentified xs and ys and zs that would have to be identified before the sentences could be said to be true or false, are called *open* formulas.

The *terms* are sequences of symbols, such as $\mathbf{0}$ or $\mathbf{0} + \mathbf{0}$ or x or x'', that correspond to grammatically well-formed phrases of English of the kind that grammarians call 'singular noun phrases'. The *closed* terms are the ones that involve no variables, and the *open* terms are the ones that involve variables whose values would have to be specified before the term as a whole could be said to have a denotation. When no function symbols are present, the only closed terms are constants, and the only open terms are variables. When function symbols are present, the closed terms also include such expressions as $\mathbf{0} + \mathbf{0}$, and the open terms such expressions as x''.

The formulas and terms of a given language are simply the ones all of whose nonlogical symbols belong to that language. Since languages are enumerable and each formula of a language is a finite string of symbols from the language plus variables and logical symbols, the set of formulas is enumerable, too. (One might at first guess that the empty language would have no formulas, but at least when identity is present, in fact it has infinitely many, among them $\forall x\, x = x$, $\forall y\, y = y$, $\forall z\, z = z$, and so on.)

An *interpretation* $\mathcal{M}$ for a language L consists of two components. On the one hand, there is a nonempty set $|\mathcal{M}|$ called the *domain* or *universe of discourse* of the interpretation, the set of things $\mathcal{M}$ interprets the language to be talking about. When we say 'for every x' or 'for some x', what we mean, according to interpretation $\mathcal{M}$, is 'for every x in $|\mathcal{M}|$' or 'there exists an x in $|\mathcal{M}|$'. On the other hand, there is for each nonlogical symbol a *denotation* assigned to it. For a constant c, the denotation $c^{\mathcal{M}}$ is to be some individual in the domain $|\mathcal{M}|$. For an n-place nonlogical predicate R, the denotation $R^{\mathcal{M}}$ is to be some n-place relation on $|\mathcal{M}|$ (which is officially just a set of n-tuples of elements of $|\mathcal{M}|$, a one-place relation being simply a subset of $|\mathcal{M}|$).

For example, for the language L_G with constants **a** and **b** and **c** and two-place predicates **P** and **Q** and **R**, the genealogical interpretation $\mathcal{G}$ of L_G indicated above would now be described by saying that the domain $|\mathcal{G}|$ is the set of all persons, $\mathbf{a}^{\mathcal{G}}$ is Sarah, $\mathbf{b}^{\mathcal{G}}$ is Isaac, $\mathbf{c}^{\mathcal{G}}$ is Jacob, $\mathbf{P}^{\mathcal{G}}$ is set of ordered pairs of persons where the first is the mother of the second, and analogously for $\mathbf{Q}^{\mathcal{G}}$ and $\mathbf{R}^{\mathcal{G}}$. Under this interpretation, the open formula $\exists z(\mathbf{P}yz \,\&\, \mathbf{Q}zx)$ amounts to 'y is the paternal grandmother of x', while $\exists z(\mathbf{Q}yz \,\&\, \mathbf{P}zx)$ amounts to 'y is the maternal grandfather of x'. The closed formula $\sim\exists x\ \mathbf{P}xx$ amounts to 'no one is her own mother', which is true, while $\exists x\ \mathbf{Q}xx$ amounts to 'someone is his own father', which is false.

When the identity symbol is present, it is *not* treated like the other, nonlogical predicates: one is *not* free to assign it an arbitrary two-place relation on the domain as its denotation; rather, its denotation must be the genuine identity relation on that domain, the relation each thing bears to itself and to nothing else. When function symbols are present, for an n-place function symbol f, the denotation $f^{\mathcal{M}}$ is an n-argument function from $|\mathcal{M}|$ to $|\mathcal{M}|$.

9.2 Example (The standard interpretation of the language of arithmetic). One interpretation that will be of especial interest to us in later chapters is called the *standard interpretation* $\mathcal{N}^*$ of the language of the language of arithmetic L^*. Its domain $|\mathcal{N}^*|$ is the set of natural numbers; the denotation $\mathbf{0}^{\mathcal{N}^*}$ of the cipher $\mathbf{0}$ is the number zero; the denotation $<^{\mathcal{N}^*}$ of the less-than sign is the usual strict less-than order relation; the denotation $'^{\mathcal{N}^*}$ of the accent is the successor function, which takes each number to the next larger number; and the denotations $+^{\mathcal{N}^*}$ and $\cdot^{\mathcal{N}^*}$ of the plus sign and times sign are the usual addition and multiplication functions. Then such an open term as $x \cdot y$ would stand for the product of x and y, whatever they are; while such a closed term as $\mathbf{0}''$ would stand for the successor of the successor of zero, which is to say the successor of one, which is to say two. And such a closed formula as

(15) $\forall x \forall y(x \cdot y = \mathbf{0}'' \rightarrow (\, x = \mathbf{0}'' \vee y = \mathbf{0}''))$

would stand for 'for every x and every y, if the product of x and y is two, then either x is two or y is two' or 'a product is two only if one of the factors is two'. This happens to be true (given that our domain consists of natural numbers, with no negatives or fractions). Other closed formulas that come out true on this interpretation include the following:

(16) $\forall x \exists y(x < y \& \sim\exists z(x < z \& z < y))$

(17) $\forall x(x < x' \;\&\; \sim\exists z(x < z \& z < x'))$.

Here (16) says that for any number x there is a *next larger* number, and (17) that x' is precisely this next larger number.

(For the empty language $L_\varnothing$, there are no nonlogical symbols to be assigned denotations, but an interpretation must still specify a domain, and that specification makes a difference as to truth for closed formulas involving $=$. For instance, $\exists x \exists y \sim x = y$ will be true if the domain has at least two distinct elements, but false if it has only one.)

Closed formulas, which are also called *sentences*, have *truth values*, true or false, when supplied with an interpretation. But they may have different truth values under different interpretations. For our original example (9)–(12), on the genealogical interpretation we have since named $\mathcal{G}$ (and equally on the alternative arithmetical interpretation that we have left nameless) all four sentences came out true. But alternative interpretations are possible. For instance, if we kept everything else the same as in the genealogical interpretation, but took **R** to denote the relation of descendant to ancestor rather than vice versa, (10) and (11) would remain true, but (9) and (12) would become false: descendants of descendants are descendants, but parents and grandparents are not descendants. Various other combinations are possible. What one will *not* find is any interpretation that makes (9)–(11) all true, but (12) false. Precisely that, to repeat, is what is meant by saying that (9)–(11) *imply* (12).

9.3 Example (Alternative interpretations of the language of arithmetic). For the language of arithmetic, there is an alternative interpretation $\mathcal{Q}$ in which the domain is the nonnegative rational numbers, but the denotation of **0** is still zero, the denotation of $'$ is still the function that adds one to a number, the denotations of $+$ and $\cdot$ are the usual addition and multiplication operations, and the denotation of $<$ is still the less-than relation among the numbers in question. On this interpretation, (16) and (17) above are both false (because there are lots of rational numbers between x and any larger y in general, and lots of rational numbers between x and x plus one in particular). There is another alternative interpretation $\mathcal{P}$ in which the domain consists of the nonnegative half integers 0, $^1/_2$, 1, $1^1/_2$, 2, $2^1/_2$, 3, and so on, but the denotation of **0** is still zero, the denotation of $'$ is still the function that adds one to a number, the denotation of $+$ is still the usual addition operation, and the denotation of $<$ is still the less-than relation among the numbers in question. (Multiplication cannot be interpreted in the usual way, since a product of two half integers is not in general a half integer, but for purposes of this example it does not matter how multiplication is interpreted.) On this interpretation, (16) would be true (because there is no half integer between x and $y = x$ plus one-half), but (17) would be false (because there is a half integer between x and x plus one, namely x plus one-half). What you *won't* find is an interpretation that makes (17) true but (16) false. And again, that is what it means to say that (16) is a consequence of (17).

The explanations given so far provide part of the precision and rigor that will be needed in our later work, but only part. For they still rely on an intuitive understanding of what it is to be a sentence of a language, and what it is for a sentence be true in an interpretation. There are two reasons why we want to avoid this reliance on intuition. The first is that when we come to apply our work on computability to logic, we are going to want the notion of sentence to be so precisely defined that a *machine* could tell whether or not a given string of symbols is a sentence. The second is that the notion of truth was historically under a certain cloud of suspicion, owing to the occurrence of certain contradictions, euphemistically called 'paradoxes', such as the ancient *Epimenides* or *liar* paradox: If I say, 'what I am now saying is not true', is what I am saying true? We are therefore going to want to give, for sentences of the kind of formal language we are considering, a definition of truth just as rigorous as the definition of any other notion in mathematics, making the notion of truth, as applied to the kind of formal language we are considering, as respectable as any other mathematical notion.

The next section will be devoted to giving precise and rigorous definitions of the notions of formula and sentence, and more generally to giving definitions of notions pertaining to *syntax*, that is, pertaining to the internal structure of formulas. The next chapter will be devoted to giving the definition of truth, and more generally to giving definitions of notions pertaining to *semantics*, that is, pertaining to the external interpretation of formulas.

9.2 Syntax

Officially we think of ourselves as working for each $k > 0$ with a fixed denumerable stock of k-place predicates:

$$\begin{array}{cccc} A_0^1 & A_1^1 & A_2^1 & \cdots \\ A_0^2 & A_1^2 & A_2^2 & \cdots \\ A_0^3 & A_1^3 & A_2^3 & \cdots \\ \vdots & \vdots & \vdots & \end{array}$$

and with a fixed denumerable stock of constants:

$$f_0^0 \quad f_1^0 \quad f_2^0 \quad \ldots .$$

When function symbols are being used, we are also going to want for each $k > 0$ a fixed denumerable stock of k-place function symbols:

$$\begin{array}{cccc} f_0^1 & f_1^1 & f_2^1 & \cdots \\ f_0^2 & f_1^2 & f_2^2 & \cdots \\ f_0^3 & f_1^3 & f_2^3 & \cdots \\ \vdots & \vdots & \vdots & . \end{array}$$

Any language will be a subset of this fixed stock. (In some contexts in later chapters where we are working with a language L we will want to be able to assume that there are infinitely many constants available that have not been used in L. This is no real difficulty, even if L itself needs to contain infinitely many constants, since we can either add the new constants to our basic stock, or assume that L used only every other constant of our original stock to begin with.)

We also work with a fixed denumerable stock of variables:

$$v_0 \quad v_1 \quad v_2 \quad \ldots .$$

Thus the more or less traditional $\mathbf{0}$ and $<$ and $'$ and $+$ and $\cdot$ we have been writing—and in practice, are going to continue to write—are in principle to be thought of as merely nicknames for f_0^0 and A_0^2 and f_0^1 and f_0^2 and f_1^2; while even writing x and y and z rather than v_i and v_j and v_k, we are using nicknames, too.

The official definition of the notion of formula begins by defining the notion of an *atomic formula*, which will be given first for the case where identity and function symbols are absent, then for the case where they are present. (If sentence letters were admitted, they would count as atomic formulas, too; but, as we have said, we generally are not going to admit them.) If identity and function symbols are absent, then an *atomic formula* is simply a string of symbols $R(t_1, \ldots, t_n)$ consisting of a predicate, followed by a left parenthesis, followed by n constants or variables, where n is the number of places of the predicate, with commas separating the successive terms, all followed by a right parenthesis. Further, if F is a formula, then so is its *negation* $\sim F$, consisting of a tilde followed by F. Also, if F and G are formulas, then so is their *conjunction* $(F \mathbin{\&} G)$, consisting of a left parenthesis, followed by F, which is called the *left* or *first conjunct*, followed by the ampersand, followed by G, which is called the *right* or *second conjunct*, followed by a right parenthesis. Similarly for disjunction. Also, if F is a formula and x is a variable, the *universal quantification* $\forall xF$ is a formula, consisting of an inverted ay, followed by x, followed by F. Similarly for existential quantification.

And that is all: the definition of (*first-order*) *formula* is completed by saying that anything that is a (first-order) formula can be built up from atomic formulas in a sequence of finitely many steps—called a *formation sequence*—by applying negation, junctions, and quantifications to simpler formulas. (Until a much later chapter, where we consider what are called *second-order* formulas, 'first-order' will generally be omitted.)

Where identity is present, the atomic formulas will include ones of the kind $=(t_1, t_2)$. Where function symbols are present, we require a preliminary definition of terms. Variables and constants are *atomic* terms. If f is an n-place function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term. And that is all: the definition of *term* is completed by stipulating that anything that is a term can be built up from atomic terms in a sequence of finitely many steps—called a *formation sequence*—by applying function symbols to simpler terms. Terms that contain variables are said to be *open*, while terms that do not are said to be *closed*. An atomic formula is now something of the type $R(t_1, \ldots, t_n)$ where the t_i may be any terms, not just constants or variables; but otherwise the definition of formula is unchanged.

Note that officially predicates are supposed to be written in front of the terms to which they apply, so writing $x < y$ rather than $<(x, y)$ is an unofficial colloquialism. We make use of several more such colloquialisms below. Thus we sometimes omit the parentheses around and commas separating terms in atomic formulas, and we generally write multiple conjunctions like $(A \mathbin{\&} (B \mathbin{\&} (C \mathbin{\&} D)))$ simply as $(A \mathbin{\&} B \mathbin{\&} C \mathbin{\&} D)$, and similarly for disjunctions, as well as sometimes omitting the outer parentheses on conjunctions and disjunctions $(F \mathbin{\&} G)$ and $(F \vee G)$ when these stand alone rather than as parts of more complicated formulas. All this is slang, from the official point of view. Note that $\rightarrow$ and $\leftrightarrow$ have been left out of the official

Table 9-2. *Some terms of the language of arithmetic*

v_0	x
f_0^0	$\mathbf{0}$
$f_0^1(f_0^0)$	$\mathbf{1}$
$f_0^1(f_0^1(f_0^0))$	$\mathbf{2}$
$f_1^2(f_0^1(f_0^1(f_0^0)), v_0)$	$\mathbf{2} \cdot x$
$f_0^2(f_1^2(f_0^1(f_0^1(f_0^0)), v_0), f_1^2(f_0^1(f_0^1(f_0^0)), v_0))$	$\mathbf{2} \cdot x + \mathbf{2} \cdot x$

language entirely: $(F \to G)$ and $(F \leftrightarrow G)$ are to be considered unofficial abbreviations for $(\sim F \vee G)$ and $((\sim F \vee G)\ \&\ (\sim G \vee F))$. In connection with the language of arithmetic we allow ourselves two further such abbreviations, the bounded quantifiers $\forall y < x$ for $\forall y(y < x \to \ldots)$ and $\exists y < x$ for $\exists y(y < x\ \&\ \ldots)$.

Where identity is present, we also write $x = y$ and $x \neq y$ rather than $=(x, y)$ and $\sim=(x, y)$. Where function symbols are present, they also are supposed to be written in front of the terms to which they apply. So our writing x' rather than $'(x)$ and $x + y$ and $x \cdot y$ rather than $+(x, y)$ and $\cdot(x, y)$ is a colloquial departure from officialese. And if we adopt—as we do—the usual conventions of algebra that allow us to omit certain parenthesis, so that $x + y \cdot z$ is conventionally understood to mean $x + (y \cdot z)$ rather than $(x + y) \cdot z$ without our having to write the parentheses in explicitly, that is another such departure. And if we go further—as we do—and abbreviate $\mathbf{0}'$, $\mathbf{0}''$, $\mathbf{0}'''$, ..., as $\mathbf{1}, \mathbf{2}, \mathbf{3}, \ldots$, that is yet another departure.

Some terms of L^* in official and unofficial notation are shown in Table 9-2. The left column is a formation sequence for a fairly complex term.

Some formulas of L^* in official (or rather, semiofficial, since the the terms have been written colloquially) notation are shown in Table 9-3. The left column is a formation sequence for a fairly complex formula.

No one writing about anything, whether about family trees or natural numbers, will write in the official notation illustrated above (any more than anyone filling out a scholarship application or a tax return is going to do the necessary calculations in the rigid format established in our chapters on computability). The reader may well wonder why, if the official notation is so awkward, we don't just take the abbreviated

Table 9-3. *Some formulas of the language of arithmetic*

$A^2{}_0(x, \mathbf{0})$	$x < \mathbf{0}$
$A^2{}_0(x, \mathbf{1})$	$x < \mathbf{1}$
$A^2{}_0(x, \mathbf{2})$	$x < \mathbf{2}$
$A^2{}_0(x, \mathbf{3})$	$x < \mathbf{3}$
$\sim A^2{}_0(x, \mathbf{3}))$	$\sim x < \mathbf{3}$
$(= (x, \mathbf{1}) \vee = (x, \mathbf{2}))$	$x = \mathbf{1} \vee x = \mathbf{2}$
$(= (x, \mathbf{0}) \vee (= (x, \mathbf{1}) \vee = (x, \mathbf{2})))$	$x = \mathbf{0} \vee x = \mathbf{1} \vee x = \mathbf{2}$
$(\sim A^2{}_0(x, \mathbf{3}) \vee (= (x, \mathbf{0}) \vee (= (x, \mathbf{1}) \vee = (x, \mathbf{2}))))$	$x < \mathbf{3} \to (x = \mathbf{0} \vee x = \mathbf{1} \vee x = \mathbf{2})$
$\forall x((\sim A^2{}_0(x, \mathbf{3}) \vee (= (x, \mathbf{0}) \vee (= (x, \mathbf{1}) \vee = (x, \mathbf{2})))))$	$\forall x < \mathbf{3}(x = \mathbf{0} \vee x = \mathbf{1} \vee x = \mathbf{2})$

notation as the official one. The reason is that in proving things *about* the terms and formulas of a language, it is easiest if the language has a very rigid format (just as, in proving things *about* computability, it is easiest if the computations take place in a very rigid format). In writing examples of terms and formulas *in* the language, it is on the contrary easiest if the language has a very flexible format. The traditional strategy of logicians is to make the *official* language about which one proves theorems a very austere and rigid one, and to make the *unofficial* language in which one writes examples a very generous and flexible one. Of course, for the theorems proved about the austere idiom to be applicable to the generous idiom, one has to have confidence that all the abbreviations permitted by the latter but not the former *could in principle* be undone. But there is no need actually to undo them in practice.

The main method of proving theorems about terms and formulas in a language is called *induction on complexity*. We can prove that all formulas have a property by proving

Base Step: Atomic formulas have the property.

Induction Step: If a more complex formula is formed by applying a logical operator to a simpler formula or formulas, then, assuming (as *induction hypothesis*) that the simpler formula or formulas have the property, so does the more complex formula. The induction step will usually be divided into *cases*, according as the operator is $\sim$ or & or $\vee$ or $\forall$ or $\exists$.

Typically the proof will first be given for the situation where identity and function symbols are absent, then for the situation with identity present but function symbols absent, and then for the case with both identity and function symbols present. Identity typically requires very little extra work if any, but where function symbols are present, we generally need to prove some preliminary result about terms, which is also done by induction on complexity: we can prove that all terms have some property by proving that atomic terms have the property, and that if a more complex term is formed by applying a function symbol to simpler terms, then, assuming the simpler terms have the property, so does the more complex term.

The method of proof by induction on complexity is so important that we want to illustrate it now by very simple examples. The following lemma may tell us more than we want to know about punctuation, but is good practice.

9.4 Lemma (Parenthesis lemma). When formulas are written in official notation the following hold:

(a) Every formula ends in a right parenthesis.

(b) Every formula has equally many left and right parentheses.

(c) If a formula is divided into a left part and a right part, both nonempty, then there are at least as many left as right parentheses in the left part, and more if that part contains at least one parenthesis.

Proof: We give first the proof for (a). *Base step:* An atomic formula $R(t_1, \ldots, t_n)$ or $=(t_1, t_2)$ of course ends in a right parenthesis. *Induction step, negation case:* If F ends in a right parenthesis, then so does $\sim F$, since the only new symbol is at the beginning. *Induction step, junction case:* A conjunction $(F \,\&\, G)$ or disjunction $(F \vee G)$ of course ends in a right parenthesis. *Induction step, quantification case:* If

F ends in a right parenthesis, then so do $\forall xF$ or $\exists xF$, for the same reason as in the case of negation, namely, that the only new symbols are at the beginning.

In giving the proof for (b), we allow ourselves to be a little less rigid about the format. We consider first the case where function symbols are absent. First note that an atomic formula $R(t_1, \ldots, t_n)$ or $=(t_1, t_2)$ has equal numbers of left and right parentheses, namely, one of each. Then note that F has equal numbers of left and right parentheses, then so does $\sim F$, since there are no new parentheses. Then note that if F has m of each kind of parenthesis, and G has n of each, then $(F \,\&\, G)$ has $m + n + 1$ of each, the only new ones being the outer ones. The proof for disjunction is the same as for conjunction, and the proofs for quantifications essentially the same as for negation.

If function symbols are present, we need the preliminary result that every term has equally many left and right parentheses. This is established by induction on complexity. An atomic term has equal numbers of left and right parentheses, namely zero of each. The nonatomic case resembles the conjunction case above: if s has m each of left and right parentheses, and t has n each, then $f(s, t)$ has $m + n + 1$ each; and similarly for $f(t_1, \ldots, t_k)$ for values of k other than two. Having this preliminary result, we must go back and reconsider the atomic case in the proof of (b). The argument now runs as follows: if s has m each of left and right parentheses, and t has n each, then $R(s, t)$ has $m + n + 1$ each, and similarly for $R(t_1, \ldots, t_k)$ for values of k other than two. No change is needed in the nonatomic cases of the proof of (b).

In giving the proof for (c), we also first consider the case where function symbols are absent. First suppose an atomic formula $R(t_1, \ldots, t_n)$ or $=(t_1, t_2)$ is divided into a left part λ and a right part ρ, both nonempty. If λ is just R or $=$, it contains zero parentheses of each kind. Otherwise, λ contains the one and only left parenthesis and not the one and only right parenthesis. In either case, (c) holds. Next assume (c) holds for F, and suppose $\sim F$ is divided. If λ consists just of $\sim$, and ρ of all of F, then λ contains zero parentheses of each kind. Otherwise, λ is of the form $\sim\lambda_0$, where λ_0 is a left part of F, and ρ is the right part of F. By assumption, then λ_0 and hence λ has at least as many left as right parentheses, and more if it contains any parentheses at all. Thus in all cases, (c) holds for $\sim F$. Next assume (c) holds for F and G, and suppose $(F \,\&\, G)$ is divided. The possible cases for the left part λ are:

Case1	Case 2	Case 3	Case 4	Case 5	Case 6
(	$(\lambda_0$	$(F$	$(F\,\&$	$(F\,\&\,\lambda_1$	$(F\,\&\,G$

where in case 2, λ_0 is a left part of F, and in case 5, λ_1 is a left part of G. In every case, the part of λ after the initial left parenthesis has at least as many left as right parentheses: obviously in case 1, by the assumption of (c) for F in case (2), by part (b) in case (3), and so on. So the whole left part λ has at least one more left than right parenthesis, and (c) holds for $(F \,\&\, G)$. The proof for disjunction is the same as for conjunction, and the proofs for quantifications essentially the same as for negation. We leave the case where function symbols are present to the reader.

We conclude this section with the official definitions of four more important syntactic notions. First, we officially define a string of consecutive symbols within a given formula to be a *subformula* of the given formula if it is itself a formula. Where function symbols are present, we can similarly define a notion of *subterm*. We stop to note one result about subformulas.

9.5 Lemma (Unique readability lemma).

(a) The only subformula of an atomic formula $R(t_1, \ldots, t_n)$ or $=(t_1, t_2)$ is itself.

(b) The only subformulas of $\sim F$ are itself and the subformulas of F.

(c) The only subformulas of $(F \;\&\; G)$ or $(F \vee G)$ are itself and the subformulas of F and G.

(d) The only subformulas of $\forall xF$ or $\exists xF$ are itself and the subformulas of F.

These assertions may seem obvious, but they only hold because we use enough parentheses. If we used none at all, the disjunction of $F \;\&\; G$ with H, that is, $F \;\&\; G \vee H$, would have the subformula $G \vee H$, which is neither the whole conjunction nor a subformula of either conjunct. Indeed, the whole formula would be the same as the conjunction of F with $G \vee H$, and we would have a serious ambiguity. A rigorous proof of the unique readability lemma requires the parenthesis lemma.

Proof: For (a), a subformula of $R(t_1, \ldots, t_n)$ or $=(t_1, t_2)$ must contain the initial predicate R or $=$, and so, if it is not the whole formula, it will be a left part of it. Being a formula, it must contain (and in fact end in) a parenthesis by 9.4(a), and so, if it is not the whole formula but only a left part, must contain an excess of left over right parentheses by 9.4(c), which is impossible for a formula by 9.4(b).

For (b), a subformula of $\sim F$ that is not a subformula of F must contain the initial negation sign $\sim$, and so, if it is not the whole formula $\sim F$, it will be a left part of it, and from this point the argument is essentially the same as in the atomic case (a).

For (c), we relegate the proof to the problems at the end of the chapter.

For (d), the argument is essentially the same as for (b).

Resuming our series of definitions, second, using the notion of subformula, we state the official definition of which occurrences of a variable x in a formula F are *free* and which are *bound*: an occurrence of variable x is bound if it is part of a subformula beginning $\forall x \ldots$ or $\exists x \ldots$, in which case the quantifier $\forall$ or $\exists$ in question is said to *bind* that occurrence of the variable x, and otherwise the occurrence of the variable x is free. As an example, in

$$x < y \;\&\; \sim\exists z(x < z \;\&\; z < y)$$

all the occurrences of x and y are free, and all the occurrences of z are bound; while in

$$\mathbf{F}x \rightarrow \forall x\mathbf{F}x$$

the first occurrence of x is free, and the other two occurrences of x are bound. [The difference between the role of a free variable x and the role of a bound variable u in a formula like $\forall u\, R(x, u)$ or $\exists u\, R(x, u)$ is not unlike the difference between the roles of x and of u in mathematical expressions like

$$\int_1^x \frac{\mathrm{du}}{u} \qquad \sum_{u=1}^{x} \frac{1}{u}$$

For some readers this analogy may be helpful, and those readers who do not find it so may ignore it.]

In general, any and all occurrences of variables in an atomic formula $R(t_1, \ldots, t_n)$ are free, since there are no quantifiers in the formula; the free occurrences of a variable in a negation $\sim F$ are just the free occurrences in F, since any subformula of $\sim F$ beginning $\forall x$ or $\exists x$ is a proper subformula of $\sim F$ and so a subformula of F; and similarly, the free occurrences of a variable in a junction $(F \,\&\, G)$ or $(F \vee G)$ are just those in F and G; and similarly, the free occurrences of a variable other than x in a quantification $\forall xF$ or $\exists xF$ are just those in F, while of course none of the occurrences of x in $\forall xF$ or $\exists xF$ is free.

Third, using the notion of free and bound occurrence of variables, we state the official definition of the notion of an *instance* of a formula. But before giving that definition, let us mention a convenient notational convention. When we write something like 'Let $F(x)$ be a formula', we are to be understood as meaning 'Let F be a formula in which no variables occur free except x'. That is, we indicate which variables occur free in the formula we are calling F by displaying them immediately after the name F we are using for that formula. Similarly, if we go on to write something like 'Let c be a constant, and consider $F(c)$', we are to be understood as meaning, 'Let c be a constant, and consider the result of substituting c for all free occurrences of x in the formula F'. That is, we indicate what substitution is to be made in the formula we are calling $F(x)$ by making that very substitution in the expression $F(x)$. Thus if $F(x)$ is $\forall y \sim y < x$, then $F(\mathbf{0})$ is $\forall y \sim y \mathbf{<} \mathbf{0}$. Then the official definition of instance is just this: an *instance* of a formula $F(x)$ is any formula of form $F(t)$ for t a closed term. Similar notations apply where there is more than one free variable, and to terms as well as formulas.

Fourth and finally, again using the notion of free and bound occurrence of variables, we state the official definition of *sentence*: a formula is a sentence if no occurrence of any variable in it is free. A *subsentence* is a subformula that is a sentence.

Problems

9.1 Indicate the form of the following argument—traditionally called 'syllogism in Felapton'—using formulas:

(a) No centaurs are allowed to vote.

(b) All centaurs are intelligent beings.

(c) Therefore, some intelligent beings are not allowed to vote.

Do the premisses (a) and (b) in the preceding argument imply the conclusion (c)?

9.2 Consider (9)–(12) of at the beginning of the chapter, and give an alternative to the genealogical interpretation that makes (9) true, (10) false, (11) true, and (12) false.

9.3 Consider a language with a two-place predicate **P** and a one-place predicate **F**, and an interpretation in which the domain is the set of persons, the denotation of **P** is the relation of parent to child, and the denotation of **F** is the set of all female persons. What do the following amount to, in colloquial terms, under that interpretation?

(a) $\exists z \exists u \exists v(u \neq v \,\&\, \mathbf{P}uy \,\&\, \mathbf{P}vy \,\&\, \mathbf{P}uz \,\&\, \mathbf{P}vz \,\&\, \mathbf{P}zx \,\&\, {\sim}\mathbf{F}\, y)$

(b) $\exists z \exists u \exists v(u \neq v \,\&\, \mathbf{P}ux \,\&\, \mathbf{P}vx \,\&\, \mathbf{P}uz \,\&\, \mathbf{P}vz \,\&\, \mathbf{P}zy \,\&\, \mathbf{F}\, y)$

9.4 Officially, a *formation sequence* is a sequence of formulas in which each either is atomic, or is obtained by some earlier formula(s) in the sequence by negation, conjunction, disjunction, or universal or existential quantification. A formation sequence *for a formula* F is just a formation sequence whose last formula is F. Prove that in a formation sequence for a formula F, every subformula of F must appear.

9.5 Prove that every formula F has a formation sequence in which the *only* formulas that appear are subformulas of F, and the number of formulas that appear is no greater than the number of symbols in F.

9.6 Here is an outline of a proof that the only subformulas of $(F \,\&\, G)$ are itself and the subformulas of F and of G. Suppose H is some other kind of subformula. If H does not contain the displayed ampersand, then H must be of one of the two forms:

(a) $(\lambda$ where λ is a left part of F, or

(b) $\rho)$ where ρ is a right part of G.

If H does contain the displayed ampersand, then some subformula of H (possibly H itself) is a conjunction $(A \,\&\, B)$ where A and B are formulas and either

(c) $A = F$ and B is a left part λ of G,

(d) A is a right part ρ of F and $B = G$, or

(e) A is a right part ρ of F and B is a left part λ of G.

Show that (a) and (b) are impossible.

9.7 Continuing the preceding problem, show that (c)–(e) are all impossible.

9.8 Our definition allows the same variable to occur both bound and free in a formula, as in $P(x) \,\&\, \forall x Q(x)$. How could we change the definition to prevent this?

10

A Précis of First-Order Logic: Semantics

This chapter continues the summary of background material on logic needed for later chapters. Section 10.1 studies the notions of truth *and* satisfaction, *and section 10.2 the so-called* metalogical *notions of validity, implication or consequence, and (un)satisfiability.*

10.1 Semantics

Let us now turn from the official definitions of syntactical notions in the preceding chapter to the official definitions of semantic notions. The task must be to introduce the same level of precision and rigor into the definition of truth of a sentence in or on or under an interpretation as we have introduced into the notion of sentence itself. The definition we present is a version or variant of the *Tarski definition* of what it is for a sentence F to be true in an interpretation $\mathcal{M}$, written $\mathcal{M} \models F$. (The double turnstile $\models$ may be pronounced 'makes true'.)

The first step is to define truth for atomic sentences. The official definition will be given first for the case where identity and function symbols are absent, then for the case where they are present. (If sentence letters were admitted, they would be atomic sentences, and specifying which of them are true and which not would be part of specifying an interpretation; but, as we have said, we generally are not going to admit them.) Where identity and function symbols are absent, so that every atomic sentence has the form $R(t_1, \ldots, t_n)$ for some nonlogical predicate R and constants t_i, the definition is straightforward:

$$\text{(1a)} \qquad \mathcal{M} \models R(t_1, \ldots, t_n) \quad \text{if and only if} \quad R^{\mathcal{M}}\big(t_1^{\mathcal{M}}, \ldots, t_n^{\mathcal{M}}\big).$$

The atomic sentence is true in the interpretation just in case the relation that the predicate is interpreted as denoting holds of the individuals that the constants are interpreted as denoting.

When identity is present, there is another kind of atomic sentence for which a definition of truth must be given:

$$\text{(1b)} \qquad \mathcal{M} \models =\!(t_1, t_2) \quad \text{if and only if} \quad t_1^{\mathcal{M}} = t_2^{\mathcal{M}}.$$

The atomic sentence is true in the interpretation just in case the individuals the constants are interpreted as denoting are the same.

When function symbols are present, we need a preliminary definition of the denotation $t^{\mathcal{M}}$ of a closed term t of a language L under an interpretation $\mathcal{M}$. Clauses (1a) and (1b) then apply, where the t_i may be any closed terms, and not just constants. For an atomic closed term, that is, for a constant c, specifying the denotation $c^{\mathcal{M}}$ of c is part of what is meant by specifying an interpretation. For more complex terms, we proceed as follows. If f is an n-place function symbol, then specifying the denotation $f^{\mathcal{M}}$ is again part of what is meant by specifying an interpretation. Suppose the denotations $t_1^{\mathcal{M}}, \ldots, t_n^{\mathcal{M}}$ of terms $t_1, \ldots, t_n$ have been defined. Then we define the denotation of the complex term $f(t_1, \ldots, t_n)$ to be the value of the function $f^{\mathcal{M}}$ that is the denotation of f applied to the individuals $t_1^{\mathcal{M}}, \ldots, t_n^{\mathcal{M}}$ that are the denotations of $t_1, \ldots, t_n$ as arguments:

(1c) $$(f(t_1, \ldots, t_n))^{\mathcal{M}} = f^{\mathcal{M}}\big(t_1^{\mathcal{M}}, \ldots, t_n^{\mathcal{M}}\big).$$

Since every term is built up from constants by applying function symbols a finite number of times, these specifications determine the denotation of every term.

So, for example, in the standard interpretation of the language of arithmetic, since $\mathbf{0}$ denotes the number zero and $'$ denotes the successor function, according to (1c) $\mathbf{0}'$ denotes the value obtained on applying the successor function to zero as argument, which is to say the number one, a fact we have anticipated in abbreviating $\mathbf{0}'$ as $\mathbf{1}$. Likewise, the denotation of $\mathbf{0}''$ is the value obtained on applying the successor function to the denotation of $\mathbf{0}'$, namely one, as argument, and this value is of course the number two, again a fact we have been anticipating in abbreviating $\mathbf{0}''$ as $\mathbf{2}$. Similarly, the denotation of $\mathbf{0}'''$ is three, as is, for instance, the denotation of $\mathbf{0}' + \mathbf{0}''$. No surprises here.

According to (1b), continuing the example, since the denotations of $\mathbf{0}'''$ or $\mathbf{3}$ and of $\mathbf{0}' + \mathbf{0}''$ or $\mathbf{1} + \mathbf{2}$ are the same, $\mathbf{0}''' = \mathbf{0}' + \mathbf{0}''$ or $\mathbf{3} = \mathbf{1} + \mathbf{2}$ is true, while by contrast $\mathbf{0}'' = \mathbf{0}' + \mathbf{0}''$ or $\mathbf{2} = \mathbf{1} + \mathbf{2}$ is false. Again no surprises. According to (1a), further continuing the example, since the denotation of $<$ is the strict less-than relation, and the denotations of $\mathbf{0}'''$ or $\mathbf{3}$ and of $\mathbf{0}' + \mathbf{0}''$ or $\mathbf{1} + \mathbf{2}$ are both three, the atomic sentence $\mathbf{0}''' < \mathbf{0}' + \mathbf{0}''$ or $\mathbf{3} < \mathbf{1} + \mathbf{2}$ is false, while by contrast $\mathbf{0}'' < \mathbf{0}' + \mathbf{0}''$ is true. Yet again, no surprises.

There is only one candidate for what the definition should be in each of the cases of negation and of the two junctions:

(2a) $\mathcal{M} \models {\sim}F$ if and only if not $\mathcal{M} \models F$

(2b) $\mathcal{M} \models (F \mathbin{\&} G)$ if and only if $\mathcal{M} \models F$ and $\mathcal{M} \models G$

(2c) $\mathcal{M} \models (F \vee G)$ if and only if $\mathcal{M} \models F$ or $\mathcal{M} \models G$.

So, for example, in the standard interpretation of the language of arithmetic, since $\mathbf{0} = \mathbf{0}$ and $\mathbf{0} < \mathbf{0}'$ are true while $\mathbf{0} < \mathbf{0}$ is false, we have that $(\mathbf{0} = \mathbf{0} \vee \mathbf{0} < \mathbf{0}')$ is true, $(\mathbf{0} < \mathbf{0} \mathbin{\&} \mathbf{0} = \mathbf{0})$ is false, $(\mathbf{0} < \mathbf{0} \mathbin{\&} (\mathbf{0} = \mathbf{0} \vee \mathbf{0} < \mathbf{0}'))$ is false, and $((\mathbf{0} < \mathbf{0} \mathbin{\&} \mathbf{0} = \mathbf{0}) \vee \mathbf{0} < \mathbf{0}')$ is true. Still no surprises.

One consequence of (2a)–(2c) worth mentioning is that $(F \,\&\, G)$ is true if and only if $\sim(\sim F \vee \sim G)$ is true, and $(F \vee G)$ is true if and only if $\sim(\sim F \,\&\, \sim G)$ is true. We could therefore if we wished drop one of the pair &, $\vee$ from the official language, and treat it as an unofficial abbreviation (for an expression involving $\sim$ and the other of the pair) on a par with → and ↔.

The only slight subtlety in the business arises at the level of quantification. Here is a simple, tempting, and *wrong* approach to defining truth for the case of quantification, called the *substitutional* approach:

$$\begin{array}{lll} \mathcal{M} \models \forall x F(x) & \text{if and only if} & \text{for every closed term } t,\ \mathcal{M} \models F(t) \\ \mathcal{M} \models \exists x F(x) & \text{if and only if} & \text{for some closed term } t,\ \mathcal{M} \models F(t). \end{array}$$

In other words, under this definition a universal quantification is true if and only if every substitution instance is true, and an existential quantification is true if and only if some substitution instance is true. This definition in general produces results not in agreement with intuition, unless it happens that every individual in the domain of the interpretation is denoted by some term of the language. If the domain of the interpretation is enumerable, we could always *expand* the language to add more constants and extend the interpretation so that each individual in the domain is the denotation of one of them. But we cannot do this when the domain is nonenumerable. (At least we cannot do so while continuing to insist that a language is supposed to involve only a finite or enumerable set of symbols. Of course, to allow a 'language' with a nonenumerable set of symbols would involve a considerable stretching of the concept. We will briefly consider this extended concept of 'language' in a later chapter, but for the moment we set it aside.)

10.1 Example. Consider the language L^* of arithmetic and three different interpretations of it: first, the standard interpretation $\mathcal{N}^*$; second, the alternative interpretation $\mathcal{Q}$ we considered earlier, with domain the nonnegative rational numbers; third, the similar alternative interpretation $\mathcal{R}$ with domain the nonnegative real numbers. Now in fact the substitutional approach gives the intuitively correct results for $\mathcal{N}^*$ in all cases. Not so, however, for the other two interpretations. For, all closed terms in the language have the same denotation in all three interpretations, and from this it follows that all closed terms denote natural numbers. And from this it follows that $t + t = \mathbf{1}$ is false for all closed terms t, since there is no natural number that, added to itself, yields one. So on the substitutional approach, $\exists x(x + x = \mathbf{1})$ would come out false on all three interpretations. But intuitively 'there is something (in the domain) that added to itself yields one' is false only on the standard interpretation $\mathcal{N}^*$, and true on the rational and real interpretations $\mathcal{Q}$ and $\mathcal{R}$.

We could try to fix this by adding more constants to the language, so that there is one denoting each nonnegative rational number. If this were done, then on the rational and real interpretations, $\mathbf{1/2} + \mathbf{1/2} = \mathbf{1}$ would come out true, and hence $\exists x(x + x = \mathbf{1})$ would come out true using the substitutional approach, and this particular example of a problem with the substitutional approach would be fixed. Indeed, the substitutional approach would then give the intuitively correct results for $\mathcal{Q}$ in all cases. Not so, however, for $\mathcal{R}$. For, all terms in the language would denote rational numbers, and from this it would follow that $t \cdot t = \mathbf{2}$ is false for all terms t, since there is no rational number that, multiplied by itself, yields two. So on the substitutional approach, $\exists x(x \cdot x = \mathbf{2})$ would come out false. But intuitively, though 'there is something (in the domain) that multiplied by itself yields two' is false on the rational interpretation, it is true on the real interpretation. We could try to fix this by adding yet more terms to the language, but by Cantor's theorem there are too many real numbers to add a term for each of them while keeping the language enumerable.

The *right* definition for the case of quantification has to be a little more indirect. In defining when $\mathcal{M} \models \forall x F(x)$ we do not attempt to extend the given language L so as to provide constants for every individual in the domain of the interpretation at once. In general, that cannot be done without making the language nonenumerable. However, if we consider any particular individual in the domain, we *could* extend the language and interpretation to give *just it* a name, and what we do in defining when $\mathcal{M} \models \forall x F(x)$ is to consider *all possible* extensions of the language and interpretation by adding just one new constant and assigning it a denotation.

Let us say that in the interpretation $\mathcal{M}$ the individual m *satisfies* $F(x)$, and write $\mathcal{M} \models F[m]$, to mean 'if we considered the extended language $L \cup \{c\}$ obtained by adding a new constant c in to our given language L, and if among all the extensions of our given interpretation $\mathcal{M}$ to an interpretation of this extended language we considered the one $\mathcal{M}^c_m$ that assigns c the denotation m, then $F(c)$ would be true':

(3*) $$\mathcal{M} \models F[m] \quad \text{if and only if} \quad \mathcal{M}^c_m \models F(c).$$

(For definiteness, let us say the constant to be added should be the first constant not in L in our fixed enumeration of the stock of constants.)

For example, if $F(x)$ is $x \cdot x = \mathbf{2}$, then on the real interpretation of the language of arithmetic $\sqrt{2}$ satisfies $F(x)$, because if we extended the language by adding a constant c and extended the interpretation by taking c to denote $\sqrt{2}$, then $c \cdot c = \mathbf{2}$ would be true, because the real number denoted by c would be one that, multiplied by itself, yields two. This definition of satisfaction can be extended to formulas with more than one free variable. For instance, if $F(x, y, z)$ is $x \cdot y = z$, then $\sqrt{2}, \sqrt{3}, \sqrt{6}$ satisfy $F(x, y, z)$, because if we added c, d, e denoting them, $c \cdot d = e$ would be true.

Here, then, is the *right* definition, called the *objectual* approach:

(3a) $\mathcal{M} \models \forall x F(x)$ if and only if for every m in the domain, $\mathcal{M} \models F[m]$

(3b) $\mathcal{M} \models \exists x F(x)$ if and only if for some m in the domain, $\mathcal{M} \models F[m]$.

So $\mathcal{R} \models \exists x F(x)$ under the above definitions, in agreement with intuition, even though there is no term t in the actual language such that $\mathcal{R} \models F(t)$, because $\mathcal{R} \models F[\sqrt{2}]$.

One immediate implication of the above definitions worth mentioning is that $\forall x F$ turns out to be true just in case $\sim\exists x \sim F$ is true, and $\exists x F$ turns out to be true just in case $\sim\forall x \sim F$ is true, so it would be possible to drop one of the pair $\forall$, $\exists$ from the official language, and treat it as an unofficial abbreviation.

The method of proof by induction on complexity can be used to prove semantic as well as syntactic results. The following result can serve as a warm-up for more substantial proofs later, and provides an occasion to review the definition of truth clause by clause.

10.2 Proposition (Extensionality lemma).

(a) Whether a sentence A is true depends only on the domain and denotations of the nonlogical symbols in A.

(b) Whether a formula $F(x)$ is satisfied by an element m of the domain depends only on the domain, the denotations of the nonlogical symbols in F, and the element m.

(c) Whether a sentence $F(t)$ is true depends only on the domain, the denotations of the nonlogical symbols in $F(x)$, and the denotation of the closed term t.

Here (a), for instance, means that the truth value of A does not depend on what the nonlogical symbols in A *themselves* are, but only on what their *denotations* are, and does not depend on the denotations of nonlogical symbols *not* in A. (So a more formal statement would be: If we start with a sentence A and interpretation I, and change A to B by changing zero or more nonlogical symbols to others of the same kind, and change I to $\mathcal{J}$, then the truth value of B in $\mathcal{J}$ will be the same as the truth value of A in I provided $\mathcal{J}$ has the same domain as I, $\mathcal{J}$ assigns each unchanged nonlogical symbol the same denotation I did, and whenever a nonlogical symbol S is changed to T, then $\mathcal{J}$ assigns to T the same denotation I assigned to S. The proof, as will be seen, is hardly longer than this formal statement!)

Proof: In proving (a) we consider first the case where function symbols are absent, so the only closed terms are constants, and proceed by induction on complexity. By the atomic clause in the definition of truth, the truth value of an atomic sentence depends only on the denotation of the predicate in it (which in the case of the identity predicate cannot be changed) and the denotations of the constants in it. For a negation $\sim B$, assuming as induction hypothesis that (a) holds for B, then (a) holds for $\sim B$ as well, since by the negation clause in the definition of truth, the truth value of $\sim B$ depends only on the truth value of B. The cases of disjunction and conjunction are similar.

For a universal quantification $\forall x B(x)$, assuming as induction hypothesis that (a) holds for sentences of form $B(c)$, then (b) holds for $B(x)$, for the following reason. By the definition of satisfaction, whether m satisfies $B(x)$ depends on the truth value of $B(c)$ where c is a constant not in $B(x)$ that is assigned denotation m. [For definiteness, we specified which constant was to be used, but the assumption of (a) for sentences of form $B(c)$ implies that it does not matter what constant is used, so long as it is assigned denotation m.] By the induction hypothesis, the truth value of $B(c)$ depends only on the domain and the denotations of the nonlogical symbols in $B(c)$, which is to say, the denotations of the nonlogical symbols in $B(x)$ and the element m that is the denotation of the nonlogical symbol c, just as asserted by (b) for $B(x)$. This preliminary observation made, (a) for $\forall x B(x)$ follows at once, since by the universal quantification clause in the definition of truth, the truth value of $\forall x B(x)$ depends only on the domain and which of its elements satisfy $B(x)$. The case of existential quantification is the same.

If function symbols are present, we must as a preliminary establish by induction on complexity of terms that the denotation of a term depends only on the denotations of the nonlogical symbols occurring in it. This is trivial in the case of a constant. If it is true for terms $t_1, \ldots, t_n$, then it is true for the term $f(t_1, \ldots, t_n)$, since the definition of denotation of term mentions only the denotation of the nonlogical symbol f and the denotations of the terms $t_1, \ldots, t_n$. This preliminary observation made, (a) for atomic sentences follows, since by the atomic clause in the definition of truth, the truth value of an atomic sentence depends only on the denotation of its predicate and the denotations of its terms. The nonatomic cases in the proof require no change.

We have proved (b) in the course of proving (a). Having (b), the proof of (c) reduces to showing that whether a sentence $F(t)$ is true depends only on whether the element m denoted by t satisfies $F(x)$, which by the definition of satisfaction is to say, on whether $F(c)$ is true, where c is a constant having the same denotation m as t. The proof that $F(c)$ and $F(t)$ have the same truth value if c and t have the same denotation is relegated to the problems at the end of the chapter.

It is also extensionality (specifically, part (c) of Proposition 10.2) that justifies our earlier passing remarks to the effect that the substitutional approach to defining quantification does work *when every element of the domain is the denotation of some closed term.* If for some closed term t the sentence $B(t)$ is true, then letting m be the denotation of t, it follows by extensionality that m satisfies $B(x)$, and hence $\exists x\, B(x)$ is true; and conversely, if $\exists x\, B(x)$ is true, then some m satisfies $B(x)$, and *assuming that every element of the domain is the denotation of some closed term*, then some term t denotes m, and by extensionality, $B(t)$ is true. Thus under the indicated assumption, $\exists x B(x)$ is true if and only if for some term t, $B(t)$ is true, and similarly $\forall x\, B(x)$ is true if and only if for every term t, $B(t)$ is true.

Similarly, if every element of the domain is the denotation of a closed term *of some special kind* then $\exists x\, B(x)$ is true if and only if $B(t)$ is true for some closed term t that is *of that special kind.* In particular, for the standard interpretation $\mathcal{N}^*$ of the language of arithmetic L^*, where every element of the domain is the denotation of one of the terms $\mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots$, we have

$$\mathcal{N}^* \models \forall x\, F(x) \quad \text{if and only if} \quad \text{for every natural number } m, \mathcal{N}^* \models F(\mathbf{m})$$
$$\mathcal{N}^* \models \exists x\, F(x) \quad \text{if and only if} \quad \text{for some natural number } m, \mathcal{N}^* \models F(\mathbf{m})$$

where $\mathbf{m}$ is the numeral for the number m (that is, the term consisting of the cipher $\mathbf{0}$ followed by m copies of the accent $'$).

10.2 Metalogical Notions

Now that rigorous definitions of formula and sentence, and of satisfaction and truth, have been given, we can proceed to the definitions of the main notions of logical theory. A set of sentences Γ *implies* or has as a *consequence* the sentence D if there is no interpretation that makes every sentence in Γ true, but makes D false. This is the same as saying that every interpretation that makes every sentence in Γ true makes D true. (Or almost the same. Actually, if D contains a nonlogical symbol not in Γ, an interpretation might make Γ true but assign no denotation to this symbol and therefore no truth value to D. But in such a case, however the denotation is extended to assign a denotation to any such symbols and therewith a truth value to D, Γ will still be true by the extensionality lemma, so D cannot be false and must be true. To avoid fuss over such points, in future we tacitly understand ‘every interpretation’ to mean ‘every interpretation that assigns denotations to all the nonlogical symbols in whatever sentences we are considering’.) We use ‘makes every sentence in Γ true’ and ‘makes Γ true’ interchangeably, and likewise ‘the sentences in the set Γ imply D’ and ‘Γ implies D’. When Γ contains but a single sentence C (in symbols, when $\Gamma = \{C\}$), we use ‘Γ implies D’ and ‘C implies D’ interchangeably. Let us give a few examples. There are more in the problems at the end of the chapter (and many, many, many more in introductory textbooks).

10.3 Example. Some implication principles

(a) $\sim\sim B$ implies B.

(b) B implies $(B \vee C)$ and C implies $(B \vee C)$.

(c) $\sim(B \vee C)$ implies $\sim B$ and $\sim C$.

(d) $B(t)$ implies $\exists x B(x)$.

(e) $\sim\exists x B(x)$ implies $\sim B(t)$.

(f) $s = t$ and $B(s)$ imply $B(t)$.

Proofs: For (a), by the negation clause in the definition of truth, in any interpretation, if $\sim\sim B$ is true, then $\sim B$ must be false, and B must be true. For (b), by the disjunction clause in the definition of truth, in any interpretation, if B is true, then $(B \vee C)$ is true; similarly for C. For (c), by what we have just shown, any interpretation that does *not* make $(B \vee C)$ true can*not* make B true; hence any intepretation that makes $\sim(B \vee C)$ true makes $\sim B$ true; and similarly for $\sim C$. For (d), in any interpretation, by the extensionality lemma $B(t)$ is true if and only if the element m of the domain that is denoted by t satisfies $B(x)$, in which case $\exists x B(x)$ is true. As for (e), it follows from what we have just shown much as (c) follows from (b). For (f), by the identity clause in the definition of truth, in any interpretation, if $s = t$ is true, then s and t denote the same element of the domain. Then by the extensionality lemma $B(s)$ is true if and only if $B(t)$ is true.

There are two more important notions to go with implication or consequence. A sentence D is *valid* if no interpretation makes D false. In this case, *a fortiori* no interpretation makes Γ true and D false; Γ implies D for *any* Γ. Conversely, if every Γ implies D, then since for every interpretation there is a set of sentences Γ it makes true, no interpretation can make D false, and D is valid. A set of sentences Γ is *unsatisfiable* if no interpretation makes Γ true (and is *satisfiable* if some interpretation does). In this case, *a fortiori* no interpretation makes Γ true and D false, so Γ implies D for any D. Conversely, if Γ implies every D, then since for every interpretation there is a sentence it makes false, there can be no interpretation making Γ true, and Γ is unsatisfiable.

Notions such as consequence, unsatisfiability, and validity are often called ‘metalogical’ in contrast to the notions of negation, conjunction, disjunction, and universal and existential quantification, which are simply called ‘logical’. Terminology aside, the difference is that there are symbols $\sim$, &, $\vee$, $\forall$, $\exists$ in our formal language (the 'object language') for negation and the rest, whereas words like 'consequence' only appear in the unformalized prose, the mathematical English, in which we talk *about* the formal language (the 'metalanguage').

Just as for implication or consequence, so for validity and for unsatisfiability and satisfiability, there are innumerable little principles that follow directly from the definitions. For instance: if a set is satisfiable, then so is every subset (since an interpretation making every sentence in the set true will make every sentence in the subset true); no set containing both a sentence and its negation is satisfiable (since no interpretation makes them both true); and so on. The plain assertions of Example 10.3 can each be elaborated into fancier versions about validity and (un)satisfiability, as we next illustrate in the case of 10.3(a).

10.4 Example. Variations on a theme

(a) $\sim\sim B$ implies B.

(b) If Γ implies $\sim\sim B$, then Γ implies B.

(c) If B implies D, then $\sim\sim B$ implies D.

(d) If $\Gamma \cup \{B\}$ implies D, then $\Gamma \cup \{\sim\sim B\}$ implies D.

(e) If $\sim\sim B$ is valid, then B is valid.

(f) If $\Gamma \cup \{B\}$ is unsatisfiable, then $\Gamma \cup \{\sim\sim B\}$ is unsatisfiable.

(g) If $\Gamma \cup \{\sim\sim B\}$ is satisfiable, then $\Gamma \cup \{B\}$ is satisfiable.

Proof: (a) is a restatement of 10.3(a). For (b), we are given that every interpretation that makes Γ true makes $\sim\sim B$ true, and want to show that any interpretation that makes Γ true makes B true. But this is immediate from (a), which says that any interpretation that makes $\sim\sim B$ true makes B true. For (c), we are given that any interpretation that makes B true makes D true, and want to show that any interpretation that makes $\sim\sim B$ true makes D true. But again, this is immediate from the fact that any interpretation that makes $\sim\sim B$ true makes B true. In (d), $\Gamma \cup \{B\}$ denotes the result of adding B to Γ. The proof in this case is a combination of the proofs of (b) and (c). For (e), we are given that every interpretation makes $\sim\sim B$ true, and want to show that every interpretation makes B true, while for (f), we are given that no interpretation makes Γ and B true, and want to show that no interpretation makes Γ and $\sim\sim B$ true. But again both are immediate from (a), that is, from the fact that every interpretation that makes $\sim\sim B$ true makes B true. Finally, (g) is immediate from (f).

We could play the same game with any of 10.3(b)–10.3(f). Some results exist only in the fancy versions, so to speak.

10.5 Example. Some satisfiability principles

(a) If $\Gamma \cup \{(A \vee B)\}$ is satisfiable, then either $\Gamma \cup \{A\}$ is satisfiable, or $\Gamma \cup \{B\}$ is satisfiable.

(b) If $\Gamma \cup \{\exists x B(x)\}$ is satisfiable, then for any constant c not occurring in Γ or $\exists x B(x)$, $\Gamma \cup \{B(c)\}$ is satisfiable.

(c) If Γ is satisfiable, then $\Gamma \cup \{t = t\}$ is satisfiable.

Proof: For (a), we are given that some interpretation makes Γ and $A \vee B$ true, and want to show that some interpretation makes Γ and A true, or some makes Γ and B true. In fact, the *same* interpretation that makes Γ and $A \vee B$ true either makes A true or makes B true, by the disjunction clause in the definition of truth. For (b), we are given that some interpretation makes Γ and $\exists x B(x)$ true, and want to show that some interpretation makes Γ and $B(c)$ true, assuming c does not occur in Γ or $\exists x B(x)$. Well, since $\exists x B(x)$ is true, some element m of the domain satisfies $B(x)$. And since c does not occur in Γ or $\exists x B(x)$, we can change the interpretation to make m the denotation of c, without changing the denotations of any nonlogical symbols in Γ or $\exists x B(x)$, and so by extensionality not changing their truth values. But then Γ is still true, and since m satisfies $B(x)$, $B(c)$ is also true. For (c), we are given that some interpretation makes Γ true and want to show that some interpretation makes Γ and $t = t$ true. But *any* interpretation makes $t = t$ true, so long as it assigns a denotation to each nonlogical symbol in t, and if our given interpretation does not, it at least assigns a denotation to every nonlogical symbol in t that occurs in Γ, and if we extend it to assign denotations to any other nonlogical symbols in t, by extensionality Γ will still be true, and now $t = t$ will be true also.

There is one more important metalogical notion: two sentences are *equivalent over an interpretation* $\mathcal{M}$ if they have the same truth value. Two formulas $F(x)$ and $G(x)$ are equivalent over $\mathcal{M}$ if, taking a constant c occurring in neither, the sentences $F(c)$ and $G(c)$ are equivalent over every interpretation $\mathcal{M}^c_m$obtained by extending $\mathcal{M}$ to provide some denotation m for c. Two sentences are (*logically*) *equivalent* if they are equivalent over all interpretations. Two formulas $F(x)$ and $G(x)$ are (*logically*) equivalent if, taking a constant c occurring in neither, the sentences $F(c)$ and $G(c)$ are (logically) equivalent. A little thought shows that formulas are (logically) equivalent if they are equivalent over every interpretation. The definitions may be extended to formulas with more than one free variable. We leave the development of the basic properties of equivalence entirely to the problems.

Before closing this chapter and bringing on those problems, a remark will be in order. The method of induction on complexity we have used in this chapter and the preceding to prove such unexciting results as the parenthesis and extensionality lemmas will eventually be used to prove some less obvious and more interesting results. Much of the interest of such results about formal languages depends on their being applicable to ordinary language. We have been concerned here mainly with how to read sentences of our formal language in ordinary language, and much less with writing sentences of ordinary language in our formal language, so we need to say a word about the latter topic.

In later chapters of this book there will be many examples of writing assertions from *number theory*, the branch of mathematics concerned with the natural numbers, as first-order sentences in the language of arithmetic. But the full scope of what can be done with first-order languages will not be apparent from these examples, or this book, alone. Works on set theory give examples of writing assertions from other branches of mathematics as first-order sentences in a *language of set theory*, and make it plausible that in virtually *all* branches of mathematics, what we want to say can be said in a first-order language. Works on logic at the introductory level contain a wealth of examples of how to say what we want to say in a first-order language from outside mathematics (as in our genealogical examples).

But this cannot *always* be done outside of mathematics, and some of our results *do not* apply unrestrictedly to ordinary language. A case in point is unique readability. In ordinary language, ambiguous sentences of the type 'A and B or C' are perfectly possible. Of course, though *possible*, they are not *desirable*: the sentence ought to be rewritten to indicate whether 'A, and either B or C' or 'Either A and B, or C' is meant. A more serious case in point is extensionality. In ordinary language it is *not* always the case that one expression can be changed to another denoting the same thing without altering truth values. To give the classic example, Sir Walter Scott was the author of the historical novel *Waverley*, but there was a time when this fact was unknown, since the work was originally published anonymously. At that time, 'It is known that Scott is Scott' was as always true, but 'It is known that the author of *Waverley* is Scott' was false, even though 'Scott' and 'the author of *Waverly*' had the same denotation.

To put the matter another way, writing s for 'Scott' and t for 'the author of *Waverley*', and writing $A(x)$ for 'x is Scott' and $\Box$ for 'it is known that', what we have just said is that $s = t$ and $\Box A(s)$ may be true without $\Box A(t)$ being true, in contrast to one of our examples above, according to which, in our formal languages, $s = t$ and $B(s)$ always imply $B(t)$. There is no contradiction with our example, of course, since our formal languages do not contain any operator like $\Box$; but for precisely this reason, *not* everything that can be expressed in ordinary language can be expressed in our formal languages. There is a separate branch of logic, called *modal logic*, devoted to operators like $\Box$, and we are eventually going to get a peek at a corner of this branch of logic, though only in the last chapter of the book.

Problems

10.1 Complete the proof of the extensionality lemma (Proposition 10.2) by showing that if c is a constant and t a closed term having the same denotation, then substituting t for c in a sentence does not change the truth value of the sentence.

10.2 Show that $\exists y \forall x R(x, y)$ implies $\forall x \exists y R(x, y)$.

10.3 Show that $\forall x \exists y F(x, y)$ does not imply $\exists y \forall x F(x, y)$.

10.4 Show that:

(a) If the sentence E is implied by the set of sentences Δ and every sentence D in Δ is implied by the set of sentences Γ, then E is implied by Γ.

(b) If the sentence E is implied by the set of sentences $\Gamma \cup \Delta$ and every sentence D in Δ is implied by the set of sentences Γ, then E is implied by Γ.

10.5 Let $\varnothing$ be the empty set of sentences, and let $\perp$ be any sentence that is not true on any interpretation. Show that:

(a) A sentence D is valid if and only if D is a consequence of $\varnothing$.

(b) A set of sentences Γ is unsatisfiable if and only if $\perp$ is a consequence of Γ.

10.6 Show that:

(a) $\{C_1, \ldots, C_m\}$ is unsatisfiable if and only if $\sim C_1 \vee \cdots \vee \sim C_m$ is valid.

(b) D is a consequence of $\{C_1, \ldots, C_m\}$ if and only if $\sim C_1 \vee \cdots \vee \sim C_m \vee D$ is valid.

(c) D is a consequence of $\{C_1, \ldots, C_m\}$ if and only if $\{C_1, \ldots, C_m, \sim D\}$ is unsatisfiable.

(d) D is valid if and only if $\sim D$ is unsatisfiable.

10.7 Show that $B(t)$ and $\exists x(x = t \,\&\, B(x))$ are logically equivalent.

10.8 Show that:

(a) $(B \,\&\, C)$ implies B and implies C.

(b) $\sim B$ implies $\sim(B \,\&\, C)$, and $\sim C$ implies $\sim(B \,\&\, C)$.

(c) $\forall x B(x)$ implies $B(t)$.

(d) $\sim B(t)$ implies $\sim \forall x B(x)$.

10.9 Show that:

(a) If $\Gamma \cup \{\sim(B \,\&\, C)\}$ is satisfiable, then either $\Gamma \cup \{\sim B\}$ is satisfiable or $\Gamma \cup \{\sim C\}$ is satisfiable.

(b) If $\Gamma \cup \{\sim \forall x B(x)\}$ is satisfiable, then for any constant c not occurring in Γ or $\forall x B(x)$, $\Gamma \cup \{\sim B(c)\}$ is satisfiable.

10.10 Show that the following hold for equivalence over any interpretation (and hence for logical equivalence), for any sentences (and hence for any formulas):

(a) F is equivalent to F.

(b) If F is equivalent to G, then G is equivalent to F.

(c) If F is equivalent to G and G is equivalent to H, then F is equivalent to H.

(d) If F and G are equivalent, then $\sim F$ and $\sim G$ are equivalent.

(e) If F_1 and G_1 are equivalent, and F_2 and G_2 are equivalent, then $F_1 \,\&\, F_2$ and $G_1 \,\&\, G_2$ are equivalent, and similarly for $\vee$.

(f) If c does not occur in $F(x)$ or $G(x)$, and $F(c)$ and $G(c)$ are equivalent, then $\forall x F(x)$ and $\forall x G(x)$ are equivalent, and similarly for $\exists$.

10.11 (*Substitution of equivalents.*) Show that the following hold for equivalence over any interpretation (and hence for logical equivalence):

(a) If sentence G results from sentence F on replacing each occurrence of an atomic sentence A by an equivalent sentence B, then F and G are equivalent.

(b) Show that the same holds for an atomic formula A and an equivalent formula B (provided, to avoid complications, that no variable occurring in A occurs bound in B or F).

(c) Show that the same holds even when A is not atomic.

10.12 Show that $F(x)$ is (logically) equivalent to $G(x)$ if and only if $\forall x(F(x) \leftrightarrow G(x))$ is valid.

10.13 (*Relettering bound variables.*) Show that:

(a) If F is a formula and y a variable not occurring free in F, then F is (logically) equivalent to a formula in which y does not occur at all. The same applies to any number of variables $y_1, \ldots, y_n$.

(b) Every formula is logically equivalent to a formula having no subformulas in which the same variable occurs both free and bound.

10.14 Show that the following pairs are equivalent:

(a) $\forall x F(x) \,\&\, \forall y G(y)$ and $\forall u(F(u) \,\&\, G(u))$.

(b) $\forall x F(x) \vee \forall y G(y)$ and $\forall u \forall v(F(u) \vee G(v))$.

(c) $\exists x F(x) \,\&\, \exists y G(y)$ and $\exists u \exists v(F(u) \,\&\, G(v))$.

(d) $\exists x F(x) \vee \exists y G(y)$ and $\exists u(F(u) \vee G(u))$.

[In (a), it is to be understood that u may be a variable not occurring free in $\forall x F(x)$ or $\forall y G(y)$; in particular, if x and y are the same variable, u may be that same variable. In (b) it is to be understood that u and v may be any distinct variables not occurring free in $\forall x F(x) \vee \forall y G(y)$; in particular, if x does not occur in free in $\forall y G(y)$ and y does not occur free in $\forall x F(x)$, then u may be x and y may be v. Analogously for (d) and (c).]

11

The Undecidability of First-Order Logic

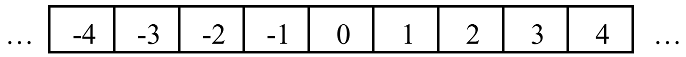
This chapter connects our work on computability with questions of logic. Section 11.1 presupposes familiarity with the notions of logic from Chapter 9 and 10 and of Turing computability from Chapters 3–4, including the fact that the halting problem is not solvable by any Turing machine, and describes an effective procedure for producing, given any Turing machine M and input n, a set of sentences Γ *and a sentence D such that M given input n will eventually halt if and only if* Γ *implies D. It follows that if there were an effective procedure for deciding when a finite set of sentences implies another sentence, then the halting problem would be solvable; whereas, by Turing's thesis, the latter problem is* not *solvable, since it is not solvable by a Turing machine. The upshot is, one gets an argument, based on Turing's thesis for (the* Turing–Büchi proof *of)* Church's theorem, *that the decision problem for implication is not effectively solvable. Section 11.2 presents a similar argument—the* Gödel-style proof *of Church's theorem—this time using not Turing machines and Turing's thesis, but primitive recursive and recursive functions and Church's thesis, as in Chapters 6–7. The constructions of the two sections, which are independent of each other, are both instructive; but an entirely different proof, not dependent on Turing's or Church's thesis, will be given in a later chapter, and in that sense the present chapter is optional. (After the present chapter we return to pure logic for the space of several chapters, to resume to the application of computability theory to logic with Chapter 15.)*

11.1 Logic and Turing Machines

We are going to show how, given the machine table or flow chart or other suitable presentation of a Turing machine, and any n, we can effectively write down a *finite* set of sentences Γ and a sentence D such that Γ implies D if and only if the machine in question does eventually halt when started with input n, that is, when started in its initial state scanning the leftmost of a block of n strokes on an otherwise blank tape. It follows that if the *decision problem* for logical implication could be solved, that is, if an effective method could be devised that, applied to any finite set of sentences Γ and sentence D, would in a finite amount of time tell us whether or not Γ implies D, then the *halting problem* for Turing machines could be solved, or in other words, an effective method would exist that, applied to any suitably presented Turing machine and number n, would in a finite amount of time tell us whether or not that machine halts when started with input n. Since we have seen in Chapter 4 that, assuming Turing's thesis, the halting problem is *not* solvable, it follows that, again assuming Turing's thesis, the decision problem is unsolvable, or, as is said, that logic is undecidable.

In principle this section requires only the material of Chapters 3–4 and 9–10. In practice some facility at recognizing simple logical implications will be required: we are going to appeal freely to various facts about one sentence implying another, leaving the verification of these facts largely to the reader.

We begin by introducing simultaneously the language in which the sentences in Γ and the sentence D will be written, and its *standard interpretation* $\mathcal{M}$. The language interpretation will depend on what machine and what input n we are considering. The domain of $\mathcal{M}$ will in all cases be the integers, positive and zero and negative. The nonnegative integers will be used to number the *times* when the machine is operating: the machine starts at time 0. The integers will also be used to number the squares on the tape: the machine starts at square 0, and the squares to the left and right are numbered as in Figure 11-1.



…	-4	-3	-2	-1	0	1	2	3	4	…

Figure 11-1. Numbering the squares of a Turing tape.

There will be a constant **0**, whose denotation in the standard interpretation will be zero, and two-place predicates **S** and **<**, whose denotations will be the successor relation (the relation an integer n bears to $n + 1$ and nothing else) and the usual order relation, respectively. To save space, we write **S**uv rather than **S**(u, v), and similarly for other predicates. As to such other predicates, there will further be, for each of the (nonhalted) states of the machine, numbered let us say from 1 (the initial state) to k, a one-place predicate. In the standard interpretation, $\mathbf{Q}_i$ will denote the set of $t \geq 0$ such that at the time numbered t the machine is in the state numbered i. Besides this we need two more two-place predicates **@** and **M**. The denotation of the former will be the set of pairs of integers $t \geq 0$ and x such that at the time number t, the machine is at the square numbered x. The denotation of the latter will be the set of $t \geq 0$ and x such that at time t, square x is 'marked', that is, contains a stroke rather than a blank. (We use t as the variable when a time is intended, and x and y when squares are intended, as a reminder of the standard interpretation. Formally, the function of a variable is signalled by its position in the first or the second place of the predicate @ or **M**.) It would be easy to adapt our construction to the case where more symbols than just the stroke and the blank are allowed, but for present purposes there is no reason to do so.

We must next describe the sentences that are to go into Γ and the sentence D. The sentences in Γ will fall into three groups. The first contains some 'background information' about **S** and **<** that would be the same for any machine and any input. The second consists of a single sentence specific to the input n we are considering. The third consists of one sentence for each 'normal' instruction of the specific machine we are considering, that is, for each instruction except for those telling us to halt.

The ‘background information’ is provided by the following:

(1) $\forall u \forall v \forall w(((\mathbf{S}uv \,\&\, \mathbf{S}uw) \to v = w) \,\&\, ((\mathbf{S}vu \,\&\, \mathbf{S}wu) \to v = w))$

(2) $\forall u \forall v(\mathbf{S}uv \to u < v) \,\&\, \forall u \forall v \forall w((u < v \,\&\, v < w) \to u < w)$

(3) $\forall u \forall v(u < v \to u \neq v)$.

These say that a number has only one successor and only one predecessor, that a number is less than its predecessor, and so on, and are all equally true in the standard interpretation.

It will be convenient to introduce abbreviations for the *m*th-successor relation, writing

$\mathbf{S}_0 uv$ for $u = v$

$\mathbf{S}_1 uv$ for $\mathbf{S}uv$

$\mathbf{S}_2 uv$ for $\exists y(\mathbf{S}uy \,\&\, \mathbf{S}yv)$

$\mathbf{S}_3 uv$ for $\exists y_1 \exists y_2(\mathbf{S}uy_1 \,\&\, \mathbf{S}y_1 y_2 \,\&\, \mathbf{S}y_2 v)$

and so on. (In $\mathbf{S}_2$, y may be any convenient variable distinct from u and v; for definiteness let us say the first on our official list of variables. Similarly for $\mathbf{S}_3$.) The following are then true in the standard interpretation.

(4) $\forall u \forall v \forall w(((\mathbf{S}_m uv \,\&\, \mathbf{S}_m uw) \to v = w) \,\&\, ((\mathbf{S}_m vu \,\&\, \mathbf{S}_m wu) \to v = w))$

(5) $\forall u \forall v(\mathbf{S}_m uv \to u < v)$ if $m \neq 0$

(6) $\forall u \forall v(\mathbf{S}_m uv \to u \neq v)$ if $m \neq 0$

(7) $\forall u \forall v \forall w((\mathbf{S}_m wu \,\&\, \mathbf{S}uv) \to \mathbf{S}_k wv)$ if $k = m + 1$

(8) $\forall u \forall v \forall w((\mathbf{S}_k wv \,\&\, \mathbf{S}uv) \to \mathbf{S}_m wu)$ if $m = k - 1$.

Indeed, these are logical consequences of (1)–(3) and hence of Γ, true in *any* interpretation where Γ is true: (4) follows on repeated application of (1); (5) follows on repeated application of (2); (6) follows from (3) and (5); (7) is immediate from the definitions; and (8) follows from (7) and (1). If we also write $\mathbf{S}_{-m}uv$ for $\mathbf{S}_m vu$, (4)–(8) still hold.

We need some further notational conventions before writing down the remaining sentences of Γ. Though officially our language contains only the numeral $\mathbf{0}$ and not numerals $\mathbf{1}$, $\mathbf{2}$, $\mathbf{3}$, or $-\mathbf{1}$, $-\mathbf{2}$, $-\mathbf{3}$, it will be suggestive to write $y = \mathbf{1}$, $y = \mathbf{2}$, $y = -\mathbf{1}$, and the like for $\mathbf{S}_1(\mathbf{0}, y)$, $\mathbf{S}_2(\mathbf{0}, y)$, $\mathbf{S}_{-1}(\mathbf{0}, y)$, and so on, and to understand the application of a predicate to a numeral in the natural way, so that, for instance, $\mathbf{Q}_i\mathbf{2}$ and $\mathbf{S2}u$ abbreviate $\exists y(y = \mathbf{2} \,\&\, \mathrm{Q}_i y)$ and $\exists y(y = \mathbf{2} \,\&\, \mathbf{S}yu)$. A little thought shows that with these conventions (6)–(8) above (applied with $\mathbf{0}$ for w) give us the following wherein $\mathbf{p}$, $\mathbf{q}$, and so on, are the numerals for the numbers p, q, and so on:

(9) $\mathbf{p} \neq \mathbf{q}$ if $p \neq q$

(10) $\forall v(\mathbf{Sm}v \to v = \mathbf{k})$ where $k = m + 1$

(11) $\forall u(\mathbf{S}u\mathbf{k} \to u = \mathbf{m})$ where $m = k - 1$.

These abbreviatory conventions enable us to write down the remaining sentences of Γ comparatively compactly.

The one member of Γ pertaining to the input n is a description of (the configuration at) time 0, as follows:

(12) $$\mathbf{Q}_0\mathbf{0}\ \&\ @\mathbf{00}\ \&\ \mathbf{M00}\ \&\ \mathbf{M01}\ \&\ \ldots\ \&\ \mathbf{M0n}\ \&$$
$$\forall x((x \neq \mathbf{0}\ \&\ x \neq \mathbf{1}\ \&\ \ldots\ \&\ x \neq \mathbf{n-1}) \rightarrow \sim\mathbf{M0}x).$$

This is true in the standard interpretation, since at time 0 the machine is in state 1, at square 0, with squares 0 through n marked to represent the input n, and all other squares blank.

To complete the specification of Γ, there will be one sentence for each nonhalting instruction, that is, for each instruction of the following form, wherein j is not the halted state:

(*) If you are in state i and are scanning symbol e,
then —— and go into state j.

In writing down the corresponding sentence of Γ, we use one further notational convention, sometimes writing $\mathbf{M}$ as $\mathbf{M}_1$ and $\sim\mathbf{M}$ as $\mathbf{M}_0$. Thus $\mathbf{M}_e tx$ says, in the standard interpretation, that at time t, square x contains symbol e (where $e = 0$ means the blank, and $e = 1$ means the stroke). Then the sentence corresponding to (*) will have the form

(13) $$\forall t\forall x((\mathbf{Q}_i t\ \&\ @tx\ \&\ \mathbf{M}_e tx) \rightarrow$$
$$\exists u(\mathbf{S}tu\ \&\ \text{——}\ \&\ \mathrm{Q}_j u\ \&$$
$$\forall y((y \neq x\ \&\ \mathbf{M}_1 ty) \rightarrow \mathbf{M}_1 uy)\ \&\ \forall y((y \neq x\ \&\ \mathbf{M}_0 ty) \rightarrow \mathbf{M}_0 uy))).$$

The last two clauses just say that the marking of squares other than x remains unchanged from one time t to the next time u.

What goes into the space '——' in (13) depends on what goes into the corresponding space in (*). If the instruction is to (remain at the same square x but) print the symbol d, the missing conjunct in (9) will be

$$@ux\ \&\ \mathbf{M}_d ux.$$

If the instruction is to move one square to the right or left (leaving the marking of the square x as it was), it will instead be

$$\exists y(\mathbf{S}_{\pm 1}xy\ \&\ @uy\ \&\ (\mathbf{M}ux \leftrightarrow \mathbf{M}tx))$$

(with the minus sign for left and the plus sign for right). A little thought shows that when filled in after this fashion, (13) exactly corresponds to the instruction (*), and will be true in the standard interpretation.

This completes the specification of the set Γ. The next task is to describe the sentence D. To obtain D, consider a halting instruction, that is, an instruction of the type

(†) If you are in state i and are scanning symbol e, then halt.

For each such instruction write down the sentence

$$\exists t \exists x(\mathbf{Q}_i t \ \& \ @tx \ \& \ \mathbf{M}_e tx). \tag{14}$$

This will be true in the standard interpretation if and only if in the course of its operations the machine eventually comes to a configuration where the applicable instruction is (†), and halts for this reason. We let D be the disjunction of all sentences of form (14) for all halting instructions (†). Since the machine will eventually halt if and only if it eventually comes to a configuration where the applicable instruction is some halting instruction or other, the machine will eventually halt if and only if D is true in the standard interpretation.

We want to show that Γ implies D if and only if the given machine, started with the given input, eventually halts. The 'only if' part is easy. All sentences in Γ are true in the standard interpretation, whereas D is true only if the given machine started with the given input eventually halts. If the machine does not halt, we have an interpretation where all sentences in Γ are true and D isn't, so Γ does not imply D.

For the 'if' part we need one more notion. If $a \geq 0$ is a time at which the machine has not (yet) halted, we mean by the *description of time a* the sentence that does for a what (12) does for 0, telling us what state the machine is in, where it is, and which squares are marked at time a. In other words, if at time a the machine is in state i, at square p, and the marked squares are $q_1, q_2, \ldots, q_m$, then the description of time a is the following sentence:

$$\begin{aligned} &\mathbf{Q}_i\mathbf{a} \ \& \ @\mathbf{ap} \ \& \ \mathbf{Maq}_1 \ \& \ \mathbf{Maq}_2 \ \& \ldots \& \ \mathbf{Maq}_m \ \& \\ &\qquad \forall x((x \neq \mathbf{q}_1 \ \& \ x \neq \mathbf{q}_2 \ \& \ldots \& \ x \neq \mathbf{q}_m) \rightarrow {\sim}\mathbf{Ma}x). \end{aligned} \tag{15}$$

It is important to note that (15) provides, directly or indirectly, the information whether the machine is scanning a blank or a stroke at time a. If the machine is scanning a stroke, then p is one of the q_r for $1 \leq r \leq m$, and $\mathbf{M}_1\mathbf{ap}$, which is to say $\mathbf{Map}$, is actually a conjunct of (15). If the machine is scanning a blank, then p is different from each of the various numbers q. In this case $\mathbf{M}_0\mathbf{ap}$, which is to say ${\sim}\mathbf{Map}$, is implied by (15) and Γ. Briefly put, the reason is that (9) gives $\mathbf{p} \neq \mathbf{q}_r$ for each q_r, and then the last conjuct of (15) gives ${\sim}\mathbf{Map}$.

[Less briefly but more accurately put, what the last conjunct of (15) *abbreviates* amounts to

$$\forall x(({\sim}\mathbf{S}_{q_1}\mathbf{0}x \ \& \ldots {\sim}\mathbf{S}_{q_m}\mathbf{0}x) \rightarrow {\sim}\exists t(\mathbf{S}_a\mathbf{0}t \ \& \ \mathbf{M}tx)).$$

What (9) applied to p and q_r *abbreviates* is

$${\sim}\exists x((\mathbf{S}_p x \ \& \ \mathbf{S}_{q_r} x).$$

These together imply

$${\sim}\exists t \exists x(\mathbf{S0}t \ \& \ \mathbf{S0}x \ \& \ \mathbf{M}tx)$$

which amounts to what ${\sim}\mathbf{Map}$ *abbreviates*.]

If the machine halts at time $b = a + 1$, that means that at time a we had configuration for which the applicable instruction as to what to do next was a halting instruction of form (†). In that case, $\mathbf{Q}_i\mathbf{a}$ and $@\mathbf{ap}$ will be conjuncts of the description of time

a, and $\mathbf{M}_e\mathbf{ap}$ will be either a conjunct of the description also (if $e = 1$) or a logical implication of the description and Γ (if $e = 0$). Hence (14) and therefore D will be a logical implication of Γ together with the description of time a. What if the machine does *not* halt at time $b = a + 1$?

11.1 Lemma. If $a \geq 0$, and $b = a + 1$ is a time at which the machine has not (yet) halted, then Γ together with the description of time a implies the description of time b.

Proof: The proof is slightly different for each of the four types of instructions (print a blank, print a stroke, move left, move right). We do the case of printing a stroke, and leave the other cases to the reader. Actually, this case subdivides into the unusual case where there is already a stroke on the scanned square, so that the instruction is just to change state, and the more usual case where the scanned square is blank. We consider only the latter subcase.

So the description of time a looks like this:

$$\begin{aligned}(16)\qquad & \mathbf{Q}_i\mathbf{a}\ \&\ @\mathbf{ap}\ \&\ \mathbf{Maq}_1\ \&\ \mathbf{Maq}_2\ \&\ \ldots\ \&\ \mathbf{Maq}_m\ \&\\ & \forall x((x \neq \mathbf{q}_1\ \&\ x \neq \mathbf{q}_2\ \&\ \ldots\ \&\ x \neq \mathbf{q}_m) \to {\sim}\mathbf{Ma}x)\end{aligned}$$

where $p \neq q_r$ for any r, so Γ implies $\mathbf{p} \neq \mathbf{q}_r$ by (9), and, by the argument given earlier, Γ and (16) together imply ${\sim}\mathbf{Map}$. The sentence in Γ corresponding to the applicable instruction looks like this:

$$\begin{aligned}(17)\qquad & \forall t\forall x((\mathbf{Q}_i t\ \&\ @tx\ \&\ {\sim}\mathbf{M}tx) \to\\ & \quad \exists u(\mathbf{S}tu\ \&\ @ux\ \&\ \mathbf{M}ux\ \&\ \mathbf{Q}_j u\ \&\ \forall y((y \neq x\ \&\ \mathbf{M}ty) \to \mathbf{M}uy)\\ & \qquad \&\ \forall y((y \neq x\ \&\ {\sim}\mathbf{M}ty) \to {\sim}\mathbf{M}uy))).\end{aligned}$$

The description of time b looks like this:

$$\begin{aligned}(18)\qquad & \mathbf{Q}_j\mathbf{b}\ \&\ @\mathbf{bp}\ \&\ \mathbf{Mbp}\ \&\ \mathbf{Mbq}_1\ \&\ \mathbf{Mbq}_2\ \&\ \ldots\ \&\ \mathbf{Mbq}_m\ \&\\ & \forall x((x \neq \mathbf{p}\ \&\ x \neq \mathbf{q}_1\ \&\ x \neq \mathbf{q}_2\ \&\ \ldots\ \&\ x \neq \mathbf{q}_m) \to {\sim}\mathbf{Mb}x).\end{aligned}$$

And, we submit, (18) is a consequence of (16), (17), and Γ.

[Briefly put, the reason is this. Putting $\mathbf{a}$ for t and $\mathbf{p}$ for x in (17), we get

$$\begin{aligned}& (\mathbf{Q}_i\mathbf{a}\ \&\ @\mathbf{ap}\ \&\ {\sim}\mathbf{Map}) \to\\ & \quad \exists u(\mathbf{Sa}u\ \&\ @u\mathbf{p}\ \&\ \mathbf{M}u\mathbf{p}\ \&\ \mathbf{Q}_j u\ \&\\ & \forall y((y \neq \mathbf{p}\ \&\ \mathbf{Ma}y) \to \mathbf{M}uy)\ \&\ \forall y((y \neq \mathbf{p}\ \&\ {\sim}\mathbf{Ma}y) \to {\sim}\mathbf{M}uy)).\end{aligned}$$

Since (16) and Γ imply $\mathbf{Q}_i\mathbf{a}\ \&\ @\mathbf{ap}\ \&\ {\sim}\mathbf{Map}$, we get

$$\begin{aligned}& \exists u(\mathbf{Sau}\ \&\ @u\mathbf{p}\ \&\ \mathbf{M}u\mathbf{p}\ \&\ \mathbf{Q}_j u\ \&\\ & \quad \forall y((y \neq \mathbf{p}\ \&\ \mathbf{Ma}y) \to \mathbf{M}uy)\ \&\ \forall y((y \neq \mathbf{p}\ \&\ {\sim}\mathbf{Ma}y) \to {\sim}\mathbf{M}uy)).\end{aligned}$$

By (10), $\mathbf{Sa}u$ gives $u = \mathbf{b}$, where $b = a + 1$, and we get

$$\begin{aligned}& @\mathbf{bp}\ \&\ \mathbf{Mbp}\ \&\ \mathbf{Q}_j\mathbf{b}\ \&\\ & \quad \forall y((y \neq \mathbf{p}\ \&\ \mathbf{Ma}y) \to \mathbf{Mb}y)\ \&\ \forall y((y \neq \mathbf{p}\ \&\ {\sim}\mathbf{M}ay) \to {\sim}\mathbf{Mb}y).\end{aligned}$$

The first three conjuncts of this last are the same, except for order, as the first three conjuncts of (18). The fourth conjunct, together with $\mathbf{p} \neq \mathbf{q}_k$ from (9) and the conjunct

Maq$_k$ of (16), gives the conjunct **Mbq**$_k$ of (18). Finally, the fifth conjunct together with the last conjunct of (16) gives the last conjunct of (18). The reader will see now what we meant when we said at the outset, 'Some facility at recognizing simple logical implications will be required.']

Now the description of time 0 is one of the sentences in Γ. By the foregoing lemma, if the machine does not stop at time 1, the description of time 1 will be a consequence of Γ, and if the machine then does not stop at time 2, the description of time 2 will be a consequence of Γ together with the description of time 1 (or, as we can more simply say, since the description of time 1 is a consequence of Γ, the description of time 2 will be a consequence of Γ), and so on until the last time a before the machine halts, if it ever does. If it does halt at time $a + 1$, we have seen that the description of time a, which we now know to be a consequence of Γ, implies D. Hence if the machine ever halts, Γ implies D.

Hence we have established that if the decision problem for logical implication were solvable, the halting problem would be solvable, which (assuming Turing's thesis) we know it is not. Hence we have established the following result, assuming Turing's thesis.

11.2 Theorem (Church's theorem). The decision problem for logical implication is unsolvable.

11.2 Logic and Primitive Recursive Functions

By the *nullity problem* for a two-place primitive recursive function f we mean the problem of devising an effective procedure that, given any m, would in a finite amount of time tell us whether or not there is an n such that $f(m, n) = 0$. We are going to show how, given f, to write down a certain finite set of sentences Γ and a certain formula $D(x)$ in a language that contains the constants **0** and the successor symbol $'$ from the language of arithmetic, and therefore contains the numerals $\mathbf{0}'$, $\mathbf{0}''$, $\mathbf{0}'''$, ... or **1**, **2**, **3**, ... as we usually write them. And then we are going to show that for any m, Γ implies $D(\mathbf{m})$ if and only if there is an n such that $f(m, n) = 0$. It follows that if the decision problem for logical implication could be solved, and an effective method devised to tell whether or not a given finite set of sentences Γ implies a sentence D, then the nullity problem for any f could be solved. Since it is known that, assuming Church's thesis, there is an f for which the nullity problem is *not* solvable, it follows, again assuming Church's thesis, that the decision problem for logical implication is unsolvable, or, as is said, that logic is undecidable. The proof of the fact just cited about the unsolvability of the nullity problem requires the apparatus of Chapter 8, but for the reader who is willing to take this fact on faith, this section otherwise presupposes only the material of Chapters 6–7 and 9–10.

To begin the construction, the function f, being primitive recursive, is built up from the basic functions (successor, zero, the identity functions) by the two processes of composition and primitive recursion. We can therefore make a finite list of primitive recursive functions $f_0, f_1, f_2, \ldots, f_r$, such that for each i from 1 to r, f_i is either the zero function or the successor function or one of the identity functions, or is obtained from earlier functions in the list by composition or primitive recursion, with the last function f_r being the function f. We introduce a language with the symbol $\mathbf{0}$, the successor symbol $'$, and a function symbol $\mathbf{f}_i$ of the appropriate number of places for each of the functions f_i. In the *standard interpretation* of the language, the domain will be the natural numbers, $\mathbf{0}$ will denote zero, $'$ will denote the successor function, and each $\mathbf{f}_i$ will denote f_i, so that in particular $\mathbf{f}_r$ will denote f.

The set of sentences Γ will consist of one or two sentence for each f_i for $i > 0$. In case f_i is the zero function, the sentence will be

$$\forall x\ \mathbf{f}_i(x) = \mathbf{0}. \tag{1}$$

In case f_i is the successor function, the sentence will be

$$\forall x\ \mathbf{f}_i(x) = x'. \tag{2}$$

(In this case $\mathbf{f}_i$ will be *another* symbol besides $'$ for the successor function; but it does not matter if we happen to have two symbols for the same function.) In case f_i is the identity function id^n_k, the sentence will be

$$\forall x_1 \cdots \forall x_n\ \mathbf{f}_i(x_1, \ldots, x_n)\ = x_k. \tag{3}$$

If case f_i is obtained from f_k and $f_{j_1}, \ldots f_{j_p}$, where $j_1, \ldots, j_p$ and k are all $< i$, by composition, the sentence will be

$$\forall x\ \mathbf{f}_i(x) = \mathbf{f}_k(\mathbf{f}_{j_1}(x), \ldots \mathbf{f}_{j_p}(x)). \tag{4}$$

In case f_i is obtained from f_j and f_k, where j and k are $< i$, by primitive recursion, there will be two sentences, as follows.

$$\forall x\ \mathbf{f}_i(x, \mathbf{0}) = \mathbf{f}_j(x). \tag{5a}$$

$$\forall x \forall y\ \mathbf{f}_i(x, y') = \mathbf{f}_k(x, y, \mathbf{f}_i(x, y)). \tag{5b}$$

[In (4) and (5) we have written x and $\forall x$ for $x_1, \ldots, x_n$ and $\forall x_1 \cdots \forall x_n$.] Clearly all these sentences are true in the intended interpretation. The formula $D(x)$ will simply be $\exists y\, f_r(x, y) = \mathbf{0}$. For given m, the sentence $D(\mathbf{m})$ will be true in the standard interpretation if and only if there is an n with $f(m, n) = 0$.

We want to show that for any m, $D(\mathbf{m})$ will be implied by Γ if and only if there is an n with $f(m, n) = 0$. The 'only if' part is easy. All sentences in Γ are true in the standard interpretation, whereas $D(\mathbf{m})$ is true only if there is an n with $f(m, n) = 0$. If there is no such n, we have an interpretation where also sentences in Γ are true and $D(\mathbf{m})$ isn't, so Γ does not imply $D(\mathbf{m})$.

For the 'if' part we need one more notion. Call Γ *adequate* for the function f_i if whenever $f_i(a) = b$, then $\mathbf{f}_i(\mathbf{a}) = \mathbf{b}$ is implied by Γ. (We have written a for $a_1, \ldots, a_n$ and $\mathbf{a}$ for $\mathbf{a}_1, \ldots, \mathbf{a}_{n}$.) The presence of (1)–(3) in Γ guarantees that it is adequate for any f_i that is a basic function (zero, successor, or an identity function). What about more complicated functions?

11.3 Lemma

(a) If f_i is obtained by composition from functions f_k and $f_{j1}, \ldots f_{jp}$ for which Γ is adequate, then Γ is adequate also for f_i.

(b) If f_i is obtained by primitive recursion from functions f_j and f_k for which Γ is adequate, then Γ is adequate also for f_i.

Proof: We leave (a) to the reader and do (b). Given a, b, and c with $f_i(a, b) = c$, for each $p \leq b$ let $c_p = f_i(a, p)$, so that $c_b = c$. Note that since f_i is obtained by primitive recursion from f_j and f_k, we have

$$c_0 = f_i(a, 0) = f_j(a)$$

and for all $p < b$ we have

$$c_{p'} = f_i(a, p') = f_k(a, p, f_i(a, p)) = f_k(a, p, c_p).$$

Since Γ is adequate for f_j and f_k,

(6a) $$\mathbf{f}_j(\mathbf{a}, \mathbf{0}) = \mathbf{c}_0$$

(6b) $$\mathbf{f}_k(\mathbf{a}, \mathbf{p}, \mathbf{c}_p) = \mathbf{c}_{p'}$$

are consequences of Γ. But (6a) and (5a) imply

(7a) $$\mathbf{f}_i(\mathbf{a}, \mathbf{0}) = \mathbf{c}_0$$

while (6b) and (5b) imply

(7b) $$\mathbf{f}i(\mathbf{a}, \mathbf{p}) = \mathbf{c}_p \rightarrow \mathbf{f}_i(\mathbf{a}, \mathbf{p}') = \mathbf{c}_{p'}.$$

But (7a) and (7b) for $p = 0$ imply $\mathbf{f}_i(\mathbf{a}, \mathbf{1}) = \mathbf{c}_1$, which with (7b) for $p = 1$ implies $\mathbf{f}_i(\mathbf{a}, \mathbf{2}) = \mathbf{c}_2$, which with (7b) for $p = 2$ implies $\mathbf{f}_i(\mathbf{a}, \mathbf{3}) = \mathbf{c}_3$, and so on up to $\mathbf{f}_i(\mathbf{a}, \mathbf{b}) = \mathbf{c}_b = \mathbf{c}$, which is what needed to be proved to show Γ adequate for $f_{i.}$

Since every f_i is either a basic function or obtained from earlier functions on our list by the processes covered by Lemma 11.3, the lemma implies that Γ is adequate for all the functions on our list, including $f_r = f$. In particular, if $f(m, n) = 0$, then $\mathbf{f}_r(\mathbf{m}, \mathbf{n}) = \mathbf{0}$ is implied by Γ, and hence so is $\forall y\ \mathbf{f}_r(\mathbf{m}, y) = \mathbf{0}$, which is $D(\mathbf{m})$.

Thus we have reduced the problem of determining whether for some n we have $f(m, n) = 0$ to the problem of determining whether Γ implies $D(\mathbf{m})$. That is, we have established that if the decision problem for logical implication were solvable, the nullity problem for f would be solvable, which it is known, as we have said, that it is not, assuming Church's thesis. Hence we have established the following result, assuming Church's thesis.

11.4 Theorem (Church's theorem). The decision problem for logical implication is unsolvable.

Problems

11.1 The *decision problem for validity* is the problem of devising an effective procedure that, applied to any sentence, would in a finite amount of time enable one to determine whether or not it is valid. Show that the unsolvability of the decision problem for implication (Theorem 11.2, or equivalently Theorem 11.4) implies the unsolvability of the decision problem for validity.

11.2 The *decision problem for satisfiability* is the problem of devising an effective procedure that, applied to any finite set of sentences, would in a finite amount of time enable one to determine whether or not it is satisfiable. Show that the unsolvability of the decision problem for implication (Theorem 11.2, or equivalently Theorem 11.4) implies the unsolvability of the decision problem for satisfiability.

The next several problems pertain specifically to section 11.1.

11.3 Show that

$$\forall w \forall v (Twv \leftrightarrow \exists y (Rwy \ \& \ Syv))$$

and

$$\forall u \forall v \forall y ((Suv \ \& \ Syv) \rightarrow u = y)$$

together imply

$$\forall u \forall v \forall w ((Twv \ \& \ Suv) \rightarrow Rwu).$$

11.4 Show that

$$\forall x (\sim Ax \rightarrow \sim\exists t (Bt \ \& \ Rtx))$$

and

$$\sim\exists x (Cx \ \& \ Ax)$$

together imply

$$\sim\exists t \exists x (Bt \ \& \ Cx \ \& \ Rtx).$$

11.5 The foregoing two problems state (in slightly simplified form in the case of the second one) two facts about implication that were used in the proof of Theorem 11.2. Where?

11.6 The *operating interval* for a Turing machine's computation beginning with input n consists of the numbers 0 through n together with the number of any time at which the machine has not (yet) halted, and of any square the machine visits during the course of its computations. Show that if the machine eventually halts, then the operating interval is the set of numbers between some $a \leq 0$ and some $b \geq 0$, and that if the machine never halts, then the operating interval consists either of all integers, or of all integers $\geq a$ for some $a \leq 0$.

11.7 A set of sentences Γ *finitely* implies a sentence D if D is true in every interpretation *with a finite domain* in which every sentence in Γ is true. *Trakhtenbrot's theorem* states that the decision problem for *finite* logical implication is unsolvable. Prove this theorem, assuming Turing's thesis.

The remaining problems pertain specifically to section 11.2.

11.8 Add to the theory Γ in the proof of Theorem 11.4 the sentence

$$\forall x \ \mathbf{0} \neq x' \ \& \ \forall x \forall y (x' = y' \rightarrow x = y).$$

Show that $\mathbf{m} \neq \mathbf{n}$ is then implied by Γ for all natural numbers $m \neq n$, where $\mathbf{m}$ is the usual numeral for m.

11.9 Add to the language of the theory Γ of the proof of Theorem 11.4 the symbol <, and add to Γ itself the sentence indicated in the preceding problem as well as the sentence

$$\forall x \sim \mathbf{0} < x \,\&\, \forall x \forall y(y < x' \leftrightarrow (y < x \vee y = x)).$$

Show that $\mathbf{m} < \mathbf{n}$ is implied by Γ whenever $m < n$ and $\sim\mathbf{m} < \mathbf{n}$ is implied by Γ whenever $m \geq n$, and that

$$\forall y\,(y < \mathbf{n} \rightarrow y = \mathbf{0} \vee y = \mathbf{1} \vee \ldots \vee y = \mathbf{m})$$

is implied by Γ whenever $n = m'$.

11.10 Let f be a recursive total function, and $f_1, \ldots, f_r$ a sequence of functions with last item $f_n = f$, such that each is either a basic function or is obtainable by earlier functions in the sequence by composition, primitive recursion, or minimization, *and all functions in the sequence are total.* (According to Problem 8.13, such a sequence exists for any recursive total function f.) Construct Γ and D as in the proof of Theorem 11.6, with the following modifications. Include the symbol < in the language, and the sentences indicated in the preceding two problems, and besides this, whenever f_i is obtained from f_j by minimization, include the sentence

$$\forall x \forall y((\mathbf{f}_j(x, y) = \mathbf{0} \,\&\, \forall z(z < y \rightarrow \mathbf{f}_j(x, z) \neq \mathbf{0})) \rightarrow \mathbf{f}_i(x) = y).$$

Show that the modified Γ is adequate for $f_r = f$.

11.11 (Requires the material of Chapter 8.) Show that there is a two-place primitive recursive function f such that the nullity problem for f is recursively unsolvable, or in other words, such that the set of x such that $\exists y(f(x, y) = \mathbf{0})$ is not recursive.

A distinction between unavoidable and lazy appeals to Church's thesis was made at the end of section 7.2*; though phrased there for recursive computability, it applies also to Turing computability.*

11.12 Distinguish the unavoidable from the lazy appeals to Turing's thesis in section 11.1.

11.13 Distinguish the unavoidable from the lazy appeals to Church's thesis in section 11.2.

12

Models

A model *of a set of sentences is any interpretation in which all sentences in the set are true. Section 12.1 discusses the sizes of the models a set of sentences may have (where by the* size *of a model is meant the size of its domain) and the number of models of a given size a set of sentences may have, introducing in the latter connection the important notion of* isomorphism. *Section 12.2 is devoted to examples illustrating the theory, with most pertaining to the important notion of an* equivalence relation. *Section 12.3 includes the statement of two major theorems about models, the* Löwenheim–Skolem (transfer) theorem *and the* (Tarski–Maltsev) compactness theorem, *and begins to illustrate some of their implications. The proof of the compactness theorem will be postponed until the next chapter. The Löwenheim–Skolem theorem is a corollary of compactness (though it also admits of an independent proof, to be presented in a later chapter, along with some remarks on implications of the theorem that have sometimes been thought 'paradoxical').*

12.1 The Size and Number of Models

By a *model* of a sentence or set of sentences we mean an interpretation in which the sentence, or every sentence in the set, comes out true. Thus Γ implies D if every model of Γ is a model of D, D is valid if *every* interpretation is a model of D, and Γ is unsatisfiable if *no* interpretation is a model of Γ.

By the *size* of a model we mean the size of its domain. Thus a model is called *finite*, *denumerable*, or whatever, if its domain is finite, denumerable, or whatever. A set of sentences is said to have *arbitrarily large finite models* if for every positive integer m there is a positive integer $n \geq m$ such that the set has a model of size n. Already in the empty language, with identity but no nonlogical symbols, where an interpretation is just a domain, one can write down sentences that have models only of some fixed finite size.

12.1 Example (A sentence with models only of a specified finite size). For each positive integer n there is a sentence I_n involving identity but no nonlogical symbols such that I_n will be true in an interpretation if and only if there are at least n distinct individuals in the domain of the interpretation. Then $J_n = {\sim}I_{n+1}$ will be true if and only if there are at most n individuals, and $K_n = I_n \,\&\, J_n$ will be true if and only if there are exactly n individuals.

There are actually several different sentences that could be used for I_n. A comparatively short one is the following:

$$\forall x_1 \forall x_2 \cdots \forall x_{n-1} \exists x_n (x_n \neq x_1 \,\&\, x_n \neq x_2 \,\&\, \ldots \,\&\, x_n \neq x_{n-1}).$$

Thus, for instance, I_3 may be written $\forall x \forall y \exists z (z \neq x \,\&\, z \neq y)$. For this to be true in an interpretation $\mathcal{M}$, it must be the case that for every p in the domain, if we added a constant c denoting p, then $\forall y \exists z (z \neq c \,\&\, z \neq y)$ would be true. For that to be true, it must be the case that for every q in the domain, if we added a constant d denoting q, then $\exists z (z \neq c \,\&\, z \neq d)$ would be true. For that to be true, it must be the case that for some r in the domain, if we added a constant e denoting r, then $e \neq c \,\&\, e \neq d$ would be true. For that, $e \neq c$ and $e \neq d$ would both have to be true, and for that, $e = c$ and $e = d$ would both have to be untrue. For that, the denotation r of e must be different from the denotations p and q of c and d. So for every p and q in the domain, there is an r in the domain different from both of them. Starting from any m_1 in the domain, and applying this last conclusion with $p = q = m_1$, there must be an r, which we call m_2, different from m_1. Applying the conclusion again with $p = m_1$ and $q = m_2$, there must be an r, which we call m_3, different from m_1 and m_2. So there are at least three distinct individuals m_1, m_2, m_3 in the domain.

The set Γ of *all* sentences I_n has only infinite models, since the number of elements in any model must be $\geq n$ for each finite n. On the other hand, any finite subset Γ_0 of Γ has a finite model, and indeed a model of size n, where n is the largest number for which I_n is in Γ. Can we find an example of a *finite* set of sentences that has only infinite models? If so, then we can in fact find a *single* sentence that has only infinite models, namely, the conjunction of all the sentences in the finite set. In fact, examples of single sentences that have only infinite models are known.

12.2 Example (A sentence with only infinite models). Let R be a two-place predicate. Then the following sentence A has a denumerable model but no finite models:

$$\forall x \exists y\, Rxy \,\&\, \forall x \forall y \sim (Rxy \,\&\, Ryx) \,\&\, \forall x \forall y \forall z ((Rxy \,\&\, Ryz) \rightarrow Rxz).$$

A has a denumerable model in which the domain is the natural numbers and the interpretation of the predicate is the usual strict less-than order relation on natural numbers. For every number there is one it is less than; no two numbers are less than each other; and if one number is less than a second and the second less than a third, then the first is less than the third. So all three conjuncts of A are true in this interpretation.

Now suppose there were a finite model $\mathcal{M}$ of A. List the elements of $|\mathcal{M}|$ as m_0, $m_1, \ldots, m_{k-1}$, where k is the number of elements in $|\mathcal{M}|$. Let $n_0 = m_0$. By the first conjunct of A (that is, by the fact that this conjunct is true in the interpretation) there must be some n in $|\mathcal{M}|$ such that $R^{\mathcal{M}}(n_0, n)$. Let n_1 be the first element on the list for which this is the case. So we have $R^{\mathcal{M}}(n_0, n_1)$. But by the second conjunct of A we do not have both $R^{\mathcal{M}}(n_0, n_1)$ and $R^{\mathcal{M}}(n_1, n_0)$, and so we do not have $R^{\mathcal{M}}(n_1, n_0)$. It follows that $n_1 \neq n_0$. By the first conjunct of A again there must be some n in $|\mathcal{M}|$ such that $R^{\mathcal{M}}(n_1, n)$. Let n_2 be the first element on the list for which this is the case, so we have $R^{\mathcal{M}}(n_1, n_2)$. By the third conjunct of A either $R^{\mathcal{M}}(n_0, n_1)$ fails or $R^{\mathcal{M}}(n_1, n_2)$ fails or $R^{\mathcal{M}}(n_0, n_2)$ holds, and since we do not have either of the first two disjuncts, we must have $R^{\mathcal{M}}(n_0, n_2)$. But by the second conjunct of A, $R^{\mathcal{M}}(n_0, n_2)$ and $R^{\mathcal{M}}(n_2, n_0)$ don't *both* hold, nor do both $R^{\mathcal{M}}(n_1, n_2)$ and $R^{\mathcal{M}}(n_2, n_1)$, so we have neither $R^{\mathcal{M}}(n_2, n_0)$ nor $R^{\mathcal{M}}(n_2, n_1)$. It follows that $n_2 \neq n_0$ and $n_2 \neq n_1$. Continuing in this way, we obtain n_3 different from all of n_0, n_1, n_2, then n_4 different from all of n_0, n_1, n_2, n_3, and so on. But by the time we get to n_k we will have exceeded the number of elements of $|\mathcal{M}|$. This shows that our supposition that $|\mathcal{M}|$ is finite leads to a contradiction. Thus A has a denumerable but no finite models.

When we ask how *many* different models a sentence or set of sentences may have of a given size, the answer is disappointing: there are always an unlimited number (a nonenumerable infinity) of models if there are any at all. To give a completely trivial example, consider the empty language, with identity but no nonlogical predicates, for which an interpretation is just a nonempty set to serve as domain. And consider the sentence $\exists x \forall y(y = x)$, which says there is just one thing in the domain. For any object a you wish, the interpretation whose domain is $\{a\}$, the set whose only element is a, is a model of this sentence. So for each real number, or each point on the line, we get a model.

Of course, these models all 'look alike': each consists of just one thing, sitting there doing nothing, so to speak. The notion of *isomorphism*, which we are about to define, is a technically precise way of saying what is meant by 'looking alike' in the case of *non*trivial languages. Two interpretations $\mathcal{P}$ and $\mathcal{Q}$ of the same language L are *isomorphic* if and only if there is a *correspondence* j between individuals p in the domain $|\mathcal{P}|$ and individuals q in the domain $|\mathcal{Q}|$ subject to certain conditions. (The definition of correspondence, or total, one-to-one, onto function, has been given in the problems at the end of Chapter 1.) The further conditions are that for every n-place predicate R and all $p_1, \ldots, p_n$ in $|\mathcal{P}|$ we have

$$\text{(I1)} \qquad R^{\mathcal{P}}(p_1, \ldots, p_n) \quad \text{if and only if} \quad R^{\mathcal{Q}}(j(p_1), \ldots, j(p_n))$$

and for every constant c we have

$$\text{(I2)} \qquad j(c^{\mathcal{P}}) = c^{\mathcal{Q}}.$$

If function symbols are present, it is further required that for every n-place function symbol f and all $p_1, \ldots, p_n$ in $|\mathcal{P}|$ we have

$$\text{(I3)} \qquad j(f^{\mathcal{P}}(p_1, \ldots, p_n)) = f^{\mathcal{Q}}(j(p_1), \ldots, j(p_n)).$$

12.3 Example (Inverse order and mirror arithmetic). Consider the language with a single two-place predicate $<$, the interpretation with domain the natural numbers $\{0, 1, 2, 3, \ldots\}$ and with $<$ denoting the usual strict less-than order relation, and by contrast the interpretation with domain the nonpositive integers $\{0, -1, -2, -3, \ldots\}$ and with $<$ denoting the usual strict greater-than relation. The correspondence associating n with $-n$ is an isomorphism, since m is less than n if and only if $-m$ is greater than $-n$, as required by (I1).

If we also let **0** denote zero, let $'$ denote the predecessor function, which takes x to $x-1$, let $+$ denote the addition function, and let $\cdot$ denote the function taking x and y to the negative of their product, $-xy$, then we obtain an interpretation isomorphic to the standard interpretation of the language of arithmetic. For the following equations show (I3) to be fulfilled:

$$\begin{aligned} -x-1 &= -(x+1) \\ (-x)+(-y) &= -(x+y) \\ -(-x)(-y) &= -xy. \end{aligned}$$

Generalizing our completely trivial example, in the case of the empty language, where an interpretation is just a domain, two interpretations are isomorphic if and only if there is a correspondence between their domains (that is, if and only if they are equinumerous, as defined in the problems at the end of Chapter 1). The analogous property for nonempty languages is stated in the next result.

12.4 Proposition. Let X and Y be sets, and suppose there is a correspondence j from X to Y. Then if $\mathcal{Y}$ is any interpretation with domain Y, there is an interpretation $\mathcal{X}$ with domain X such that $\mathcal{X}$ is isomorphic to $\mathcal{Y}$. In particular, for any interpretation with a finite domain having n elements, there is an isomorphic interpretation with domain the set $\{0, 1, 2, \ldots, n-1\}$, while for any interpretation with a denumerable domain there is an isomorphic interpretation with domain the set $\{0, 1, 2, \ldots\}$ of natural numbers.

Proof: For each relation symbol R, let $R^{\mathcal{X}}$ be the relation that holds for $p_1, \ldots, p_n$ in X if and only if $R^{\mathcal{Y}}$ holds for $j(p_1), \ldots, j(p_n)$. This makes (I1) hold automatically. For each constant c, let $c^{\mathcal{X}}$ be the unique p in X such that $j(p) = c^{\mathcal{Y}}$. (There will be such a p because j is onto, and it will be unique because j is one-to-one.) This makes (I2) hold automatically. If function symbols are present, for each function symbol f, let $f^{\mathcal{X}}$ be the function on X whose value for $p_1, \ldots, p_n$ in X is the unique p such that $j(p) = f^{\mathcal{Y}}(j(p_1), \ldots, j(p_n))$. This makes (I3) hold automatically.

The next result is a little more work. Together with the preceding, it implies what we hinted earlier, that a sentence or set of sentences has an unlimited number of models if it has any models at all: given one model, by the preceding proposition there will be an unlimited number of interpretations isomorphic to it, one for each set equinumerous with its domain. By the following result, these isomorphic interpretations will all be models of the given sentence or set of sentences.

12.5 Proposition (Isomorphism lemma). If there is an isomorphism between two interpretations $\mathcal{P}$ and $\mathcal{Q}$ of the same language L, then for every sentence A of L we have

(1) $$\mathcal{P} \models A \quad \text{if and only if} \quad \mathcal{Q} \models A.$$

Proof: We first consider the case where identity and function symbols are absent, and proceed by induction on complexity. First, for an atomic sentence involving a nonlogical predicate R and constants $t_1, \ldots, t_n$, the atomic clause in the definition of truth gives

$$\mathcal{P} \models R(t_1, \ldots, t_n) \quad \text{if and only if} \quad R^{\mathcal{P}}\big(t_1^{\mathcal{P}}, \ldots, t_n^{\mathcal{P}}\big)$$
$$\mathcal{Q} \models R(t_1, \ldots, t_n) \quad \text{if and only if} \quad R^{\mathcal{Q}}\big(t_1^{\mathcal{Q}}, \ldots, t_n^{\mathcal{Q}}\big)$$

while the clause (I1) in the definition of isomorphism gives

$$R^{\mathcal{P}}\big(t_1^{\mathcal{P}}, \ldots, t_n^{\mathcal{P}}\big) \quad \text{if and only if} \quad R^{\mathcal{Q}}\big(j\big(t_1^{\mathcal{P}}\big), \ldots, j\big(t_n^{\mathcal{P}}\big)\big)$$

and the clause (I2) in the definition of isomorphism gives

$$R^{\mathcal{Q}}\big(j\big(t_1^{\mathcal{P}}\big), \ldots, j\big(t_n^{\mathcal{P}}\big)\big) \quad \text{if and only if} \quad R^{\mathcal{Q}}\big(t_1^{\mathcal{Q}}, \ldots, t_n^{\mathcal{Q}}\big).$$

Together the four displayed equivalences give (1) for $R(t_1, \ldots, t_n)$.

Second, suppose (1) holds for less complex sentences than $\sim F$, including the sentence F. Then (1) for $\sim F$ is immediate from this assumption together with the negation clause in the definition of truth, by which we have

$$\mathcal{P} \models \sim F \quad \text{if and only if} \quad \text{not } \mathcal{P} \models F$$
$$\mathcal{Q} \models \sim F \quad \text{if and only if} \quad \text{not } \mathcal{Q} \models F.$$

The case of junctions is similar.

Third, suppose (1) holds for less complex sentences than $\forall x\, F(x)$, including sentences of the form $F(c)$. For any element p of $|\mathcal{P}|$, if we extend the language by adding a new constant c and extend the interpretation $\mathcal{P}$ so that c denotes p, then there is one and only one way to extend the interpretation $\mathcal{Q}$ so that j remains an isomorphism of the extended interpretations; namely, we extend the interpretation $\mathcal{Q}$ so that c denotes $j(p)$, and therefore clause (I2) in the definition of isomorphism still holds for the extended language. By our assumption that (1) holds for $F(c)$ it follows on the one hand that

$$(2) \qquad \mathcal{P} \models F[p] \quad \text{if and only if} \quad \mathcal{Q} \models F[j(p)].$$

By the universal quantifier clause in the definition of truth

$$\mathcal{P} \models \forall x\, F(x) \quad \text{if and only if} \quad \mathcal{P} \models F[p] \text{ for all } p \text{ in } |\mathcal{P}|.$$

Hence

$$\mathcal{P} \models \forall x\, F(x) \quad \text{if and only if} \quad \mathcal{Q} \models F[j(p)] \text{ for all } p \text{ in } |\mathcal{P}|.$$

On the other hand, again by the universal quantifier clause in the definition of truth we have

$$\mathcal{Q} \models \forall x\, F(x) \quad \text{if and only if} \quad \mathcal{Q} \models F[q] \text{ for all } q \text{ in } |\mathcal{Q}|.$$

But since j is a correspondence, and therefore is onto, *every* q in $|\mathcal{Q}|$ is of the form $j(p)$, and (1) follows for $\forall x\, F(x)$. The existential-quantifier case is similar.

If identity is present, we have to prove (1) also for atomic sentences involving $=$. That is, we have to prove

$$p_1 = p_2 \quad \text{if and only if} \quad j(p_1) = j(p_2).$$

But this is simply the condition that j is one-to-one, which is part of the definition of being a correspondence, which in turn is part of the definition of being an isomorphism.

If function symbols are present, we must first prove as a preliminary that for any closed term t we have

$$j(t^{\mathcal{P}}) = t^{\mathcal{Q}}. \tag{3}$$

This is proved by induction on complexity of terms. For constants we have (3) by clause (I2) in the definition of isomorphism. And supposing (3) holds for $t_1, \ldots, t_n$, then it holds for $f(t_1, \ldots, t_n)$ since by clause (I3) in the definition of isomorphism we have

$$\begin{aligned} j((f(t_1, \ldots, t_n))^{\mathcal{P}}) &= j\big(f^{\mathcal{P}}\big(t_1^{\mathcal{P}}, \ldots, t_n^{\mathcal{P}}\big)\big) \\ &= f^{\mathcal{Q}}\big(j\big(t_1^{\mathcal{P}}\big), \ldots, j\big(t_n^{\mathcal{P}}\big)\big) = f^{\mathcal{Q}}\big(t_1^{\mathcal{Q}}, \ldots, t_n^{\mathcal{Q}}\big) = (f(t_1, \ldots, t_n))^{\mathcal{Q}}. \end{aligned}$$

The proof given above for the atomic case of (1) now goes through even when the t_i are complex closed terms rather than constants, and no further changes are required in the proof.

12.6 Corollary (Canonical-domains lemma).

(a) Any set of sentences that has a finite model has a model whose domain is the set $\{0, 1, 2, \ldots, n\}$ for some natural number n.

(b) Any set of sentences having a denumerable model has a model whose domain is the set $\{0, 1, 2, \ldots\}$ of natural numbers.

Proof: Immediate from Propositions 12.4 and 12.5.

Two models that are isomorphic are said to be of the same *isomorphism type*. The intelligent way to count the models of a given size that a sentence has is to count not literally the number of models (which is always a nonenumerable infinity if it is nonzero), but the number of isomorphism types of models. The import of the rather abstract results of this section should become clearer as they are illustrated concretely in the next section.

12.2 Equivalence Relations

Throughout this section we will work with a language whose only nonlogical symbol is a single two-place predicate $\equiv$. We will write $x \equiv y$ for what officially ought to be $\equiv(x, y)$. Our interest will be in models—and especially in denumerable models—of the following sentence *Eq* of the language:

$$\begin{aligned} &\forall x\, x \equiv x \,\& \\ &\forall x \forall y (x \equiv y \to y \equiv x) \,\& \\ &\forall x \forall y \forall z ((x \equiv y \,\&\, y \equiv z) \to x \equiv z). \end{aligned}$$

Such a model $\mathcal{X}$ will consist of a nonempty set X and a two-place relation $\equiv^{\mathcal{X}}$ or E on X. In order to make the three clauses of *Eq* true, E will have to have three properties.

Namely, for all a, b, c in X we must have the following:

(E1) *Reflexivity*: $a\ E\ a$.
(E2) *Symmetry*: If $a\ E\ b$ then $b\ E\ a$.
(E3) *Transitivity*: If $a\ E\ b$ and $b\ E\ c$ then $a\ E\ c$.

A relation with these properties is called an *equivalence relation* on X.

One way to get an equivalence relation on X is to start with what is called a *partition* of X. This is a set Π of nonempty subsets of X such that the following hold:

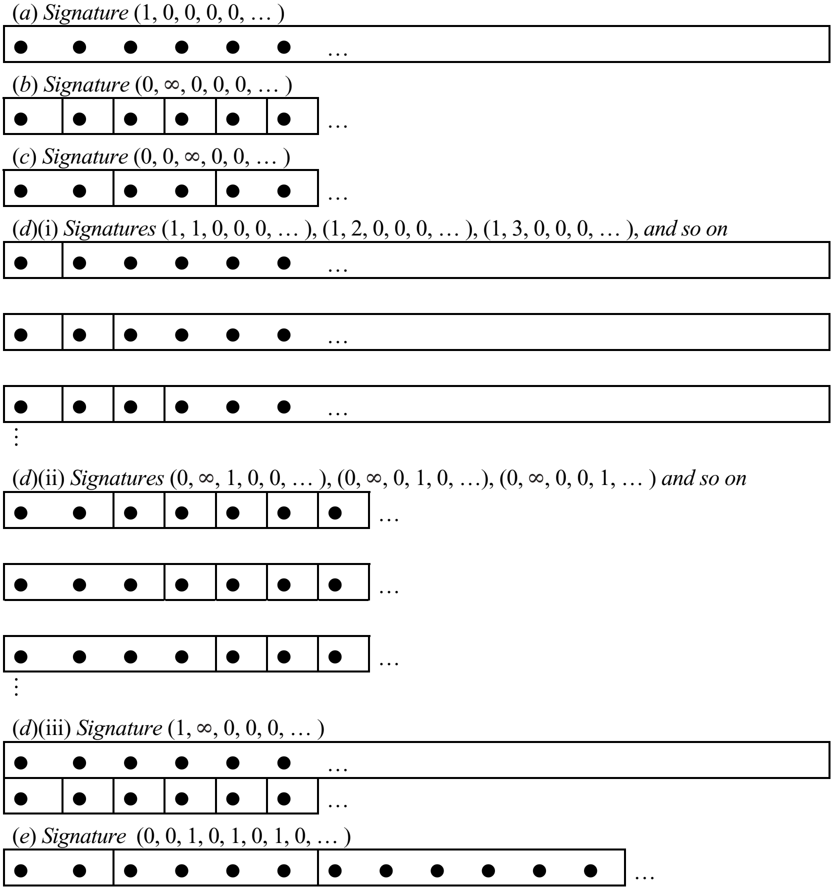
(P1) *Disjointness*: If A and B are in Π, then either $A = B$ or A and B have no elements in common.
(P2) *Exhaustiveness*: Every a in X belongs to some A in Π.

The sets in Π are called the *pieces* of the partition.

Given a partition, define $a E b$ to hold if a and b are in the same piece of the partition, that is, if, for some A in Π, a and b are both in A. Now by (P2), a is in *some* A in Π. To say a and a are 'both' in A is simply to say a is in A twice, and since it was true the first time, it will be true the second time also, showing that $a\ E\ a$, and that (E1) holds. If $a E b$, then a and b are both in some A in Π, and to say that b and a are both in A is to say the same thing in a different order, and is equally true, showing that $b\ E\ a$, and that (E2) holds. Finally, if $a E b$ and $b\ E\ c$, then a and b are both in some A in Π and b and c are both in some B in Π. But by (P1), since A and B have the common element b, they are in fact the same, so a and c are both in $A = B$, and $a\ E\ c$, showing that (E3) holds. So E is an equivalence relation, called the equivalence relation *induced* by the partition.

Actually, this is in a sense the *only* way to get an equivalence relation: every equivalence relation is induced by a partition. For suppose E is any such relation; for any a in X let $[a]$ be the *equivalence class* of a, the set of all b in X such that $a E b$; and let Π be the set of all these equivalence classes. We claim Π is a partition. Certainly any element a of X is in some A in Π, namely, a is in $[a]$, by (E1). So (P2) holds. As for (P1), if $[a]$ and $[b]$ have a common element c, we have $a\ E\ c$ and $b\ E\ c$, and having $b\ E\ c$, by (E2) we have also $c\ E\ b$, and then, having $a\ E\ c$ and $c\ E\ b$, by (E3) we have also $a E b$, and by (E2) again we have also $b\ E\ a$. But then if d is any element of $[a]$, having $a\ E\ d$ and $b\ E\ a$, by (E3) again we have $b\ E\ d$, and d is in $[b]$. In exactly the same way, any element of $[b]$ is in $[a]$, and $[a] = [b]$. So Π is a partition, as claimed. We also claim the original E is just the equivalence relation induced by this partition Π. For along the way we have shown that if $a E b$ then a and b belong to the same piece $[a] = [b]$ of the partition, while of course if b belongs to the same piece $[a]$ of the partition that a does, then we have $a E b$, so E is the equivalence relation induced by this partition.

We can draw a picture of a denumerable model of *Eq*, by drawing dots to represent elements of X with boxes around those that are in the same equivalence class. We can also describe such a model by describing its *signature*, the infinite sequence of numbers whose 0th entry is the number (which may be 0, 1, 2, . . . , or infinite) of equivalence classes having infinitely many elements and whose nth entry for $n > 0$ is the number of equivalence classes with exactly n elements. The examples to follow are illustrated by pictures for equivalence relations of a variety of different signatures in Figure 12-1.



(*a*) *Signature* (1, 0, 0, 0, 0, …)

(*b*) *Signature* (0, ∞, 0, 0, 0, …)

(*c*) *Signature* (0, 0, ∞, 0, 0, …)

(*d*)(i) *Signatures* (1, 1, 0, 0, 0, …), (1, 2, 0, 0, 0, …), (1, 3, 0, 0, 0, …), *and so on*

(*d*)(ii) *Signatures* (0, ∞, 1, 0, 0, …), (0, ∞, 0, 1, 0, …), (0, ∞, 0, 0, 1, …) *and so on*

(*d*)(iii) *Signature* (1, ∞, 0, 0, 0, …)

(*e*) *Signature* (0, 0, 1, 0, 1, 0, 1, 0, …)

Figure 12-1. Equivalence relations.

12.7 Example (A promiscuous model). Let Γ_a be the set containing *Eq* and the following sentence E_a:

$$\forall x \forall y\, x \equiv y.$$

A denumerable model of Γ_a consists of a denumerable set X with an equivalence relation E in which *all* elements are in the same equivalence class, as in Figure 12-1(a). We claim all such models are isomorphic. Indeed, if

$$X = \{a_1, a_2, a_3, \ldots\} \quad \text{and} \quad Y = \{b_1, b_2, b_3, \ldots\}$$

are any two denumerable sets, if $\mathcal{X}$ is the model with domain X and $\equiv^{\mathcal{X}}$ the relation that holds among *all* pairs a_i, a_j of elements of X, and if $\mathcal{Y}$ is the model with domain Y and $\equiv^{\mathcal{Y}}$ the relation that holds among *all* pairs b_i, b_j of elements of Y, then the function sending a_i to

b_i is an isomorphism between $\mathcal{X}$ and $\mathcal{Y}$. Condition (I1) in the definition of isomorphism—the only applicable condition—says that we must in all cases have $a_i \equiv^{\mathcal{X}} a_j$ if and only if $f(a_i) \equiv^{\mathcal{Y}} f(a_j)$; and of course we do, since we always have both $a_i \equiv^{\mathcal{X}} a_j$ and $b_i \equiv^{\mathcal{Y}} b_j$. Thus Γ_a has only *one* isomorphism type of denumerable model.

12.8 Example (An eremitic model). Let Γ_b be the set containing *Eq* and the following sentence E_b:

$$\forall x \forall y\, (x \equiv y \leftrightarrow x = y).$$

A denumerable model of Γ_b consists of a denumerable set X with an equivalence relation E in which each element is equivalent only to itself, so each equivalence class consists of but a single element, as in Figure 12-1(b). Again any two such models are isomorphic. With the notation as in the preceding example, this time we have $a_i \equiv^{\mathcal{X}} a_j$ if and only if $f(a_i) \equiv^{\mathcal{Y}} f(a_j)$, because we only have $a_i \equiv^{\mathcal{X}} a_j$ when $i = j$, which is precisely when we have $b_i \equiv^{\mathcal{Y}} b_j$.

12.9 Example (Two isomorphism types). Let Γ_{ab} be the set containing *Eq* and the disjunction $E_a \vee E_b$. Any model of Γ_{ab} must be either a model of Γ_a or one of Γ_b, and all models of either are models of Γ_{ab}. Now all denumerable models of Γ_a are isomorphic to each other, and all denumerable models of Γ_b are isomorphic to each other. But a model of Γ_a cannot be isomorphic to a model of Γ_b, by the isomorphism lemma, since E_a is true in the former and false in the latter, and inversely for E_b. So Γ_{ab} has exactly *two* isomorphism types of denumerable model.

12.10 Example (An uxorious model). Let Γ_c be the set containing *Eq* and the following sentence E_c:

$$\forall x \exists y (x \neq y \,\&\, x \equiv y \,\&\, \forall z (z \equiv x \rightarrow (z = x \vee z = y))).$$

A denumerable model of Γ_c consists of a denumerable set X with an equivalence relation E in which each element is equivalent to just one other element than itself, so each equivalence class consists of exactly two elements, as in Figure 12-1(c). Again there is only one isomorphism type of denumerable model. If we renumber the elements of X so that a_2 is the equivalent of a_1, a_4 of a_3, and so on, and if we similarly renumber the elements of Y, again the function $f(a_i) = b_i$ will be an isomorphism.

12.11 Example (Three isomorphism types). Let Γ_{abc} be the set containing *Eq* and the disjunction $E_a \vee E_b \vee E_c$. Then Γ_{abc} has three isomorphism types of denumerable models. The reader will see the pattern emerging: we can get an example with n isomorphism types of denumerable models for any positive integer n.

12.12 Example (Denumerably many isomorphism types). Let Γ_d be the set containing *Eq* and the following sentence E_d:

$$\forall x \forall y ((\exists u (u \neq x \,\&\, u \equiv x) \,\&\, \exists v (v \neq y \,\&\, v \equiv y)) \rightarrow x \equiv y).$$

A denumerable model of Γ_d will consist of a denumerable set X with an equivalence relation in which any two elements a and b that are not *isolated*, that is, that are such that each is equivalent to something other than itself, are equivalent to each other. Here there are a number of possible pictures. It could be that all elements are equivalent, or that all elements are isolated, as in Figure 12-1(a) or (b). It could also be the case that there is one isolated element with all the other elements being equivalent. Or there could be two isolated elements with all the other elements being equivalent. Or three, and so on, as in Figure 12-1(d)(i).

There are further possibilities. For, supposing there are *infinitely many* isolated elements, the remaining equivalence class, consisting of all nonisolated elements, may contain two or three or . . . elements, as in Figure 12-1(d)(ii)—or it could contain zero, but that is Figure 12-1(b) again. Finally there is the possibility (whose picture takes two lines to draw) of infinitely many isolated elements plus an infinite class of other elements, all equivalent to each other, as in Figure 12-1(d)(iii).

Any two models corresponding to the same picture (or, what comes to the same thing, the same signature) are isomorphic. If there are only n isolated elements, renumber so that these are a_1 through a_n. If there are only n nonisolated elements, renumber so that these are a_1 through a_n instead. And if there are infinitely many of each, renumber so that a_1, a_3, a_5, . . . are the isolated ones, and a_2, a_4, a_6, . . . the nonisolated ones. Renumber the b_i similarly, and then, as always, the function $f(a_i) = b_i$ can be checked to be an isomorphism. No two models corresponding to different pictures are isomorphic, for if a is nonisolated, a satisfies the formula

$$\exists y(y \neq x \,\&\, y \equiv x).$$

So by the isomorphism lemma, if f is an isomorphism, $f(a)$ must also satisfy this formula, and so must be nonisolated. And for the same reason, applied to the negation of this formula, if a is isolated, $f(a)$ must be isolated. So an isomorphism must carry nonisolated to nonisolated and isolated to isolated elements, and the numbers of nonisolated and of isolated elements must be the same in both models. Here, then, is an example where there are denumerably many of isomorphism types of denumerable models.

12.13 Example (Nonenumerably many isomorphism types). The sentence *Eq* all by itself has nonenumerably many isomorphism types of denumerable models. For any infinite set of positive integers S there is a model in which there is exactly one equivalence class with exactly n elements for each n in S, and no equivalence class with exactly n elements for any n not in S. For instance, if S is the set of even numbers, the model will look like Figure 12-1(e). We leave it to the reader to show how the isomorphism lemma can be used to show that no two models corresponding to different sets S are isomorphic. Since there are nonenumerably many such sets, there are nonenumerably many isomorphism types of models.

12.3 The Löwenheim–Skolem and Compactness Theorems

We have seen that there are sentences that have only infinite models. One might wonder whether there are sentences that have only *nonenumerable* models. We have also seen that there are enumerable sets of sentences that have only infinite models, though every finite subset has a finite model. One might wonder whether there are sets of sentences that have *no models at all*, though every finite subset has a model. The answer to both these questions is negative, according to the following pair of theorems. They are basic results in the theory of models, with many implications about the existence, size, and number of models.

12.14 Theorem (Löwenheim–Skolem theorem). If a set of sentences has a model, then it has an enumerable model.

12.15 Theorem (Compactness theorem). If every finite subset of a set of sentences has a model, then the whole set has a model.

We explore a few of the implications of these theorems in the problems at the end this chapter. We stop here just to note three immediate implications.

12.16 Corollary (Overspill principle). If a set of sentences has arbitrarily large finite models, then it has a denumerable model.

Proof: Let Γ be a set of sentences having arbitrarily large finite models, and for each m let I_m be the sentence with identity but no nonlogical symbols considered in Example 12.1, which is true in a model if and only if the model has size $\geq m$. Let

$$\Gamma^* = \Gamma \cup \{I_1, I_2, I_3, \dots\}$$

be the result of adding all the I_m to Γ. Any finite subset of Γ^* is a subset of $\Gamma \cup \{I_1, I_2, \dots, I_m\}$ for some m, and since Γ has a model of size $\geq m$, such a set has a model. By the compactness theorem, therefore, Γ^* has a model. Such a model is of course a model of Γ, and being also a model of each I_m, it has size $\geq m$ for all finite m, and so is infinite. By the Löwenheim–Skolem theorem, we could take it to be enumerable.

A set Γ of sentences is (*implicationally*) *complete* if for every sentence A in its language, either A or $\sim A$ is a consequence of Γ, and *denumerably categorical* if any two denumerable models of Γ are isomorphic.

12.17 Corollary (Vaught's test). If Γ is a denumerably categorical set of sentences having no finite models, then Γ is complete.

Proof: Suppose Γ is not complete, and let A be some sentence in its language such that neither A nor $\sim A$ is a consequence of Γ. Then both $\Gamma \cup \{\sim A\}$ and $\Gamma \cup \{A\}$ are satisfiable, and by the Löwenheim–Skolem theorem they have enumerable models $\mathcal{P}^-$ and $\mathcal{P}^+$. Since Γ has no finite models, $\mathcal{P}^-$ and $\mathcal{P}^+$ must be denumerable. Since Γ is denumerably categorical, they must be isomorphic. But by the isomorphism lemma, since A is untrue in one and true in the other, they can*not* be isomorphic. So the assumption that Γ is not complete leads to a contradiction, and Γ must be complete after all.

Thus if Γ is any of the examples of the preceding section in which we found there was only one isomorphism type of denumerable model, then adding the sentences $I_1, I_2, I_3, \dots$ to Γ (in order to eliminate the possibility of finite models) produces an example that is complete.

The Löwenheim–Skolem theorem also permits a sharpening of the statement of the canonical-domains lemma (Lemma 12.6).

12.18 Corollary (Canonical-domains theorem).

(a) Any set of sentences that has a model, has a model whose domain is either the set of natural numbers $< n$ for some positive n, or else the set of all natural numbers.

(b) Any set of sentences not involving function symbols or identity that has a model, has a model whose domain is the set of all natural numbers.

Proof: (a) is immediate from the Löwenheim–Skolem theorem and Corollary 12.6. For (b), given a set of sentences Γ not involving function symbols or identity, if Γ has a model, apply part (a) to get, at worst, a model $\mathcal{Y}$ with domain the finite set $\{0, 1, \ldots, n-1\}$ for some n. Let f be the function from the set of all natural numbers to this finite set given by $f(m) = \min(m, n-1)$. Define an interpretation $\mathcal{X}$ with domain the set of all natural numbers by assigning to each k-place relation symbol R as denotation the relation $R^{\mathcal{X}}$ that holds for $p_1, \ldots, p_k$ if and only if $R^{\mathcal{Y}}$ holds for $f(p_1), \ldots, f(p_k)$. Then f has all the properties of an isomorphism except for not being one-to-one. Examining the proof of the isomorphism lemma (Proposition 12.5), which tells us the same sentences are true in isomorphic interpretations, we see that the property of being one-to-one was used *only* in connection with sentences involving identity. Since the sentences in Γ do not involve identity, they will be true in $\mathcal{X}$ because they are true in $\mathcal{Y}$.

The remainder of this section is devoted to an advance description of what will be done in the following two chapters, which contain proofs of the Löwenheim–Skolem and compactness theorems and a related result. Our preview is intended to enable the readers who are familiar with the contents of an introductory textbook to decide how much of this material they need or want to read. The next chapter, Chapter 13, is devoted to a proof of the compactness theorem. Actually, the proof shows that if every finite subset of a set Γ has a model, then Γ has an *enumerable* model. This version of the compactness theorem implies the Löwenheim–Skolem theorem, since if a set has a model, so does every subset, and in particular every finite subset. An optional final section 13.5 considers what happens if we admit *nonenumerable* languages. (It turns out that the compactness theorem still holds, but the 'downward' Löwenheim–Skolem theorem fails, and one gets instead an 'upward' theorem to the effect that any set of sentences having an infinite model has a nonenumerable model.)

Every introductory textbook introduces some notion of a *deduction* of a sentence D from a finite set of sentences Γ. The sentence D is defined to be *deducible* from a finite set Γ if and only if there is a deduction of the sentence from the set. A deduction from a subset of a set always counts as a deduction from that set itself, and a sentence D is defined to be *deducible* from an infinite set Γ if and only if it is deducible from some finite subset. A sentence D is defined to be *demonstrable* if it is deducible from the empty set of sentences $\varnothing$, and a set of sentences Γ is defined to be *inconsistent* if the constant false sentence $\perp$ is deducible from it. The better introductory textbooks include proofs of the *soundness* theorem, according to which if D is deducible from Γ, then D is a consequence of Γ (from which it follows that if D is demonstrable, then D is valid, and that if Γ is inconsistent, then Γ is unsatisfiable), and of the *Gödel completeness theorem*, according to which, conversely, if D is a consequence of Γ, then D is deducible from Γ (from which it follows that if D is valid, then D is demonstrable, and that if Γ is unsatisfiable, then Γ is inconsistent). Since by definition a set is consistent if and only if every finite subset is, it follows that a set is satisfiable if and only if every finite subset is: the compactness theorem follows from the soundness and completeness theorems. Actually, the proof of completeness shows that if Γ is consistent, then Γ has an *enumerable* model, so the form of the compactness theorem implying the Löwenheim–Skolem theorem follows.

In Chapter 14 we introduce a notion of deduction of the kind used in advanced, rather than introductory, works on logic, and prove soundness and completeness for it. However, rather than derive the compactness theorem (and thence the Löwenheim–Skolem theorem) from soundness and completeness, we obtain completeness in Chapter 14 from the main lemma used to obtain compactness in Chapter 13. Thus our proof of the compactness theorem (and similarly the Löwenheim–Skolem theorem) does not mention the notion of deduction any more than does the statement of the theorem itself. For the reader who is familiar with a proof of the soundness and completeness theorems, however, Chapter 14 is optional and Chapter 13 (containing the main lemma) with it, since the compactness theorem (and thence the Löwenheim–Skolem theorem) does follow. It does not matter if the notion of deduction with which such a reader is familiar is different from ours, since *no reference to the details of any particular deduction procedure is made outside Chapter 14* (except in one optional section at the end of the chapter after that, Chapter 15). All that matters for our later work is that there is some procedure or other of deduction that is sound and complete, and—for purposes of later application of our work on computability to logic—is such that one can effectively decide whether or not a given finite object is or is not a deduction of a given sentence D from a given finite set of sentences Γ. And this last feature is shared by all deduction procedures in all works on logic, introductory or advanced, ours included.

Problems

12.1 By the *spectrum* of a sentence C (or set of sentences Γ) is meant the set of all positive integers n such that C (or Γ) has a finite model with a domain having exactly n elements. Consider a language with just two nonlogical symbols, a one-place predicate P and a one-place function symbol f. Let A be the following sentence:

$$\begin{aligned}&\forall x_1 \forall x_2 (f(x_1) = f(x_2) \rightarrow x_1 = x_2) \,\&\\ &\forall y \exists x (f(x) = y) \,\&\\ &\forall x \forall y (f(x) = y \rightarrow (Px \leftrightarrow {\sim} Py)).\end{aligned}$$

Show that the spectrum of A is the set of all even positive integers.

12.2 Give an example of a sentence whose spectrum is the set of all odd positive integers.

12.3 Give an example of a sentence whose spectrum is the set of all positive integers that are perfect squares.

12.4 Give an example of a sentence whose spectrum is the set of all positive integers divisible by three.

12.5 Consider a language with just one nonlogical symbol, a two-place predicate Q. Let $\mathcal{U}$ be the interpretation in which the domain consists of the four sides of a square, and the denotation of Q is the relation between sides of being parallel. Let $\mathcal{V}$ be the interpretation in which the domain consists of the four vertices of a square, and the denotation of Q is the relation between vertices of being diagonally opposite. Show that $\mathcal{U}$ and $\mathcal{V}$ are isomorphic.

12.6 Consider a language with just one nonlogical symbol, a two-place predicate $<$. Let $\mathcal{Q}$ be the interpretation in which the domain is the set of real numbers *strictly greater than zero and strictly less than one* and the denotation of $<$ is the usual order relation. Let $\mathcal{R}$ be the interpretation in which the domain is the set of *all* real numbers and the denotation of $<$ is the usual order relation. Show that $\mathcal{Q}$ and $\mathcal{R}$ are isomorphic.

12.7 Let L be a language whose only nonlogical symbols are a two-place function symbol § and a two-place predicate $<$. Let $\mathcal{P}$ be the interpretation of this language in which the domain is the set of *positive* real numbers, the denotation of § is the usual *multiplication* operation, and the denotation of $<$ is the usual order relation. Let $\mathcal{Q}$ be the interpretation of this language in which the domain is the set of *all* real numbers, the denotation of § is the usual *addition* operation, and the denotation of $<$ is the usual order relation. Show that $\mathcal{P}$ and $\mathcal{Q}$ are isomorphic.

12.8 Write $\mathcal{A} \cong \mathcal{B}$ to indicate that $\mathcal{A}$ is isomorphic to $\mathcal{B}$. Show that for all interpretations $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ of the same language the following hold:

(a) $\mathcal{A} \cong \mathcal{A}$;

(b) if $\mathcal{A} \cong \mathcal{B}$, then $\mathcal{B} \cong \mathcal{A}$;

(c) if $\mathcal{A} \cong \mathcal{B}$ and $\mathcal{B} \cong \mathcal{C}$, then $\mathcal{A} \cong \mathcal{C}$.

12.9 By *true arithmetic* we mean the set Γ of all sentences of the language of arithmetic that are true in the standard interpretation. By a *nonstandard model of arithmetic* we mean a model of this Γ that (unlike the model in Example 12.3) is not isomorphic to the standard interpretation. Let Δ be the set of sentences obtained by adding a constant c to the language and adding the sentences $c \neq \mathbf{0}$, $c \neq \mathbf{1}$, $c \neq \mathbf{2}$, and so on, to Γ. Show that any model of Δ would give us a nonstandard model of arithmetic.

12.10 Consider the language with just the one nonlogical symbol $\equiv$ and the sentence Eq whose models are precisely the sets with equivalence relations, as in the examples in section 12.2.

(a) For each n, indicate how to write down a sentence B_n such that the models of Eq & B_n will be sets with equivalence relations *having at least n equivalence classes*.

(b) For each n, indicate how to write down a formula $F_n(x)$ such that in a model of Eq, an element a of the domain will satisfy $F_n(x)$ if and only if there are at least n elements in the equivalence class of a.

(c) For each n, indicate how to write down a sentence C_n that is true in a model of Eq if and only if there are exactly n equivalence classes.

(d) For each n, indicate how to write down a formula $G_n(x)$ that is satisfied by an element of the domain if and only if its equivalence class has exactly n elements.

12.11 For each m and n indicate how to write down a sentence D_{mn} that is true in a model of Eq if and only if there are at least m equivalence classes with exactly n elements.

12.12 Show that if two models of *Eq* are isomorphic, then the equivalence relations of the models have the same signature.

12.13 Suppose E_1 and E_2 are equivalence relations on denumerable sets X_1 and X_2 both having the signature $\sigma(n) = 0$ for $n \geq 1$ and $\sigma(0) = \infty$, that is, both having infinitely many equivalence classes, all infinite. Show that the models involved are isomorphic.

12.14 Show that two denumerable models of *Eq* are isomorphic if and only if they have the same signature.

In the remaining problems you may, when relevant, use the Löwenheim–Skolem and compactness theorems, even though the proofs have been deferred to the next chapter.

12.15 Show that:

(a) Γ is unsatisfiable if and only if $\sim C_1 \vee \cdots \vee \sim C_m$ is valid for some $C_1, \ldots, C_m$ in Γ.

(b) D is a consequence of Γ if and only if D is a consequence of some finite subset of Γ.

(c) D is a consequence of Γ if and only if $\sim C_1 \vee \cdots \vee \sim C_m \vee D$ is valid for some $C_1, \ldots, C_m$ in Γ.

12.16 For any prime $p = 2, 3, 5, \ldots$, let $D_p(x)$ be the formula $\exists y\ \mathbf{p} \cdot y = x$ of the language of arithmetic, so that for any natural number n, $D_p(\mathbf{n})$ is true if and only if p divides n without remainder. Let S be any set of primes. Say that a nonstandard model $\mathcal{M}$ of arithmetic *encrypts* S if there is an individual m in the domain $|\boldsymbol{\mathcal{M}}|$ such that $\boldsymbol{\mathcal{M}} \models D_p[m]$ for all p belonging to S, and $\boldsymbol{\mathcal{M}} \models \sim D_p[m]$ for all p not belonging to S. Show that for any set S of primes there is a denumerable nonstandard model of arithmetic that encrypts S.

12.17 Show that there are nonenumerably many isomorphism types of denumerable nonstandard models of arithmetic.

12.18 Show that if two sentences have the same enumerable models, then they are logically equivalent.

12.19 Work with a language whose only nonlogical symbol is a single two-place predicate $<$. Consider the set of sentences of this language that are true in the interpretation where the domain is the set of real numbers and the denotation of the predicate is the usual order on real numbers. According to the Löwenheim–Skolem theorem, there must be an *enumerable* model of this set of sentences. Can you guess what one is?

The next several problems provide a significant example of a denumerably categorical set of sentences.

12.20 Work with a language whose only nonlogical symbol is a single two-place predicate $<$. The models of the following sentence *LO* of the language are called *linear orders*:

$$\begin{aligned}&\forall x \sim x < x\ \&\\&\forall x \forall y \forall z((x < y\ \&\ y < z) \rightarrow x < z)\ \&\\&\forall x \forall y(x < y \vee x = y \vee y < x).\end{aligned}$$

Such a model $\mathcal{A}$ will consist of a nonempty set $|\mathcal{A}|$ or A and a two-place relation $<^{\mathcal{A}}$ or $<_A$ on it. Show that the above sentence implies

$$\forall x \forall y \sim (x < y \,\&\, y < x).$$

12.21 Continuing the preceding problem, a *finite partial isomorphism* between linear orders $(A, <_A)$ and $(B, <_B)$ is a function j from a finite subset of A onto a finite subset of B such that for all a_1 and a_2 in the domain of j, $a_1 <_A a_2$ if and only if $j(a_1) <_A j(a_1)$. Show that if j is a finite partial isomorphism from a linear order $(A, <_A)$ to the rational numbers with their usual order, and a is any element of A not in the domain of j, then j can be extended to a finite partial isomorphism whose domain is the domain of j together with a. (Here *extended* means that the new isomorphism assigns the same rational numbers as the old to elements of A there were already in the domain of the old.)

12.22 Continuing the preceding problem, if $j_0, j_1, j_2, \ldots$ are finite partial isomorphisms from an enumerable linear order to the rational numbers with their usual order, and if each j_{i+1} is an extension of the preceding j_i, and if every element of A is in the domain of one of the j_i (and hence of all j_k for $k \geq i$), then $(A, <_A)$ is isomorphic to some *suborder* of the rational numbers with their usual order. (Here *suborder* means a linear order $(B, <_B)$ where B is some subset of the rational numbers, and $<_B$ the usual order on rational numbers as it applies to elements of this subset.)

12.23 Continuing the preceding problem, show that every enumerable linear order $(A, <_A)$ is isomorphic to a suborder of the rational numbers with their usual order.

12.24 Continuing the preceding problem, a linear order is said to be *dense* if it is a model of

$$\forall x \forall y (x < y \rightarrow \exists z (x < z \,\&\, z < y)).$$

It is said to have *no endpoints* if it is a model of

$$\sim \exists x \forall y (x < y \vee x = y) \,\&\, \sim \exists x \forall y (x = y \vee y < x).$$

Which of the following is dense: the natural numbers, the integers, the rational numbers, the real numbers, in each case with their usual order? Which have no endpoints?

12.25 Continuing the preceding problem, show that the set of sentences whose models are the dense linear orders without endpoints is denumerably categorical.

12.26 A linear order is said to have endpoints if it is a model of

$$\exists x \forall y (x < y \vee x = y) \,\&\, \exists x \forall y (x = y \vee y < x).$$

Show that the set of sentences whose models are the dense linear orders with endpoints is denumerably categorical.

12.27 How many isomorphism types of denumerable dense linear orders are there?

13

The Existence of Models

This chapter is entirely devoted to the proof of the compactness theorem. Section 13.1 outlines the proof, which reduces to establishing two main lemmas. These are then taken up in sections 13.2 through 13.4 to complete the proof, from which the Löwenheim–Skolem theorem also emerges as a corollary. The optional section 13.5 discusses what happens if nonenumerable languages are admitted: compactness still holds, but the Löwenheim–Skolem theorem in its usual 'downward' form fails, while an alternative 'upward' theorem holds.

13.1 Outline of the Proof

Our goal is to prove the compactness theorem, which has already been stated in the preceding chapter (in section 12.3). For convenience, we work with a version of first-order logic in which the only logical operators are $\sim$, $\vee$, and $\exists$, that is, in which & and $\forall$ are treated as unofficial abbreviations. The hypothesis of the theorem, it will be recalled, is that every finite subset of a given set of sentences is satisfiable, and the conclusion we want to prove is that the set itself is satisfiable, or, as we more elaborately put it, belongs to the set S of all satisfiable sets of sentences. As a first step towards the proof, we set down some properties enjoyed by this target set S. The reason for not including & and $\forall$ officially in the language is simply that in this and subsequent lemmas we would need four more clauses, two for & and two for $\forall$. These would not be difficult to prove, but they would be tedious.

13.1 Lemma (Satisfaction properties lemma). Let S be the set of all sets Γ of sentences of a given language such that Γ is satisfiable. Then S has the following properties:

(S0) If Γ is in S and Γ_0 is a subset of Γ, then Γ_0 is in S.
(S1) If Γ is in S, then for no sentence A are both A and $\sim A$ in Γ.
(S2) If Γ is in S and $\sim\sim B$ is in Γ, then $\Gamma \cup \{B\}$ is in S.
(S3) If Γ is in S and $(B \vee C)$ is in Γ, then either $\Gamma \cup \{B\}$ is in S or $\Gamma \cup \{C\}$ is in S.
(S4) If Γ is in S and $\sim(B \vee C)$ is in Γ, then $\Gamma \cup \{\sim B\}$ is in S and $\Gamma \cup \{\sim C\}$ is in S.
(S5) If Γ is in S and $\{\exists x\, B(x)\}$ is in Γ, and the constant c does not occur in Γ or $\exists x B(x)$, then $\Gamma \cup \{B(c)\}$ is in S.
(S6) If Γ is in S and $\sim\exists x\, B(x)$ is in Γ, then for every closed term t, $\Gamma \cup \{\sim B(t)\}$ is in S.

(S7) If Γ is in S, then $\Gamma \cup \{t = t\}$ is in S for any closed term t of the language of Γ.
(S8) If Γ is in S and $B(s)$ and $s = t$ are in Γ, then $\Gamma \cup \{B(t)\}$ is in S.

Proof: These have been established in Chapter 10. (S0) and (S1) were mentioned just before Example 10.4. (S2) appeared as Example 10.4(g), where it was derived from Example10.3(a). (S4), (S6), and (S8) can be derived in exactly the same way from Example 10.3(c), 10.3(e), and 10.3(f), as remarked after the proof of Example 10.4. (S3), (S5), and (S7) were established in Example 10.5.

We call (S0)–(S8) the *satisfaction properties*. Of course, at the outset we do not know that the set we are interested in belongs to S. Rather, what we are given is that it belongs to the set S^* of all sets of sentences whose every finite subset belongs to S. (Of course, once we succeed in proving the compactness theorem, S and S^* will turn out to be the *same* set.) It will be useful to note that S^* shares the above properties of S.

13.2 Lemma (Finite character lemma). If S is a set of sets of sentences having the satisfaction properties, then the set S^* of all sets of formulas whose every finite subset is in S also has properties (S0)–(S8).

Proof: To prove (S0) for S^*, note that if every finite subset of Γ is in S, and Γ_0 is subset of Γ, then every finite subset of Γ_0 is in S, since any finite subset of Γ_0 is a finite subset of Γ. To prove (S1) for S^*, note that if every finite subset of Γ is in S, then Γ cannot contain both A and $\sim A$, else $\{A, \sim A\}$ would be a finite subset of Γ, though $\{A, \sim A\}$ is not in S by property (S1) of S. To prove (S2) for S^*, note that if every finite subset of $\Gamma \cup \{\sim\sim B\}$ is in S, then any finite subset of $\Gamma \cup \{B\}$ is either a finite subset of Γ and hence of $\Gamma \cup \{\sim\sim B\}$ and therefore is in S, or else is of form $\Gamma_0 \cup \{B\}$ where Γ_0 is a finite subset of Γ. In the latter case, $\Gamma_0 \cup \{\sim\sim B\}$ is a finite subset of $\Gamma \cup \{\sim\sim B\}$ and therefore in S, so $\Gamma \cup \{B\}$ is in S by property (S2) of S. Thus the finite subset $\Gamma_0 \cup \{B\}$ is in S^*. (S4)–(S8) for S^* follow from (S4)–(S8) for S *exactly* as in the case of (S2). It remains only to prove (S3) for S^*.

So suppose every finite subset of $\Gamma \cup \{(B \vee C)\}$ is in S, but that it is not the case that every finite subset of $\Gamma \cup \{B\}$ is in S, or in other words that there is some finite subset of $\Gamma \cup \{B\}$ that is not in S. This cannot just be a subset of Γ, since then it would be a finite subset of $\Gamma \cup \{(B \vee C)\}$ and would be in S. So it must be of the form $\Gamma_0 \cup \{B\}$ for some finite subset Γ_0 of Γ. We now claim that every finite subset of $\Gamma \cup \{C\}$ is in S. For any such set is either a finite subset of Γ and therefore in S, or is of form $\Gamma_1 \cup \{C\}$ for some finite subset Γ_1 of Γ. In the latter case, $\Gamma_0 \cup \Gamma_1 \cup \{(B \vee C)\}$ is a finite subset of $\Gamma \cup \{(B \vee C)\}$ and so is in S. It follows that either $\Gamma_0 \cup \Gamma_1 \cup \{B\}$ or $\Gamma_0 \cup \Gamma_1 \cup \{C\}$ is in S by property (S3) of S. But if $\Gamma_0 \cup \Gamma_1 \cup \{B\}$ were in S, then by property (S1) of S, $\Gamma_0 \cup \{B\}$ would be in S, which it is not. So it must be that $\Gamma_0 \cup \Gamma_1 \cup \{C\}$ is in S and hence $\Gamma_1 \cup \{C\}$ is in S by property (S0) of S.

By these preliminary manoeuvres, we have reduced proving the compactness theorem to proving the following lemma, which is a kind of converse to Lemma 13.1. In stating it we suppose we have available an infinite set of constants not occurring in the set of sentences we are interested in.

13.3 Lemma (Model existence lemma). Let L be a language, and L^+ a language obtained by adding infinitely many new constants to L. If S^* is a set of sets of sentences of L^+ having the satisfaction properties, then every set of sentences of L in S^* has a model in which each element of the domain is the denotation of some closed term of L^+.

Note that the condition that every element of the domain is the denotation of some closed term guarantees that, since we are working in an enumerable language, the domain will be enumerable, which means that we get not only the compactness but also the Löwenheim–Skolem theorem, as remarked in the preceding chapter (following the statement of the two theorems in section 12.3).

So it 'only' remains to prove Lemma 13.3. The conclusion of Lemma 13.3 asserts the existence of an interpretation in which every element of the domain is the denotation of some closed term of the relevant language, and we begin by listing some properties that the set of all sentences true in such an interpretation would have to have.

13.4 Proposition (Closure properties lemma). Let L^+ be a language and $\mathcal{M}$ an interpretation thereof in which every element of the domain is the denotation of some closed term. Then the set Γ^* of sentences true in $\mathcal{M}$ has the following properties:

(C1) For no sentence A are both A and $\sim A$ in Γ^*.
(C2) If $\sim\sim B$ is in Γ^*, then B is in Γ^*.
(C3) If $B \vee C$ is in Γ^*, then either B is in Γ^* or C is in Γ^*.
(C4) If $\sim(B \vee C)$ is in Γ^*, then both $\sim B$ and $\sim C$ are in Γ^*.
(C5) If $\exists x B(x)$ is in Γ^*, then for some closed term t of L^+, $B(t)$ is in Γ^*.
(C6) If $\sim\exists x B(x)$ is in Γ^*, then for every closed term t of L^+, $\sim B(t)$ is in Γ^*.
(C7) For every closed term t of L^+, $t = t$ is in Γ^*.
(C8) If $B(s)$ and $s = t$ are in Γ^*, then $B(t)$ is in Γ^*.

Proof: For (C1), for no A are both A and $\sim A$ true in the same interpretation. For (C2), anything implied by anything true in a given interpretation is itself true in that interpretation, and B is implied by $\sim\sim B$. Similarly for (C4) and (C6)–(C8).

For (C3), any interpretation that makes a disjunct true must make at least one of its disjuncts true.

For (C5), if $\exists x B(x)$ is true in a given interpretation, then $B(x)$ is satisfied by some element m of the domain, and if that element m is the denotation of some closed term t, then $B(t)$ is true.

We call the properties (C1)–(C8) the *closure properties*. Actually, it is not Proposition 13.4 itself that will be useful to us here, but the following converse.

13.5 Lemma (Term models lemma). Let Γ^* be a set of sentences with the closure properties. Then there is an interpretation $\mathcal{M}$ in which every element of the domain is the denotation of some closed term, such that every sentence in Γ^* is true in $\mathcal{M}$.

To prove Lemma 13.3, it would suffice to prove the foregoing lemma plus the following one.

13.6 Lemma (Closure lemma). Let L be a language, and L^+ a language obtained by adding infinitely many new constants to L. If S^* is a set of sets of sentences of L^+ having the satisfaction properties, then every set Γ of sentences of L in S^* can be extended to a set Γ^* of sentences of L^+ having the closure properties.

Sections 13.2 and 13.3 will be devoted to the proof of the term models lemma, Lemma 13.5. As in so many other proofs, we consider first, in section 13.2, the case where identity and function symbols are absent, so that (C7) and (C8) may be ignored, and the only closed terms are constants, and then, in section 13.3, consider the additional complications that arise when identity is present, as well as those created by the presence of function symbols. The proof of the closure lemma, Lemma 13.6, will be given in section 13.4, with an alternative proof, avoiding any dependence on the assumption that the language is enumerable, to be outlined in the optional section 13.5.

13.2 The First Stage of the Proof

In this section we are going to prove the term models lemma, Lemma 13.5, in the case where identity and function symbols are absent. So let there be given a set Γ^* with the closure properties (C1)–(C6), as in the hypothesis of the lemma to be proved. We want to show that, as in the conclusion of that lemma, there is an interpretation in which every element of the domain is the denotation of some constant of the language of Γ^*, in which every sentence in Γ^* will be true.

To specify an interpretation $\mathcal{M}$ in this case, we need to do a number of things. To begin with, we must specify the domain $|\mathcal{M}|$. Also, we must specify for each constant c of the language which element $c^{\mathcal{M}}$ of the domain is to serve as its denotation. Moreover, we must do all this in such a way that *every* element of the domain is the denotation of *some* constant. This much is easily accomplished: simply pick for each constant c some object $c^{\mathcal{M}}$, picking a distinct object for each distinct constant, and let the domain consist of these objects.

To complete the specification of the interpretation, we must specify for each predicate R of the language what relation $R^{\mathcal{M}}$ on elements of the domain is to serve as its denotation. Moreover, we must do so in such a way that it will turn out that for every sentence B in the language we have

$$\text{(1)} \qquad \text{if} \quad B \text{ is in } \Gamma^* \quad \text{then} \quad \mathcal{M} \models B.$$

What we do is to specify $R^{\mathcal{M}}$ in such a way that (1) *automatically* becomes true for *atomic* B. We define $R^{\mathcal{M}}$ by the following condition:

$$R^{\mathcal{M}}\big(c_1^{\mathcal{M}}, \ldots, c_n^{\mathcal{M}}\big) \quad \text{if and only if} \quad R(c_1, \ldots, c_n) \text{ is in } \Gamma^*.$$

Now the definition of truth for atomic sentences reads as follows:

$$\mathcal{M} \models R(c_1, \ldots, c_n) \quad \text{if and only if} \quad R^{\mathcal{M}}\big(c_1^{\mathcal{M}}, \ldots, c_n^{\mathcal{M}}\big).$$

We therefore have the following:

$$\text{(2)} \qquad \mathcal{M} \models R(c_1, \ldots, c_n) \quad \text{if and only if} \quad R(c_1, \ldots, c_n) \text{ is in } \Gamma^*$$

and this implies (1) for atomic B.

We also have (1) for negated atomic sentences. For if $\sim R(c_1, \ldots, c_n)$ is in Γ^*, then by property (C1) of Γ^*, $R(c_1, \ldots, c_n)$ is *not* in Γ^*, and therefore by (2), $R(c_1, \ldots, c_n)$ is *not* true in $\mathcal{M}$, and so $\sim R(c_1, \ldots, c_n)$ *is* true in $\mathcal{M}$, as required.

To prove (1) for other formulas, we proceed by induction on complexity. There are three cases, according as A is a negation, a disjunction, or an existential quantification. However, we divide the negation case into subcases. Apart from the subcase of the negation of an atomic sentence, which we have already handled, there are three of these: the negation of a negation, the negation of a disjunction, and the negation of an existential quantification. So there are five cases in all:

to prove (1) for $\sim\sim B$	assuming (1) for B
to prove (1) for $B_1 \vee B_2$	assuming (1) for each B_i
to prove (1) for $\sim(B_1 \vee B_2)$	assuming (1) for each $\sim B_i$
to prove (1) for $\exists x B(x)$	assuming (1) for each $B(c)$
to prove (1) for $\sim\exists x B(x)$	assuming (1) for each $\sim B(c)$.

The five cases correspond to the five properties (C2)–(C6), which are just what is needed to prove them.

If $\sim\sim B$ is in Γ^*, then B is in Γ^* by property (C2). Assuming that (1) holds for B, it follows that B is true in $\mathcal{M}$. But then $\sim B$ is untrue, and $\sim\sim B$ is true as required. If $B_1 \vee B_2$ is in Γ^*, then B_i is in Γ^* for at least one of $i = 1$ or 2 by property (C3) of Γ^*. Assuming (1) holds for this B_i, it follows that B_i is true in $\mathcal{M}$. But then $B_1 \vee B_2$ is true as required. If $\sim(B_1 \vee B_2)$ is in Γ^*, then each $\sim B_i$ is in Γ^* for $i = 1$ or 2 by property (C4) of Γ^*. Assuming (1) holds for the $\sim B_i$, it follows that each $\sim B_i$ is true in $\mathcal{M}$. But then each B_i is untrue, so $B_1 \vee B_2$ is untrue, so $\sim(B_1 \vee B_2)$ is true as required.

In connection with existential quantification, note that since every individual in the domain is the denotation of some constant, $\exists x B(x)$ will be true if and only if $B(c)$ is true for some constant c. If $\exists x B(x)$ is in Γ^*, then $B(c)$ is in Γ^* for some constant c by property (C5) of Γ^*. Assuming (1) holds for this $B(c)$, it follows that $B(c)$ is true in $\mathcal{M}$. But then $\exists x B(x)$ is true as required. If $\sim\exists x B(x)$ is in Γ^*, then $\sim B(c)$ is in Γ^* for every constant c by property (C6) of Γ^*. Assuming (1) holds for each $\sim B(c)$, it follows that $\sim B(c)$ is true in $\mathcal{M}$. But then $B(c)$ is untrue for each c, and so $\exists x B(x)$ is untrue, and $\sim\exists x B(x)$ is true as required. We are done with the case without identity or function symbols.

13.3 The Second Stage of the Proof

In this section we want to extend the result of the preceding section to the case where identity is present, and then to the case where function symbols are also present. Before describing the modifications of the construction of the preceding section needed to accomplish this, we pause for a lemma.

13.7 Lemma. Let Γ^* be a set of sentences with properties (C1)–(C8). For closed terms t and s write $t \equiv s$ to mean that the sentence $t = s$ is in Γ^*. Then the following hold:

(E1) $t \equiv t$.

(E2) If $s \equiv t$, then $t \equiv s$.

(E3) If $t \equiv s$ and $s \equiv r$, then $t \equiv r$.

(E4) If $t_1 \equiv s_1, \ldots, t_n \equiv s_n$, then for any predicate R, $R(t_1, \ldots, t_n)$ is in Γ^* if and only if $R(s_1, \ldots, s_n)$ is in Γ^*.

(E5) If $t_1 \equiv s_1, \ldots, t_n \equiv s_n$, then for any function symbol f, $f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)$ is in Γ^*.

Proof: (E1) is simply a restatement of (C7). For (E2), let $B(x)$ be the formula $x = s$. We now know that the sentence $B(s)$, which is to say the sentence $s = s$, is in Γ^*, so if $s = t$ is in Γ^*, it follows by (C8) that the sentence $B(t)$, which is to say the sentence $t = s$, is in Γ^*. For (E3), let $B(x)$ be the formula $x = r$. If $t = s$ is in Γ^*, then we now know $s = t$ is in Γ^*, and if $B(s)$, which is $s = r$, is in Γ^*, it follows by (C8) that $B(t)$, which is $t = r$, is in Γ^*. For (E4), if all $t_i = s_i$ are in Γ^* and $R(s_1, \ldots, s_n)$ is in Γ^*, then repeated application of (C8) tells us that $R(t_1, s_2, s_3, \ldots, s_n)$ is in Γ^*, that $R(t_1, t_2, s_3, \ldots, s_n)$ is in Γ^*, and so on, and finally that $R(t_1, \ldots, t_n)$ is in Γ^*. This gives the 'only if' direction of (E4). For the 'if' direction, if all $t_i = s_i$ are in Γ^*, then so are all $s_i = t_i$, so if $R(t_1, \ldots, t_n)$ is in Γ^*, then by the direction we have already proved, $R(s_1, \ldots, s_n)$ is in Γ^*. For (E5), the proof just given for (E4) applies not only to atomic formulas $R(x_1, \ldots, x_n)$ but to arbitrary formulas $F(x_1, \ldots, x_n)$. Applying this fact where F is the formula $f(t_1, \ldots, t_n) = f(x_1, \ldots, x_n)$ gives (E5).

Note that (E1)–(E3) say that $\equiv$ is an equivalence relation. If we write $[t]$ for the equivalence class of t, then (E4) and (E5) may be rewritten as follows:

(E4′) If $[t_1] = [s_1], \ldots, [t_n] = [s_n]$, then for any predicate R, $R(t_1, \ldots, t_n)$ is in Γ^* if and only if $R(s_1, \ldots, s_n)$ is in Γ^*

(E5′) If $[t_1] = [s_1], \ldots, [t_n] = [s_n]$, then for any function symbol f, $[f(t_1, \ldots, t_n)] = [f(s_1, \ldots, s_n)]$.

We now return to the proof of the term models lemma, taking up the case where identity is present but function symbols are absent, so the only closed terms are constants. To specify the domain for our interpretation, instead of picking a distinct object for each distinct constant, we pick a distinct object C^* for each distinct *equivalence class C* of constants. We let the domain of the interpretation consist of these objects, and for the denotations of constants we specify the following:

$$c^{\mathcal{M}} = [c]^*. \tag{3}$$

Since $[c] = [d]$ if and only if $c = d$ is in Γ^*, we then have:

$$c^{\mathcal{M}} = d^{\mathcal{M}} \quad \text{if and only if} \quad c = d \text{ is in } \Gamma^*.$$

This is (the analogue of) (2) of the preceding section for atomic sentences involving the logical predicate $=$, and gives us (1) of the preceding section for such sentences and their negations.

What remains to be done is to define the denotation $R^{\mathcal{M}}$ for a nonlogical predicate R, in such a way that (2) of the preceding section will hold for atomic sentences involving nonlogical predicates. *From that point, the rest of the proof will be exactly the same* as where identity was not present. Towards framing the definition of $R^{\mathcal{M}}$, note that (E4′) allows us to give the following definition:

$$R^{\mathcal{M}}(C_1{*}, \ldots, C_n{*}) \quad \text{if and only if} \quad \begin{array}{l} R(c_1, \ldots, c_n) \text{ is in } \Gamma{*} \\ \text{for } \textit{some} \text{ or equivalently } \textit{any} \\ c_i \text{ with } C_i = [c_i]. \end{array}$$

Thus

$$R^{\mathcal{M}}([c_1], \ldots, [c_n]) \quad \text{if and only if} \quad R(c_1, \ldots, c_n) \text{ is in } \Gamma{*}.$$

Together with (3), this gives (2) of the preceding section. Since as already indicated the proof is the same from this point on, we are done with the case with identity but without function symbols.

For the case with function symbols, we pick a distinct object $T{*}$ for each equivalence class of *closed terms*, and let the domain of the interpretation consist of these objects. Note that (3) above still holds for constants. We must now specify for each function symbol f what function $f^{\mathcal{M}}$ on this domain is to serve as its denotation, and in such a way that (3) will hold for all closed terms. From that point, the rest of the proof will be exactly the same as in the preceding case where function symbols were not present.

(E5′) allows us to give the following definition:

$$f^{\mathcal{M}}(T_1{*}, \ldots, T_n{*}) = T{*} \qquad \text{where} \quad \begin{array}{l} T = [f(t_1, \ldots, t_n)] \\ \text{for } \textit{some} \text{ or equivalently } \textit{any} \\ t_i \text{ with } T_i = [t_i]. \end{array}$$

Thus

(4) $$f^{\mathcal{M}}([t_1]{*}, \ldots, [t_n]{*}) = [f(t_1, \ldots, t_n)]{*}.$$

We can now prove by induction on complexity that (3) above, which holds by definition for constants, in fact holds for any closed term t. For suppose (3) holds for $t_1, \ldots, t_n$, and consider $f(t_1, \ldots, t_n)$. By the general definition of the denotation of a term we have

$$(f(t_1, \ldots, t_n))^{\mathcal{M}} = f^{\mathcal{M}}\big(t_1^{\mathcal{M}}, \ldots, t_n^{\mathcal{M}}\big).$$

By our induction hypothesis about the t_i we have

$$t_i^{\mathcal{M}} = [t_i]{*}.$$

Putting these together, we get

$$(f(t_1, \ldots, t_n))^{\mathcal{M}} = f^{\mathcal{M}}([t_1]{*}, \ldots, [t_n]{*}).$$

And this together with the definition (4) above gives

$$(f(t_1, \ldots, t_n))^{\mathcal{M}} = [f(t_1, \ldots, t_n)]{*}.$$

which is precisely (3) above for the closed term $f(t_1, \ldots, t_n)$. Since, as already indicated, the proof is the same from this point on, we are done.

13.4 The Third Stage of the Proof

What remains to be proved is the closure lemma, Lemma 13.6. So let there be given a language L, a language L^+ obtained by adding infinitely many new constants to L, a set S^* of sets of sentences of L^+ having the satisfaction properties (S0)–(S8), and a set Γ of sentences of L in S^*, as in the hypotheses of the lemma to be proved. We want to show that, as in the conclusion of that lemma, Γ can be extended to a set Γ^* of sentences of L^+ with closure properties (C1)–(C8).

The idea of the proof will be to obtain Γ^* as the union of a sequence of sets $\Gamma_0, \Gamma_1, \Gamma_2, \ldots$, where each Γ_n belongs to S^* and each contains all earlier sets Γ_m for $m < n$, and where Γ_0 is just Γ. (C1) will easily follow, because if A and $\sim A$ were both in Γ^*, A would be in some Γ_m and $\sim A$ would be in some Γ_n, and then both would be in Γ_k, where k is whichever of m and n is the larger. But since Γ_k is in S^*, this is impossible, since (S0) says precisely that no element of S^* contains both A and $\sim A$ for any A.

What need to be worried about are (C2)–(C8). We have said that each Γ_{k+1} will be a set in S^* containing Γ_k. In fact, each Γ_{k+1} be obtained by adding to Γ_k a *single* sentence B_k, so that $\Gamma_{k+1} = \Gamma_k \cup \{B_k\}$. (It follows that each Γ_n will be obtained by adding only finitely many sentences to Γ, and therefore will involve only finitely many of the constants of L^+ that are not in the language L of Γ, leaving at each stage infinitely many as yet unused constants.) At each stage, having Γ_k in S^*, we are free to choose as B_k any sentence such that $\Gamma_k \cup \{B_k\}$ is still in S^*. But we must make the choices in such a way that in the end (C2)–(C8) hold.

Now how can we arrange that Γ^* fulfills condition (C2), for example? Well, if $\sim\sim B$ is in Γ^*, it is in some Γ_m. *If* we can so arrange matters that whenever m and B are such that $\sim\sim B$ is in Γ_m, then B is in Γ_{k+1} for some $k \geq m$, *then* it will follow that B is in Γ^*, as required by (C2). To achieve this, it will be more than enough if we can so arrange matters that the following holds:

$$\text{If} \quad \sim\sim B \text{ is in } \Gamma_m, \quad \text{then} \quad \text{for some } k \geq m, \quad \Gamma_{k+1} = \Gamma_k \cup \{B\}.$$

But *can* we so arrange matters that this holds? Well, what does (S2) tell us? If $\sim\sim B$ is in Γ_m, then $\sim\sim B$ will still be in Γ_k for any $k \geq m$, since the sets get larger. Since each Γ_k is to be in S^*, (S2) promises that $\Gamma_k \cup \{B\}$ will be in S^*. That is:

$$\text{If} \quad \sim\sim B \text{ is in } \Gamma_m, \quad \text{then} \quad \text{for any } k \geq m, \quad \Gamma_k \cup \{B\} \text{ is in } S^*.$$

So we *could* take $\Gamma_{k+1} = \Gamma_k \cup \{B\}$ if we chose to do so.

To understand better what is going on here, let us introduce some suggestive terminology. If $\sim\sim B$ is in Γ_m, let us say that *the demand for admission of B is raised* at stage m; and if $\Gamma_{k+1} = \Gamma_k \cup \{B\}$, let us say that *the demand is granted* at stage k. What is required by (C2) is that *any* demand that is raised at *any* stage m should be granted at some later stage k. And what is promised by (S2) is that at any stage k, any *one* demand raised at any *one* earlier stage m could be granted. There is a gap here between what is demanded and what is promised, since it may well be that there are infinitely many demands raised at stage m, which is to say, infinitely many sentences of form $\sim\sim B$ in Γ_m, and in any case, there are infinitely many stages m at which new demands may arise—and all this only considering demands of the type associated with condition (C2), whereas there are several other conditions, also raising demands, that we also wish to fulfill.

Let us look at these. The relationship between (C3)–(C8) and (S3)–(S8) is exactly the same as between (C2) and (S2). Each of (C2)–(C8) corresponds to a demand of a certain type:

(C2) If $\sim\sim B$ is in Γ_m, then for some $k \geq m$, $\Gamma_{k+1} = \Gamma_k \cup \{B\}$.
(C3) If $B \vee C$ is in Γ_m, then for some $k \geq m$, $\Gamma_{k+1} = \Gamma_k \cup \{B\}$ or $\Gamma_k \cup \{C\}$.
(C4) If $\sim(B \vee C)$ or $\sim(C \vee B)$ is in Γ_m, then for some $k \geq m$, $\Gamma_{k+1} = \Gamma_k \cup \{\sim B\}$.
(C5) If $\exists x B(x)$ is in Γ_m, then for some $k \geq m$, for some constant c, $\Gamma_{k+1} = \Gamma_k \cup \{B(c)\}$.
(C6) If $\sim\exists x B(x)$ is in Γ_m and t is a closed term in the language of Γ_m, then for some $k \geq m$, $\Gamma_{k+1} = \Gamma_k \cup \{\sim B(t)\}$.
(C7) If t is a closed term in the language of Γ_m, then for some $k \geq m$, $\Gamma_{k+1} = \Gamma_k \cup \{t = t\}$.
(C8) If $B(s)$ and $s = t$ are in Γ_m, where s and t are closed terms $B(x)$ a formula, then for some $k \geq m$, $\Gamma_{k+1} = \Gamma_k \cup \{B(t)\}$.

Each of (S2)–(S8) promises that any one demand of the relevant type can be granted:

(S2) If $\sim\sim B$ is in Γ_m, then for any $k \geq m$, $\Gamma_k \cup \{B\}$ is in S^*.
(S3) If $B \vee C$ is in Γ_m, then for any $k \geq m$, $\Gamma_k \cup \{B\}$ or $\Gamma_k \cup \{C\}$ is in S^*.
(S4) If $\sim(B \vee C)$ or $\sim(C \vee B)$ is in Γ_m, then for any $k \geq m$, $\Gamma_k \cup \{\sim B\}$ is in S^*.
(S5) If $\exists x B(x)$ is in Γ_m, then for any $k \geq m$, for any as yet unused constant c, $\Gamma_k \cup \{B(c)\}$ is in S^*.
(S6) If $\sim\exists x B(x)$ is in Γ_m and t is a closed term in the language of Γ_m, then for any $k \geq m$, $\Gamma_k \cup \{\sim B(t)\}$ is in S^*.
(S7) If t is a closed term in the language of Γ_m, then for any $k \geq m$, $\Gamma_k \cup \{t = t\}$ is in S^*.
(S8) If $B(s)$ and $s = t$ are in Γ_m, where s and t are closed terms $B(x)$ a formula, then for any $k \geq m$, $\Gamma_k \cup \{B(t)\}$ is in S^*.

At any stage k of the construction, we can grant *any one demand we choose* from among those that have been raised at earlier stages, but for the construction to succeed, we must make our successive choices so that in the end *any demand that is ever raised at any stage* is granted at some later stage. Our difficulty is that at each stage many different demands may be raised. Our situation is like that of Herakles fighting the hydra: every time we chop off one head (grant one demand), multiple new heads appear (multiple new demands are raised). At least in one respect, however, we have made progress: we have succeeded in redescribing our problem in abstract terms, eliminating all details about which particular formulas are of concern.

And indeed, with this redescription of the problem we are now not far from a solution. We need only recall two facts. First, our languages are *enumerable*, so that at each stage, though an infinity of demands may be raised, it is still only an *enumerable* infinity. Each demand may be worded 'admit such-and-such a sentence' (or 'admit one or the other of two such-and-such sentences'), and an enumeration of the sentences of our language therefore gives rise to an enumeration of all the demands raised at any given stage. Thus each demand that is ever raised may be described as the ith demand raised at stage m, for some numbers i and m, and so may be described by a pair of numbers (i, m). Second, we have seen in Chapter 1 that there is a way—in fact, there are many ways—of coding any pair of numbers by a single number $j(i, m)$, and if one looks closely at this coding, one easily sees that $j(i, m)$ is greater than m (and greater than i). We can solve our problem, then, by proceeding as follows. At stage k, see what pair (i, m) is coded by k, and grant the ith demand that was raised at stage $m < k$. In this way, though we grant only one demand at a time, all the demands that are ever raised will eventually be granted.

The proof of the compactness theorem is now complete.

13.5* Nonenumerable Languages

In Chapter 12 we mentioned in passing the possibility of allowing nonenumerable languages. The Löwenheim–Skolem theorem would then fail.

13.8 Example (The failure of the downward Löwenheim–Skolem theorem for a nonenumerable language). Take one constant c_ξ for each real number ξ, and let Γ be the set of all sentences $c_\xi \neq c_\eta$ for $\xi \neq \eta$. Clearly Γ has a model with domain the real numbers, in which c_ξ denotes ξ. Equally clearly, any model of Γ will be nondenumerable.

However, it can be shown that the compactness theorem still holds. The proof we have given does not work for a nonenumerable language: no essential use of the enumerability of the language was made in the proof of the term models lemma, but the proof given in the preceding section for the closure lemma did make heavy use at the end of the enumerability assumption. In this section we outline a different proof of the closure lemma, which can be generalized to cover nonenumerable languages, and note one consequence of the generalized version of the compactness theorem. Many verifications are relegated to the problems.

It is not hard to show that if Γ is a satisfiable set of sentences, $\exists x F(x)$ a sentence of the language of Γ, and c a constant *not* in the language of Γ, then $\Gamma \cup \{\exists x F(x) \rightarrow F(c)\}$ is satisfiable [imitating the proof of Example 10.5(*b*), which gave us (S5) in Lemma 13.1]. Now let L be a language. Let $L_0 = L$, and given L_n, let L_{n+1} be the result of adding to L_n a new constant c_F for each formula $\exists x F(x)$ of L_n. Let L^+ be the union of all the L_n. The set of *Henkin axioms* is the set H of all sentences $\exists x F(x) \rightarrow F(c_F)$ of L^+. It is not hard to show that if Γ is a set of sentences of L and every finite subset of Γ has a model then every finite subset of $\Gamma \cup H$ has a model (using the observation with which we began this paragraph). Let S^* be the set of all sets of sentences Γ of L^+ such that every finite subset of $\Gamma \cup H$ has a model. What we have just remarked is that if Γ is a set of sentences of L and every finite subset of Γ has a model, then Γ is in S^*. It is not hard to show that S^* has the satisfiability properties (S1)–(S4) and (S6)–(S8) (imitating the proof of Lemma 13.2).

We now introduce some set-theoretic terminology. Let I be a nonempty set. A family P of subsets of I is said to be of *finite character* provided that for each subset Γ of I, Γ is in P if and only if each finite subset of Γ is in P. A subset Γ^* of I is said to be *maximal* with respect to P if Γ^* is in P, but no subset Δ of I properly including Γ^* is in P.

To apply this terminology to the situation we are considering, it is not hard to show that S^* is of finite character (essentially by definition). Nor is it hard to show that any maximal element Γ^* of S^* will contain H (by showing that adding a Henkin axiom to a given set in S^* produces a set still in S^*, so that if the given set was maximal, the Henkin axiom must already have belonged to it). Nor is it hard to show that any maximal element Γ^* of S^* will have closure properties (C1)–(C4) and (C6)–(C8) [since, for instance, if $\sim\sim B$ is in Γ^*, then adding B to Γ^* produces a set still in S^* by (S2)]. Nor, for that matter, is it hard to show that such a Γ^* will also have closure property (C5) [using the fact that whether or not $\exists x F(x)$ is in Γ^*, Γ^* contains the Henkin axioms $\sim\exists x F(x) \vee F(c_F)$, and applying (C1) and (C3)]. Thus by Lemma 13.5, whose proof made no essential use of enumerability, Γ^* will have a model.

Putting everything together from the preceding several paragraphs, if Γ is a set of sentences of L such that every finite subset of Γ has a model, then Γ itself will have a model, *provided* we can prove that for every set Γ in S^* there is a maximal element Γ^* in S^* that contains Γ.

And this does follow using a general set-theoretic fact, the *maximal principle*, according to which for any nonempty set I and any set P of subsets of I that has finite character, and any Γ in P, there is a maximal element Γ^* of P that contains Γ. It is not hard to prove this principle in the case where I is enumerable (by enumerating its elements $i_0, i_1, i_2, \ldots$, and building Γ^* as the union of sets Γ_n in P, where $\Gamma_0 = \Gamma$, and $\Gamma_{n+1} = \Gamma_n \cup \{i_n\}$ if $\Gamma_n \cup \{i_n\}$ is in P, and $= \Gamma_n$ otherwise). In fact, the maximal principle is known to hold even for nonenumerable I, though the proof in this case requires a formerly controversial axiom of set theory, the *axiom of choice*—indeed, given the other, less controversial axioms of set theory, the maximal principle is *equivalent* to the axiom of choice, a fact whose proof is given in any textbook on set theory, but will not be given here.

Reviewing our work, one sees that using the maximal principle for a nonenumerable set, we get a proof of the compactness theorem for nonenumerable languages. This general version of the compactness theorem has one notable consequence.

13.9 Theorem (The upward Löwenheim–Skolem theorem). Any set of sentences that has an infinite model has a nonenumerable model.

The proof is not hard (combining the ideas of Corollary 12.16 and Example 13.8). But like the proofs of several of our assertions above, we relegate this one to the problems.

Problems

The first several problems pertain to the optional section 13.5.

13.1 Prove the maximal principle for the case where I is enumerable.

13.2 Show that if Γ is a satisfiable set of sentences, $\exists x F(x)$ a sentence of the language of Γ, and c a constant *not* in the language of Γ, then $\Gamma \cup \{\exists x F(x) \rightarrow F(c)\}$ is satisfiable.

13.3 Let L be a language, and construct the language L^+ and the set H of Henkin axioms as in section 13.5. Let S^* be the set of all sets of sentences Γ of L^+ such that every finite subset of $\Gamma \cup H$ has a model. Show that:

(a) Any set Γ of sentences of L whose every finite subset is satisfiable is in S^*.

(b) S^* has satisfiability properties (S1)–(S4) and (S6)–(S8).

13.4 Continuing the notation of the preceding problem, show that:

(a) S^* is of finite character.

(b) Any maximal set Γ^* in S^* contains H.

13.5 Continuing the notation of the preceding problem, let Γ^* be a maximal set in S^*. Show that Γ^* has closure properties (C1)–(C4) and (C6)–(C8).

13.6 Continuing the notation of the preceding problem, let Γ^* be a set of sentences of L^+ containing H and having closure properties (C1)–(C4) and (C6)–(C8). Show that Γ^* also has property (C5).

13.7 Use the compactness theorem for nonenumerable languages to prove the upward Löwenheim–Skolem theorem, Theorem 13.9.

In the remaining problems, for simplicity assume that function symbols are absent, though the results indicated extend to the case where they are present.

13.8 An *embedding* of one interpretation $\mathcal{P}$ in another interpretation $\mathcal{Q}$ is a function j fulfilling all the conditions in the definition of isomorphism in section 13.1, except that j need not be onto. Given an interpretation $\mathcal{P}$, let $L^{\mathcal{P}}$ be the result of adding to the language a constant c_p for each element p of the domain $|\mathcal{P}|$, and and let $\mathcal{P}^*$ be the extension of $\mathcal{P}$ to an interpretation of $L^{\mathcal{P}}$ in which each c_p denotes the corresponding p. The set $\Delta(\mathcal{P})$ of all atomic and negated atomic sentences of $L^{\mathcal{P}}$, whether involving a nonlogical predicate R or the logical predicate $=$, that are true in $\mathcal{P}^*$, is called the *diagram* of $\mathcal{P}$. Show that if $\mathcal{Q}$ is any interpretation of the language of $\mathcal{P}$ that can be extended to a model $\mathcal{Q}^*$ of $\Delta(\mathcal{P})$, then there is an embedding of $\mathcal{P}$ into $\mathcal{Q}$.

13.9 A sentence is called *existential* if and only if it is of the form $\exists x_1 \ldots \exists x_n F$ where F contains no further quantifiers (universal or existential). A sentence is said to be *preserved upwards* if and only if, whenever it is true in an interpretation $\mathcal{P}$, and there is an embedding of $\mathcal{P}$ in another interpretation $\mathcal{Q}$, then it is true in $\mathcal{Q}$. Show that every existential sentence is preserved upwards.

13.10 Let A be a sentence that is preserved upwards, $\mathcal{P}$ a model of A, and $\Delta(\mathcal{P})$ the diagram of $\mathcal{P}$. Show that $\Delta \cup \{\sim A\}$ is unsatisfiable, and that some finite subset of $\Delta \cup \{\sim A\}$ is unsatisfiable.

13.11 Let A be a sentence of a language L that is preserved upwards. Show that:

(a) $\mathcal{P}$ is a model of A if and only if there is a quantifier-free sentence B of the language $L^{\mathcal{P}}$ such that B implies A and $\mathcal{P}^*$ is a model of B.

(b) $\mathcal{P}$ is a model of A if and only if there is an existential sentence B of the language L such that B implies A and $\mathcal{P}$ is a model of B.

13.12 Let A be a sentence that is preserved upwards, and Γ the set of existential sentences of the language of A that imply A. Writing $\sim\Gamma$ for the set of negations of elements of Γ, show that:

(a) $\{A\} \cup \sim\Gamma$ is unsatisfiable.

(b) $\{A\} \cup \sim\Gamma_0$ is unsatisfiable for some finite subset Γ_0 of Γ.

(c) $\{A\} \cup \{\sim B\}$ is unsatisfiable for some single element of Γ.

13.13 Let A be a sentence that is preserved upwards. Show that A is logically equivalent to an existential sentence (in the same language).

13.14 A sentence is called *universal* if and only if it is of the form $\forall x_1 \ldots \forall x_n F$ where F contains no further quantifiers (universal or existential). A sentence is said to be *preserved downwards* if and only if, whenever it is true in an interpretation $\mathcal{Q}$, and there is an embedding of $\mathcal{P}$ in another interpretation $\mathcal{Q}$, then it is true in $\mathcal{P}$. Prove that a sentence is preserved downwards if and only if it is logically equivalent to a universal sentence (in the same language).

13.15 The proof in the preceding several problems involves (at the step of Problem 13.10) applying the compactness theorem to a language that may be nonenumerable. How could this feature be avoided?

14

Proofs and Completeness

Introductory textbooks in logic devote much space to developing one or another kind of proof procedure*, enabling one to recognize that a sentence D is implied by a set of sentences* Γ*, with different textbooks favoring different procedures. In this chapter we introduce the kind of proof procedure, called a* Gentzen system *or* sequent calculus*, that is used in more advanced work, where in contrast to introductory textbooks the emphasis is on general theoretical results about the existence of proofs, rather than practice in constructing specific proofs. The details of any particular procedure, ours included, are less important than some features shared by all procedures, notably the features that whenever there is a proof of D from* Γ*, D is a consequence of* Γ*, and conversely, whenever D is a consequence of* Γ*, there is a proof of D from* Γ*. These features are called* soundness *and* completeness*, respectively. (Another feature is that definite, explicit rules can be given for determining in any given case whether a purported proof or deduction really is one or not; but we defer detailed consideration of this feature to the next chapter.) Section 14.1 introduces our version or variant of sequent calculus. Section 14.2 presents proofs of soundness and completeness. The former is easy; the latter is not so easy, but all the hard work for it has been done in the previous chapter. Section 14.3, which is optional, comments briefly on the relationship of our formal notion to other such formal notions, as might be found in introductory textbooks or elsewhere, and of any formal notion to the unformalized notion of a deduction of a conclusion from a set of premisses, or proof of a theorem from a set of axioms.*

14.1 Sequent Calculus

The idea in setting up a *proof procedure* is that even when it is not obvious that Γ implies D, we may hope to break the route from Γ to D down into a series of small steps that *are* obvious, and thus render the implication relationship recognizable. Every introductory textbook develops some kind of formal notion of proof or deduction. Though these take different shapes in different books, in every case a formal deduction is some kind of finite array of symbols, and there are definite, explicit rules for determining whether a given finite array of symbols is or is not a formal deduction. The notion of deduction is 'syntactic' in the sense that these rules mention the internal structure of formulas, but do not mention interpretations. In the end, though, the condition that there exists a deduction of D from Γ turns out to be *equivalent* to the condition that every interpretation making all sentences in Γ true makes the sentence D true, which was the original 'semantic' definition of consequence. This equivalence has two directions. The result that whenever D is deducible from Γ, D is a consequence of Γ, is the *soundness theorem*. The result that whenever D is a consequence of Γ, then D is deducible from Γ, is the *Gödel completeness theorem*.

Our goal in this chapter will be to present a particular system of deduction for which soundness and completeness can be established. The proof of completeness uses the main lemma from the preceding chapter. Our system, which is of the general sort used in more advanced, theoretical studies, will be different from that used in virtually any introductory textbook—or to put a positive spin on it, virtually no reader will have an advantage over any other reader of previous acquaintance with the particular kind of system we are going to be using. Largely for the benefit of readers who have been or will be looking at other books, in the final section of the chapter we briefly indicate the kinds of variations that are possible and are actually to be met with in the literature. But as a matter of fact, it is not the details of any particular system that really matter, but rather the common features shared by all such systems, and except for a brief mention at the end of the next chapter (in a section that itself is optional reading), we will when this chapter is over never again have occasion to mention the details of our particular system or any other. The existence of *some* proof procedure or other with the properties of soundness and completeness will be the result that will matter.

[Let us indicate one consequence of the existence of such a procedure that will be looked at more closely in the next chapter. It is known that the consequence relation is not *effectively decidable*: that there cannot be a procedure, governed by definite and explicit rules, whose application would, in every case, in principle enable one to determine in a finite amount of time *whether or not* a given finite set Γ of sentences implies a given sentence D. Two proofs of this fact appear in sections 11.1 and 11.2, with another to come in chapter 17. But the existence of a sound and complete proof procedure shows that the consequence relation is at least (*positively*) *effectively semidecidable*. There is a procedure whose application would, in case Γ does imply D, in principle enable one to determine in a finite amount of time *that it does so*. The procedure is simply to search systematically through all finite objects of the appropriate kind, determining for each whether or not it constitutes a deduction of D from Γ. For it is part of the notion of a proof *procedure* that there are definite and explicit rules for determining whether a given finite object of the appropriate sort does or does not constitute such a deduction. If Γ does imply D, then checking through all possible deductions one by one, one would by completeness eventually find one that is a deduction of D from Γ, thus by soundness showing that Γ does imply D; but if Γ does *not* imply D, checking through all possible deductions would go on forever without result. As we said, these matters will be further discussed in the next chapter.]

At the same time one looks for a syntactic notion of deduction to capture and make recognizable the semantic notion of consequence, one would like to have also a syntactic notion of *refutation* to capture the semantic notion of unsatisfiability, and a syntactic notion of *demonstration* to capture the semantic notion of validity. At the cost of some very slight artificiality, the three notions of consequence, unsatisfiability, and validity can be subsumed as special cases under a single, more general notion. We say that one set of sentences Γ *secures* another set of sentences Δ if every interpretation that makes all sentences in Γ true makes some sentence in

Δ true. (Note that when the sets are finite, $\Gamma = \{C_1, \ldots, C_m\}$ and $\Delta = \{D_1, \ldots, D_n\}$, this amounts to saying that every interpretation that makes C_1 & ... & C_m true makes $D_1 \vee \cdots \vee D_n$ true: the elements of Γ are being taken *jointly* as premisses, but the elements of Δ are being taken *alternatively* as conclusions, so to speak.) When a set contains but a single sentence, then of course making *some* sentence in the set true and making *every* sentence in the set true come to the same thing, namely, making *the* sentence in the set true; and in this case we naturally speak of the sentence as doing the securing or as being secured. When the set is empty, then of course the condition that *some* sentence in it is made true is not fulfilled, since there is no sentence in it to be made true; and we count the condition that *every* sentence in the set is made true as being 'vacuously' fulfilled. (After all, there is no sentence in the set that is *not* made true.) With this understanding, consequence, unsatisfiability, and validity can be seen to be special cases of security in the way listed in Table 14-1.

Table 14-1. *Metalogical notions*

D is a consequence of Γ	if and only if	Γ secures $\{D\}$
Γ is unsatisfiable	if and only if	Γ secures $\varnothing$
D is valid	if and only if	$\varnothing$ secures $\{D\}$

Correspondingly, our approach to deductions will subsume them along with refutations and demonstrations under a more general notion of *derivation*. Thus for us the soundness and completeness theorems will be theorems relating a syntactic notion of derivability to a semantic notion of security, from which relationship various other relationships between syntactic and semantic notions will follow as special cases. The objects with which we are going to work in this chapter—the objects of which derivations will be composed—are called *sequents*. A sequent $\Gamma \Rightarrow \Delta$ consists of a finite set of sentences Γ on the left, the symbol $\Rightarrow$ in the middle, and a finite set of sentences Δ on the right. We call the sequent *secure* if its left side Γ secures its right side Δ. The goal will be to define a notion of derivation so that there will be a derivation of a sequent if and only if it is secure.

Deliberately postponing the details of the definition, we just for the moment say that a derivation will be a kind of finite sequence of sequents, called the *steps* (or *lines*) of the derivation, subject to certain syntactic conditions or rules that remain to be stated. A derivation will be a derivation *of* a sequent $\Gamma \Rightarrow \Delta$ if and only if that sequent is its last step (or bottom line). A sequent will be *derivable* if and only if there is some derivation of it. It is in terms of this notion of derivation that we will define other syntactic notions of interest, as in Table 14-2.

Table 14-2. *Metalogical notions*

A *deduction* of D from Γ	is a	derivation of $\Gamma \Rightarrow \{D\}$
A *refutation* of Γ	is a	derivation of $\Gamma \Rightarrow \varnothing$
A *demonstration* of D	is a	derivation of $\varnothing \Rightarrow \{D\}$

We naturally say that D is *deducible* from Γ if there is a deduction of D from Γ, that Γ is *refutable* if there is a refutation of Γ, and that D is *demonstrable* if there is a demonstration of D, where deduction, refutation, and demonstration are defined in terms of derivation as in Table 14-2. An irrefutable set of sentences is also called *consistent*, and a refutable one *inconsistent*. Our main goal will be so to define the notion of derivation that we can prove the following two theorems.

14.1 Theorem (Soundness theorem). Every derivable sequent is secure.

14.2 Theorem (Gödel completeness theorem). Every secure sequent is derivable.

It will then immediately follow (on comparing Tables 14-1 and 14-2) that there is an exact coincidence between two parallel sets of metalogical notions, the semantic and the syntactic, as shown in Table 14-3.

Table 14-3. *Correspondences between metalogical notions*

D is deducible from Γ	if and only if	D is a consequence of Γ
Γ is inconsistent	if and only if	Γ is unsatisfiable
D is demonstrable	if and only if	D is valid

To generalize to the case of infinite sets of sentences, we simply *define* Δ to be derivable from Γ if and only if some finite subset Δ_0 of Δ is derivable from some finite subset Γ_0 of Γ, and define deducibility and inconsistency in the infinite case similarly. As an easy corollary of the compactness theorem, Γ secures Δ if and only if some finite subset Γ_0 of Γ secures some finite subset Δ_0 of Δ. Thus Theorems 14.1 and 14.2 will extend to the infinite case: Δ will be derivable from Γ if and only if Δ is secured by Γ, even when Γ and Δ are infinite.

So much by way of preamble. It remains, then, to specify what conditions a sequence of sequents must fulfill in order to count as a derivation. In order for a sequence of steps to qualify as a derivation, each step must either be of the form $\{A\} \Rightarrow \{A\}$ or must follow from earlier steps according to one of another of several *rules of inference* permitting passage from one or more sequents taken as *premisses* to some other sequent taken as *conclusion*. The usual way of displaying rules is to write the premiss or premisses of the rule, a line below them, and the conclusion of the rule. The provision that a step may be of the form $\{A\} \Rightarrow \{A\}$ may itself be regarded as a special case of a rule of inference with *zero* premisses; and in listing the rules of inference, we in fact list this one first. In general, in the case of any rule, any sentence that appears in a premiss but not the conclusion of a rule is said to be *exiting*, any that appears in the conclusion but not the premisses is said to be *entering*, and any that appears in both a premiss and the conclusion is said to be *standing*. In the special case of the zero-premiss rule and steps of the form $\{A\} \Rightarrow \{A\}$, the sentence A counts as entering. It will be convenient in this chapter to work as in the preceding chapter with a version of first-order logic in which the only logical symbols are $\sim$, $\vee$, $\exists$, $=$, that is, in which & and $\forall$ are treated as unofficial abbreviations. (If we admitted & and $\forall$, there would be a need for four more rules, two for each. Nothing

Table 14-4. *Rules of sequent calculus*

(R0)	$\dfrac{}{\{A\} \Rightarrow \{A\}}$	
(R1)	$\dfrac{\Gamma \Rightarrow \Delta}{\Gamma' \Rightarrow \Delta'}$	Γ subset of Γ', Δ subset of Δ'
(R2*a*)	$\dfrac{\Gamma \cup \{A\} \Rightarrow \Delta}{\Gamma \Rightarrow \{\sim A\} \cup \Delta}$	
(R2*b*)	$\dfrac{\Gamma \Rightarrow \{A\} \cup \Delta}{\Gamma \cup \{\sim A\} \Rightarrow \Delta}$	
(R3)	$\dfrac{\Gamma \Rightarrow \{A, B\} \cup \Delta}{\Gamma \Rightarrow \{(A \vee B)\} \cup \Delta}$	
(R4)	$\dfrac{\begin{array}{c}\Gamma \cup \{A\} \Rightarrow \Delta \\ \Gamma \cup \{B\} \Rightarrow \Delta\end{array}}{\Gamma \cup \{A \vee B\} \Rightarrow \Delta}$	
(R5)	$\dfrac{\Gamma \Rightarrow \{A(s)\} \cup \Delta}{\Gamma \Rightarrow \{\exists x A(x)\} \cup \Delta}$	
(R6)	$\dfrac{\Gamma \cup \{A(c)\} \Rightarrow \Delta}{\Gamma \cup \{\exists x A(x)\} \Rightarrow \Delta}$	c not in Γ or Δ or $A(x)$
(R7)	$\dfrac{\Gamma \cup \{s = s\} \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}$	
(R8*a*)	$\dfrac{\Gamma \Rightarrow \{A(t)\} \cup \Delta}{\Gamma \cup \{s = t\} \Rightarrow \{A(s)\} \cup \Delta}$	
(R8*b*)	$\dfrac{\Gamma \cup \{A(t)\} \Rightarrow \Delta}{\Gamma \cup \{s = t, A(s)\} \Rightarrow \Delta}$	
(R9*a*)	$\dfrac{\Gamma \cup \{\sim A\} \Rightarrow \Delta}{\Gamma \Rightarrow \{A\} \cup \Delta}$	
(R9*b*)	$\dfrac{\Gamma \Rightarrow \{\sim A\} \cup \Delta}{\Gamma \cup \{A\} \Rightarrow \Delta}$	

would be harder, but everything would be more tedious.) With this understanding, the rules are those give in Table 14-4.

These rules roughly correspond to patterns of inference used in unformalized deductive argument, and especially mathematical proof. (R2*a*) or *right negation introduction* corresponds to 'proof by contradiction', where an assumption A is shown to be inconsistent with background assumptions Γ and it is concluded that those background assumptions imply its negation. (R2*b*) or *left negation introduction* corresponds to the inverse form of inference. (R3) or *right disjunction introduction*, together with (R1), allows us to pass from $\Gamma \Rightarrow \{A\} \cup \Delta$ or $\Gamma \Rightarrow \{B\} \cup \Delta$ by way of $\Gamma \Rightarrow \{A, B\} \cup \Delta$ to $\Gamma \Rightarrow \{(A \vee B)\} \cup \Delta$, which corresponds to inferring a disjunction from one disjunct. (R4) or *left disjunction introduction* corresponds to 'proof by cases', where something that has been shown to follow from each disjunct is concluded to follow from a disjunction. (R5) or *right existential quantifier introduction* corresponds to inferring an existential generalization from a particular instance. (R6) or *left existential-quantifier introduction* is a bit subtler: it corresponds to a common procedure in mathematical proof where, assuming there is something for which a condition A holds, we 'give it a name' and say 'let c be something for which the condition A holds', where c is some previously unused name, and thereafter proceed to count whatever statements not mentioning c that can be shown to follow from the assumption that condition A holds for c as following from the original assumption that there is *something* for which condition A holds. (R8a, b) correspond to two forms of 'substituting equals for equals'.

A couple of trivial examples will serve show how derivations are written.

14.3 Example. The deduction of a disjunction from a disjunct.

(1)	$A \Rightarrow A$	(R0)
(2)	$A \Rightarrow A, B$	(R1), (1)
(3)	$A \Rightarrow A \vee B$	(R3), (2)

The first thing to note here is that though officially what occur on the left and right sides of the double arrow in a sequent are sets, and sets have no intrinsic order among their elements, in *writing* a sequent, we do have to write those elements in some order or other. $\{A, B\}$ and $\{B, A\}$ and for that matter $\{A, A, B\}$ are the same *set*, and therefore $\{A\} \Rightarrow \{A, B\}$ and $\{A\} \Rightarrow \{B, A\}$ and for that matter $\{A\} \Rightarrow \{A, A, B\}$ are the same *sequent*, but we have chosen to *write* the sequent the first way. Actually, we have not written the braces at all, nor will they be written in future when writing out derivations. [For that matter, have also been writing $A \vee B$ for $(A \vee B)$, and will be writing Fx for $F(x)$ below.] An alternative approach would be to have *sequences* rather than *sets* of formulas on both sides of a sequent, and introduce additional 'structural' rules allowing one to reorder the sentences in a sequences, and for that matter, to introduce or eliminate repetitions.

The second thing to note here is that the numbering of the lines on the left, and the annotations on the right, are not officially part of the derivation. In practice, their presence makes it easier to check that a purported derivation really is one; but in principle it can be checked whether a string symbols constituties a derivation even without such annotation. For there are, after all, at each step only finitely many rules that could possibly have been applied to get that step from earlier steps, and only finitely many earlier steps any rule could possibly have been applied to, and in principle we need only check through these finitely many possibilities to find whether there is a justification for the given step.

14.4 Example. The deduction of a conjunct from a conjunction

(1)	$A \Rightarrow A$	(R0)
(2)	$A, B \Rightarrow A$	(R1), (1)
(3)	$B \Rightarrow A, \sim A$	(R2a), (2)
(4)	$\Rightarrow A, \sim A, \sim B$	(R2a), (3)
(5)	$\Rightarrow A, \sim A \vee \sim B$	(R3), (4)
(6)	$\sim(\sim A \vee \sim B) \Rightarrow A$	(R2b), (5)
(7)	$A \,\&\, B \Rightarrow A$	abbreviation, (6)

Here the last step, reminding us that $\sim(\sim A \vee \sim B)$ is what $A \,\&\, B$ abbreviates, is unofficial, so to speak. We omit the word 'abbreviation' in such cases in the future. It is *because* & is not in the official notation, and we do not directly have rules for it, that the derivation in this example needs more steps than that in the preceding example.

Since the two examples so far have both been of derivations constituting deductions, let us give two equally short examples of derivations constituting refutations and demonstrations.

14.5 Example. Demonstration of a tautology

(1)	$A \Rightarrow A$	(R0)
(2)	$\Rightarrow A, \sim A$	(R2*b*), (1)
(3)	$\Rightarrow A \vee \sim A$	(R3), (2)

14.6 Example. Refutation of a contradiction

(1)	$\sim A \Rightarrow \sim A$	(R0)
(2)	$\Rightarrow \sim A, \sim\sim A$	(R2*b*), (1)
(3)	$\Rightarrow \sim A \vee \sim\sim A$	(R3), (2)
(4)	$\sim(\sim A \vee \sim\sim A) \Rightarrow$	(R2*a*), (3)
(5)	$A \,\&\, \sim A \Rightarrow$	(4)

The remarks above about the immateriality of the *order* in which sentences are written are especially pertinent to the next example.

14.7 Example. Commutativity of disjunction

(1)	$A \Rightarrow A$	(R0)
(2)	$A \Rightarrow B, A$	(R1), (1)
(3)	$A \Rightarrow B \vee A$	(R3), (2)
(4)	$B \Rightarrow B$	(R0)
(5)	$B \Rightarrow B, A$	(R1), (4)
(6)	$B \Rightarrow B \vee A$	(R3), (5)
(7)	$A \vee B \Rightarrow B \vee A$	(R4), (3), (6)

The commutativity of conjunction would be obtained similarly, though there would be more steps, for the same reason that there are more steps in Examples 14.4 and 14.6 than in Examples 14.3 and 14.5. Next we give a couple of somewhat more substantial examples, illustrating how the quantifier rules are to be used, and a couple of counter-examples to show how they are *not* to be used.

14.8 Example. Use of the first quantifier rule

(1)	$Fc \Rightarrow Fc$	(R0)
(2)	$\Rightarrow Fc, \sim Fc$	(R2*b*), (1)
(3)	$\Rightarrow \exists x Fx, \sim Fc$	(R5), (2)
(4)	$\Rightarrow \exists x Fx, \exists x \sim Fx$	(R5), (3)
(5)	$\sim\exists x \sim Fx \Rightarrow \exists x Fx$	(R2*a*), (4)
(6)	$\forall x Fx \Rightarrow \exists x Fx$	(5)

14.9 Example. Proper use of the two quantifier rules

(1)	$Fc \Rightarrow Fc$	(R0)
(2)	$Fc \Rightarrow Fc,\ Gc$	(R1), (1)
(3)	$Gc \Rightarrow Gc$	(R0)
(4)	$Gc \Rightarrow Fc,\ Gc$	(R1), (3)
(5)	$Fc \vee Gc \Rightarrow Fc,\ Gc$	(R4), (2), (4)
(6)	$Fc \vee Gc \Rightarrow \exists x Fx,\ Gc$	(R5), (5)
(7)	$Fc \vee Gc \Rightarrow \exists x Fx, \exists x Gx$	(R5), (6)
(8)	$Fc \vee Gc \Rightarrow \exists x Fx \vee \exists x Gx$	(R3), (7)
(9)	$\exists x(Fx \vee Gx) \Rightarrow \exists x Fx \vee \exists x Gx$	(R6), (8)

14.10 Example. Improper use of the second quantifier rule

(1)	$Fc \Rightarrow Fc$	(R0)
(2)	$Fc, {\sim}Fc \Rightarrow$	(R2*b*), (1)
(3)	$\exists x Fx, {\sim}Fc \Rightarrow$	(R6), (2)
(4)	$\exists x Fx, \exists x \sim Fx \Rightarrow$	(R6), (3)
(5)	$\exists x Fx \Rightarrow {\sim}\exists x \sim Fx$	(R2*b*), (4)
(6)	$\exists x Fx \Rightarrow \forall x Fx$	(5)

Since $\exists x Fx$ does *not* imply $\forall x Fx$, there must be something wrong in this last example, either with our rules, or with the way they have been deployed in the example. In fact, it is the deployment of (R6) at line (3) that is illegitimate. Specifically, the side condition 'c not in Γ' in the official statement of the rule is not met, since the relevant Γ in this case would be $\{{\sim}Fc\}$, which contains c. Contrast this with a legitimate application of (R6) as at the last line in the preceding example. Ignoring the side condition 'c not in Δ' can equally lead to trouble, as in the next example. (Trouble can equally arise from ignoring the side condition 'c not in $A(x)$', but we leave it to the reader to provide an example.)

14.11 Example. Improper use of the second quantifier rule

(1)	$Fc \Rightarrow Fc$	(R0)
(2)	$\exists x Fx \Rightarrow Fc$	(R6), (1)
(3)	$\exists x Fx, {\sim}Fc \Rightarrow$	(R2*b*), (2)
(4)	$\exists x Fx, \exists x \sim Fx \Rightarrow$	(R6), (3)
(5)	$\exists x Fx \Rightarrow {\sim}\exists x \sim Fx$	(R2*a*), (4)
(6)	$\exists x Fx \Rightarrow \forall x Fx$	(5)

Finally, let us illustrate the use of the identity rules.

14.12 Example. Reflexivity of identity

(1)	$c = c \Rightarrow c = c$	(R0)
(2)	$\Rightarrow c = c$	(R7), (1)
(3)	${\sim}c = c \Rightarrow$	(R2*b*), (2)
(4)	$\exists x \sim x = x \Rightarrow$	(R6), (3)
(5)	$\Rightarrow {\sim}\exists x \sim x = x$	(R2*a*), (4)
(6)	$\Rightarrow \forall x\, x = x$	(5)

14.13 Example. Symmetry of identity

(1)	$d = d \Rightarrow d = d$	(R0)
(2)	$d = d,\ c = d \Rightarrow d = c$	(R8*a*), (1)
(3)	$c = d \Rightarrow d = c$	(R7), (2)
(4)	$\Rightarrow {\sim}c = d,\ d = c$	(R2*a*), (3)
(5)	$\Rightarrow {\sim}c = d \lor d = c$	(R3), (4)
(6)	$\Rightarrow c = d \to d = c$	(5)
(7)	${\sim}(c = d \to d = c) \Rightarrow$	(R2*b*), (6)
(8)	$\exists y\, {\sim}(c = y \to y = c) \Rightarrow$	(R6), (7)
(9)	$\Rightarrow {\sim}\exists y\, {\sim}(c = y \to y = c)$	(R2*a*), (8)
(10)	$\Rightarrow \forall y(c = y \to y = c)$	(9)
(11)	${\sim}\forall y(c = y \to y = c) \Rightarrow$	(R2*b*), (10)
(12)	$\exists x\, {\sim}\forall y(x = y \to y = x) \Rightarrow$	(R6), (11)
(13)	$\Rightarrow {\sim}\exists x\, {\sim}\forall y(x = y \to y = x)$	(R2*a*), (12)
(14)	$\Rightarrow \forall x \forall y(x = y \to y = x)$	(13)

The formula $A(x)$ to which (R8*a*) has been applied at line (2) is $d = x$.

14.2 Soundness and Completeness

Let us now begin the proof of soundness, Theorem 14.1, according to which every derivable sequent is secure. We start with the observation that every (R0) sequent $\{A\} \Rightarrow \{A\}$ is clearly secure. It will then suffice to show that each rule (R1)–(R9) is *sound* in the sense that when applied to secure premisses it yields secure conclusions.

Consider, for instance, an application of (R1). Suppose $\Gamma \Rightarrow \Delta$ is secure, where Γ is a subset of Γ' and Δ is a subset of Δ', and consider any interpretation that makes all the sentences in Γ' true. What (R1) requires is that it should make some sentence in Δ' true, and we show that it does as follows. Since Γ is a subset of Γ', it makes all the sentences in Γ true, and so by the security of $\Gamma \Rightarrow \Delta$ it makes some sentence in Δ true and, since Δ is a subset of Δ', thereby makes some sentence of Δ' true.

Each of the rules (R2)–(R9) must now be checked in a similar way. Since this proof is perhaps the most tedious in our whole subject, it may be well to remark in advance that it does have one interesting feature. The feature is this: that as we argue for the soundness of the formal rules, we are going to find ourselves using something like the unformalized counterparts of those very rules in our argumentation. This means that a mathematical heretic who rejected one of another of the usual patterns of argument as employed in unformalized proofs in orthodox mathematics—and there have been benighted souls who have rejected the informal counterpart of (R9), for example—would not accept our proof of the soundness theorem. The point of the proof is not to convince such dissenters, but merely to check that, in putting everything into symbols, we have not made some slip and allowed some inference that, stated in unformalized terms, we ourselves would recognize as fallacious. (This is a kind of mistake that it is not hard to make, especially over the side conditions in the quantifier rule, and it is one that has been made in the past in some textbooks.) This noted, let us now return to the proof.

Consider (R2*a*). We suppose $\Gamma \cup \{A\} \Rightarrow \Delta$ is secure, and consider any interpretation that makes all the sentences in Γ true. What (R2*a*) requires is that it should make some sentence in $\{\sim A\} \cup \Delta$ true, and we show it does as follows. On the one hand, if the given interpretation also makes A true, then it makes all the sentences in $\Gamma \cup \{A\}$ true, and therefore by the security of $\Gamma \cup \{A\} \Rightarrow \Delta$ makes some sentence in Δ true, and therefore makes some sentence in $\{\sim A\} \cup \Delta$ true. On the other hand, if the interpretation does not make A true, then it makes $\sim A$ true, and therefore it again makes some sentence in $\{\sim A\} \cup \Delta$ true.

Consider (R2*b*). We suppose $\Gamma \Rightarrow \{A\} \cup \Delta$ is secure, and consider any interpretation making all sentences in $\Gamma \cup \{\sim A\}$ true. What (R2*b*) requires is that it should make some sentence in Δ true, and we show it does as follows. The given interpretation makes all sentences in Γ true, and so by the security of $\Gamma \Rightarrow \{A\} \cup \Delta$ makes some sentence in $\{A\} \cup \Delta$ true. But since the interpretation makes $\sim A$ true, it does not make A true, so it must be that it makes some sentence in Δ true.

For (R3), we suppose that $\Gamma \Rightarrow \{A, B\} \cup \Delta$ is secure, and consider any interpretation making all sentences in Γ true. By the security of $\Gamma \Rightarrow \{A, B\} \cup \Delta$ the interpretation makes some sentence in $\{A, B\} \cup \Delta$ true. This sentence must be either A or B or some sentence in Δ. If the sentence is A or B, then the interpretation makes $(A \vee B)$ true, and so makes a sentence in $\{(A \vee B)\} \cup \Delta$ true. If the sentence is one of those in Δ, then clearly the interpretation makes a sentence in $\{(A \vee B)\} \cup \Delta$ true. So in any case, some sentence in $\{(A \vee B)\} \cup \Delta$ is made true, which is what (R3) requires.

For (R4), we suppose that $\Gamma \cup \{A\} \Rightarrow \Delta$ and $\Gamma \cup \{B\} \Rightarrow \Delta$ are secure, and consider any interpretation that makes all sentences in $\Gamma \cup \{(A \vee B)\}$ true. The interpretation in particular makes $(A \vee B)$ true, and so it must either make A true or make B true. In the former case it makes all sentences in $\Gamma \cup \{A\}$ true, and by the security of $\Gamma \cup \{A\} \Rightarrow \Delta$ it makes some sentence in Δ true. Similarly in the latter case. So in either case it makes some sentence in Δ true, which is what (R4) requires.

For (R5), we suppose that $\Gamma \Rightarrow \{A(s)\} \cup \Delta$ is secure and consider any interpretation that makes all sentences in Γ true. By the security of $\Gamma \Rightarrow \{A(s)\} \cup \Delta$ it makes some sentence in $\{A(s)\} \cup \Delta$ true. If the sentence is one in Δ, then clearly the interpretation makes some sentence in $\{\exists x A(x)\} \cup \Delta$ true. If the sentence is $A(s)$, then the interpretation makes $\exists x A(x)$ true, and so again the interpretation makes some sentence in $\{\exists x A(x)\} \cup \Delta$ true. This suffices to show that $\Gamma \Rightarrow \{\exists x A(x)\} \cup \Delta$ is secure, which is what (R5) requires.

For (R6), we suppose that $\Gamma \cup \{A(c)\} \Rightarrow \Delta$ is secure and consider any interpretation making all sentences in $\Gamma \cup \{\exists x A(x)\}$ true. Since the interpretation makes $\exists x A(x)$ true, there is some element i in the domain of the interpretation that satisfies $A(x)$. If c does not occur in Γ or Δ or $A(x)$, then while leaving the denotations of all symbols that occur in Γ and Δ and $A(x)$ unaltered, we can alter the interpretation so that the denotation of c becomes i. By extensionality, in the new interpretation every sentence in Γ will still be true, i will still satisfy $A(x)$ in the new interpretation, and every sentence in Δ will have the same truth value as in the old interpretation. But since i is now the denotation of c, and i satisfies $A(x)$, it follows that $A(c)$ will be true in the new interpretation. And since the sentences in Γ are still true and $A(c)$ is now true, by the security of $\Gamma \cup \{A(c)\} \Rightarrow \Delta$, some sentence in Δ true will be true in the new interpretation and hence will have been true in old interpretation. This suffices to show that $\Gamma \cup \{\exists x A(x)\} \Rightarrow \Delta$ is secure.

For (R7), we suppose $\Gamma \cup \{s = s\} \Rightarrow \Delta$ is secure, and consider any interpretation of a language containing all symbols in Γ and Δ that makes all sentences in Γ true. If there is some symbol in s not occurring in Γ or Δ to which this interpretation fails to assign a denotation, alter it so that it does. The new interpretation will still make every sentence in Γ true by extensionality, and will make $s = s$ true. By the security of $\Gamma \cup \{s = s\} \Rightarrow \Delta$, the new interpretation will make some sentence in Δ true, and extensionality implies that the original interpretation already made this same sentence in Δ true. This suffices to show that $\Gamma \Rightarrow \Delta$ is secure.

For (R8*a*), we suppose $\Gamma \Rightarrow \{A(t)\} \cup \Delta$ is secure and consider any interpretation making all sentences in $\Gamma \cup \{s = t\}$ true. Since it makes every sentence in Γ true, by the security of $\Gamma \Rightarrow \{A(t)\} \cup \Delta$ it must make some sentence in $\{A(t)\} \cup \Delta$ true. If this sentence is one of those in Δ, then clearly the interpretation makes a sentence in $\{A(s)\} \cup \Delta$ true. If the sentence is $A(t)$, then note that since the interpretation makes $s = t$ true, it must assign the same denotation to s and to t, and therefore by the must also make $A(s)$ true by extensionality. Thus again it makes some sentence in $\{A(s)\} \cup \Delta$ true. This suffices to show that $\Gamma \cup \{s = t\} \Rightarrow \{A(s)\} \cup \Delta$ is secure. (R8b) is entirely similar.

(R9) is just like (R2), to finish the proof of soundness.

Now, for completeness, Theorem 14.2, according to which every secure sequent is derivable. We begin with a quick reduction of the problem. Write $\sim\Delta$ for the set of negations of sentences in Δ.

14.14 Lemma. $\Gamma \Rightarrow \Delta$ is derivable if and only if $\Gamma \cup \sim\Delta$ is inconsistent.

Proof: If

$$\{C_1, \ldots, C_m\} \Rightarrow \{D_1, \ldots, D_n\}$$

is derivable, then

$$\begin{aligned} \{C_1, \ldots, C_m, \sim D_1\} &\Rightarrow \{D_2, \ldots, D_n\} \\ \{C_1, \ldots, C_m, \sim D_1, \sim D_2\} &\Rightarrow \{D_3, \ldots, D_n\} \\ &\vdots \\ \{C_1, \ldots, C_m, \sim D_1, \ldots, \sim D_n\} &\Rightarrow \varnothing \end{aligned}$$

are derivable by repeated application of (R2*b*). If the last of these is derivable, then

$$\begin{aligned} \{C_1, \ldots, C_m, \sim D_2, \ldots, \sim D_n\} &\Rightarrow \{D_1\} \\ \{C_1, \ldots, C_m, \sim D_3, \ldots, \sim D_n\} &\Rightarrow \{D_1, D_2\} \\ &\vdots \\ \{C_1, \ldots, C_m\} &\Rightarrow \{D_1, \ldots, D_n\} \end{aligned}$$

are derivable by repeated application of (R9*a*).

Since it is easily seen that Γ secures Δ if and only if $\Gamma \cup {\sim}\Delta$ is unsatisfiable, proving that if Γ secures Δ, then $\Gamma \Rightarrow \Delta$ is derivable, which is what we want to do, reduces to showing that any consistent set is satisfiable. (For if Γ secures Δ, then $\Gamma \cup {\sim}\Delta$ is unsatisfiable, and supposing we have succeeded in showing that it would be satisfiable if it were consistent, it follows $\Gamma \cup {\sim}\Delta$ is inconsistent, and so by the preceding lemma $\Gamma \Rightarrow \Delta$ is derivable.) By the main lemma of the preceding chapter, in order to show every consistent set is satisfiable, it will suffice to show that the set S of all consistent sets has the satisfiability properties (S0)–(S8). (For any consistent set Γ will by definition belong to S, and what Lemma 13.3 tells us is that if S has the satisfaction properties, then any element of S is satisfiable.) This we now proceed to verify, recalling the statements of properties (S0)–(S8) one by one as we prove S has them.

Consider (S0). This says that if Γ is in S and Γ_0 is a subset of Γ, then Γ_0 is in S. So what we need to prove is that if $\Gamma \Rightarrow \varnothing$ is not derivable, and Γ_0 is a subset of Γ, then $\Gamma_0 \Rightarrow \varnothing$ is not derivable. Contraposing, this is equivalent to proving:

(S0) If $\Gamma_0 \Rightarrow \varnothing$ is derivable, and Γ_0 is a subset of Γ, then $\Gamma \Rightarrow \varnothing$ is derivable.

We show this by indicating how to extend any given derivation of $\Gamma_0 \Rightarrow \varnothing$ to a derivation of $\Gamma \Rightarrow \varnothing$. In fact, only one more step need be added, as follows:

$$\begin{array}{cr} \vdots & \\ \Gamma_0 \Rightarrow \varnothing & \text{Given} \\ \Gamma \Rightarrow \varnothing. & \text{(R1)} \end{array}$$

(Here the three dots represent the earlier steps of the hypothetical derivation of $\Gamma_0 \Rightarrow \varnothing$.)

For each of (S1)–(S8) we are going to give a restatement, in contraposed form, of what is to be proved, and then show how to prove it by extending a given derivation to a derivation of the sequent required. First (S1)

(S1) If A and ${\sim}A$ are both in Γ, then $\Gamma \Rightarrow \varnothing$ is derivable.

The hypothesis may be restated as saying that $\{A, {\sim}A\}$ is a subset of Γ. We then have

$$\begin{array}{rr} \{A\} \Rightarrow \{A\} & \text{(R0)} \\ \{A, {\sim}A\} \Rightarrow \varnothing & \text{(R2}a\text{)} \\ \Gamma \Rightarrow \varnothing. & \text{(R1)} \end{array}$$

As for (S2), literally, this says that:

(S2) If $\Gamma \Rightarrow \varnothing$ is not derivable and ${\sim\sim}B$ is in Γ, then $\Gamma \cup \{B\} \Rightarrow \varnothing$ is not derivable.

Contraposing, this says that if $\Gamma \cup \{B\} \Rightarrow \varnothing$ *is* derivable and ${\sim\sim}B$ is in Γ, then $\Gamma \Rightarrow \varnothing$ is derivable. What we actually show is that if $\Gamma \cup \{B\} \Rightarrow \varnothing$ is derivable, then whether or not $\sim\sim B$ is in Γ, $\Gamma \cup \{\sim\sim B\} \Rightarrow \varnothing$ is derivable. In case $\sim\sim B$ is in Γ, we have $\Gamma \cup \{\sim\sim B\} = \Gamma$, so what we actually show is something a little more general than what we need:

$$\begin{array}{rr}
\vdots & \\
\Gamma \cup \{B\} \Rightarrow \varnothing & \text{Given} \\
\Gamma \Rightarrow \{\sim B\} & (\text{R2}b) \\
\Gamma \cup \{\sim\sim B\} \Rightarrow \varnothing. & (\text{R2}a)
\end{array}$$

Analogous remarks apply to (S3)–(S8) below.

(S3) If $\Gamma \cup \{B\} \Rightarrow \varnothing$ and $\Gamma \cup \{C\} \Rightarrow \varnothing$ are both derivable, then $\Gamma \cup \{B \vee C\} \Rightarrow \varnothing$ is derivable.

Here we concatenate the two given derivations, writing one after the other:

$$\begin{array}{rr}
\vdots & \\
\Gamma \cup \{B\} \Rightarrow \varnothing & \text{Given} \\
\vdots & \\
\Gamma \cup \{C\} \Rightarrow \varnothing & \text{Given} \\
\Gamma \cup \{B \vee C\} \Rightarrow \varnothing. & (\text{R4})
\end{array}$$

(S4) If either $\Gamma \cup \{\sim B\} \Rightarrow \varnothing$ or $\Gamma \cup \{\sim C\} \Rightarrow \varnothing$ is derivable, then $\Gamma \cup \{\sim(B \vee C)\} \Rightarrow \varnothing$ is derivable.

The two cases are *exactly* alike, and we do only the first:

$$\begin{array}{rr}
\vdots & \\
\Gamma \cup \{\sim B\} \Rightarrow \varnothing & \text{Given} \\
\Gamma \Rightarrow \{B\} & (\text{R9}a) \\
\Gamma \Rightarrow \{B, C\} & (\text{R1}) \\
\Gamma \Rightarrow \{B \vee C\} & (\text{R3}) \\
\Gamma \cup \{\sim(B \vee C)\} \Rightarrow \varnothing. & (\text{R2}a)
\end{array}$$

(S5) If $\Gamma \cup \{B(c)\} \Rightarrow \varnothing$ is derivable, where c does not occur in $\Gamma \cup \{\exists x B(x)\}$, then $\Gamma \cup \{\exists x B(x)\} \Rightarrow \varnothing$ is derivable:

$$\begin{array}{rr}
\vdots & \\
\Gamma \cup \{B(c)\} \Rightarrow \varnothing & \text{Given} \\
\Gamma \cup \{\exists x B(x)\} \Rightarrow \varnothing. & (\text{R6})
\end{array}$$

Note that the hypothesis that c does not occur in Γ or $\exists x B(x)$ (nor of course in $\varnothing$) means that the side conditions for the proper application of (R6) are met.

(S6) If $\Gamma \cup \{\sim B(t)\} \Rightarrow \varnothing$ is derivable for some closed term t, then $\Gamma \cup \{\sim\exists x\, B(x)\} \Rightarrow \varnothing$ is derivable:

$$\begin{array}{rr}
\vdots & \\
\Gamma \cup \{\sim B(t)\} \Rightarrow \varnothing & \text{Given} \\
\Gamma \Rightarrow \{B(t)\} & \text{(R9}a\text{)} \\
\Gamma \Rightarrow \{\exists x\, B(x)\} & \text{(R5)} \\
\Gamma \cup \{\sim\exists x\, B(x)\} \Rightarrow \varnothing. & \text{(R2}a\text{)}
\end{array}$$

(S7) $\Gamma \cup \{t = t\} \Rightarrow \varnothing$ derivable for some closed term t, then $\Gamma \Rightarrow \varnothing$ is derivable:

$$\begin{array}{rr}
\vdots & \\
\Gamma \cup \{t = t\} \Rightarrow \varnothing & \text{Given} \\
\Gamma \Rightarrow \varnothing. & \text{(R7)}
\end{array}$$

(S8) If $\Gamma \cup \{B(t)\} \Rightarrow \varnothing$ is derivable, then $\Gamma \cup \{B(s), s = t\} \Rightarrow \varnothing$ is derivable:

$$\begin{array}{rr}
\vdots & \\
\Gamma \cup \{B(t)\} \Rightarrow \varnothing & \text{Given} \\
\Gamma \Rightarrow \{\sim B(t)\} & \text{(R2}b\text{)} \\
\Gamma \cup \{s = t\} \Rightarrow \{\sim B(s)\} & \text{(R8}a\text{)} \\
\Gamma \cup \{s = t,\ B(s)\} \Rightarrow \varnothing. & \text{(R9}b\text{)}
\end{array}$$

This verification finishes the proof of completeness.

14.3* Other Proof Procedures and Hilbert's Thesis

A great many other sound and complete proof procedures are known. We begin by considering modifications of our own procedure that involve only adding or dropping a rule or two, and first of all consider the result of dropping (R9). The following lemma says that it will not be missed. Its proof gives just a taste of the methods of *proof theory*, a branch of logical studies that otherwise will be not much explored in this book.

14.15 Lemma (Inversion lemma). Using (R0)–(R8):

(a) If there is a derivation of $\Gamma \cup \{\sim A\} \Rightarrow \Delta$, then there is a derivation of $\Gamma \Rightarrow \{A\} \cup \Delta$.

(b) If there is a derivation of $\Gamma \Rightarrow \{\sim A\} \cup \Delta$, then there is a derivation of $\Gamma \cup \{A\} \Rightarrow \Delta$.

Proof: The two parts are similarly proved, and we do only (a). A counterexample to the lemma would be a derivation of a sequent $\Gamma \cup \{\sim A\} \Rightarrow \Delta$ for which no derivation of $\Gamma \Rightarrow \{A\} \cup \Delta$ is possible. We want to show there can be no counterexample by showing that a contradiction follows from the supposition that there is one. Now if there are any counterexamples, among them there must be one that is as short as possible, so that no strictly shorter derivation would be a counterexample. So suppose that $\Gamma \cup \{\sim A\} \Rightarrow \Delta$ is the sequent derived in such a shortest possible counterexample. We ask by what rule the last step $\Gamma \cup \{\sim A\} \Rightarrow \Delta$ could have been justified.

Could it have been (R0)? If that were so, the counterexample would simply be the one-step derivation of $\{\sim A\} \Rightarrow \{\sim A\}$, and we would have $\Gamma = \varnothing$, $\Delta = \{\sim A\}$. The sequent $\Gamma \Rightarrow \{A\} \cup \Delta$ for which supposedly no derivation exists would then just be $\Rightarrow \{A, \sim A\}$. But there *is* a derivation of this sequent, in two steps, starting with $\{A\} \Rightarrow \{A\}$ by (R0) and proceeding to $\Rightarrow \{A, \sim A\}$ by (R2*a*). So (R0) is excluded, and $\Gamma \cup \{\sim A\} \Rightarrow \Delta$ must have been inferred from some earlier step or steps by one of the other rules.

Could it have been (R3)? If that were so, the counterexample would be a derivation of

$$\Gamma \cup \{\sim A\} \Rightarrow \{(B \vee C)\} \cup \Delta'$$

where the last step was obtained from

$$\Gamma \cup \{\sim A\} \Rightarrow \{B, C\} \cup \Delta'.$$

But then, since the derivation down to this last-displayed sequent is too short to be a counterexample, there will be a derivation of

$$\Gamma \Rightarrow \{A\} \cup \{B, C\} \cup \Delta',$$

and by applying (R3) we can then get

$$\Gamma \Rightarrow \{A\} \cup \{(B \vee C)\} \cup \Delta',$$

which is precisely what we are supposed *not* to be able to get in the case of a counterexample to the lemma. Thus (R3) is excluded. Moreover, every case where $\sim A$ is not an entering sentence is excluded for entirely similar reasons.

There remain to be considered three cases where $\sim A$ is an entering sentence. One case where $\sim A$ enters arises when $\Gamma \cup \{\sim A\} \Rightarrow \Delta$ is obtained by (R1) from $\Gamma' \Rightarrow \Delta'$, where Γ' is a subset of Γ not containing $\sim A$ and Δ' is a subset of Δ. But in this case $\Gamma \Rightarrow \{A\} \cup \Delta$ equally follows by (R1) from $\Gamma' \Rightarrow \Delta'$, and we have no counterexample.

If $\sim A$ enters when $\Gamma \cup \{\sim A\} \Rightarrow \Delta$ is obtained by (R2*b*), the premiss must be $\Gamma \Rightarrow \{A\} \cup \Delta$ itself or $\Gamma \cup \{\sim A\} \Rightarrow \{A\} \cup \Delta$, and in the latter case, since the derivation of the premiss is too short to be a counterexample, there must exist a derivation of $\Gamma \Rightarrow \{A\} \cup \{A\} \cup \Delta$ or $\Gamma \Rightarrow \{A\} \cup \Delta$; so we have no counterexample.

The other case where $\sim A$ enters arises when $\sim A$ is of the form $\sim B(s)$ and the last lines of the derivation are

$$\Gamma \cup \{\sim B(t)\} \Rightarrow \Delta$$
$$\Gamma \cup \{s = t, \sim B(s)\} \Rightarrow \Delta$$

using (R8*b*).

to which may be added the step

$$\Gamma \cup \{s = t\} \Rightarrow \{B(s)\} \cup \Delta$$

which follows by (R8*a*), and again we have no counterexample.

14.16 Corollary. Any sequent derivable using (R0)–(R9) is in fact derivable using only (R0)–(R8).

Proof: Suppose there were a counterexample, that is, a derivation using (R0)–(R9) the last step $\Gamma \Rightarrow \Delta$ of which was *not* derivable using just (R0)–(R8). Among all such derivations, consider a derivation Σ that is as short as possible for a counterexample. $\Gamma \Rightarrow \Delta$ is not of the form $\{A\} \Rightarrow \{A\}$, since any sequent of that form can be derived in one step by (R0). So in Σ the sequent $\Gamma \Rightarrow \Delta$ is inferred by from one or more premisses appearing as earlier steps. Since the derivation down to any earlier step is too short to be a counterexample, for each premiss there is a derivation of it using just (R0)–(R8). If there is only one premiss, let Σ_0 be such a derivation of it. If there are more than one premiss, let Σ_0 be the result of *concatenating* such a derivation for each premiss, writing one after the other. In either case, Σ_0 is a derivation using only (R0)–(R8) that includes any and all premisses among its steps. Let Σ' be the derivation that results on adding $\Gamma \Rightarrow \Delta$ as one last step, inferred by the same rule as in Σ. If that rule was one of (R0)–(R8), we have a derivation of $\Gamma \Rightarrow \Delta$ using only (R0)–(R8). If the rule was (R9*a*), then Δ is of the form $\{A\} \cup \Delta'$, where we have a derivation of $\Gamma \cup \{\sim A\} \Rightarrow \Delta'$ using only (R0)–(R8). In that case, the inversion lemma tells us we have a derivation of $\Gamma \Rightarrow \Delta$, that is, of $\Gamma \Rightarrow \{A\} \cup \Delta'$, using only (R0)–(R8). Likewise if the rule was (R9*b*). So in any case, we have a derivation of $\Gamma \Rightarrow \Delta$ using only (R0)–(R8), and our original supposition that we had a counterexample has led to a contradiction, completing the proof.

14.17 Corollary. The proof procedure consisting of rules (R0)–(R8) is sound and complete.

Proof: Soundness is immediate from the soundness theorem for (R0)–(R9), since taking away rules cannot make a sound system unsound. Completeness follows from completeness for (R0)–(R9) together with the preceding corollary.

Instead of dropping (R9), one might consider adding the following.

(R10)
$$\frac{\begin{array}{l}\Gamma \Rightarrow \{(A \to B)\} \cup \Delta \\ \Gamma \Rightarrow \{A\} \cup \Delta\end{array}}{\Gamma \Rightarrow \{B\} \cup \Delta}.$$

14.18 Lemma (Cut elimination theorem). Using (R0)–(R9), if there are derivations of $\Gamma \Rightarrow \{(A \to B)\} \cup \Delta$ and of $\Gamma \Rightarrow \{A\} \cup \Delta$, then there is a derivation of $\Gamma \Rightarrow \{B\} \cup \Delta$.

14.19 Corollary. Any sequent derivable using (R0)–(R10) is in fact derivable using only (R0)–(R9).

14.20 Corollary. The proof procedure consisting of rules (R0)–(R10) is sound and complete.

Proofs: We begin with Corollary14.20. It is easily seen that rule (R10) is sound, so soundness for (R0)–(R10) follows from the soundness theorem for (R0)–(R9). Completeness for (R0)–(R10) follows from the completeness theorem for (R0)–(R9), since adding rules cannot make a complete system incomplete.

Now Corollary14.19 follows, since the same sequents are derivable in any two sound and complete proof procedures: by Corollary14.17 a sequent will be derivable using (R0)–(R10) if and only if it is secure, and by Theorems 14.1 and 14.2 it will be secure if and only if it is derivable using (R0)–(R9).

And now Lemma 14.18 follows also, since if there are derivations of $\Gamma \Rightarrow \{(A \rightarrow B)\} \cup \Delta$ and of $\Gamma \Rightarrow \{A\} \cup \Delta$ using (R0)–(R9), then there is certainly a derivation of $\Gamma \Rightarrow \{B\} \cup \Delta$ using (R0)–(R10) [namely, the one consisting simply of concatenating the two given derivations and adding a last line inferring $\Gamma \Rightarrow \{B\} \cup \Delta$ by (R10)], and by Corollary 14.19, this implies there must be a derivation of $\Gamma \Rightarrow \{B\} \cup \Delta$ using only (R0)–(R9).

Note the contrast between the immediately foregoing proof of the cut elimination lemma, Lemma 14.18, and the earlier proof of the inversion lemma, Lemma 14.15. The inversion proof is *constructive*: it actually contains implicit instructions for converting a derivation of $\Gamma \cup \{\sim A\} \Rightarrow \Delta$ into a derivation of $\Gamma \Rightarrow \{A\} \cup \Delta$. The cut elimination proof we have given is *nonconstructive*: it gives no hint how to *find* a derivation of $\Gamma \Rightarrow \{B\} \cup \Delta$ given derivations of $\Gamma \Rightarrow \{A\} \cup \Delta$ and $\Gamma \Rightarrow \{(A \rightarrow B)\} \cup \Delta$, though it promises us that such a derivation *exists*.

A constructive proof of the corollary is known, *Gentzen's proof*, but it is very much more complicated than the proof of the inversion lemma, and the result is that while the derivation of $\Gamma \Rightarrow \{A\} \cup \Delta$ obtained from the proof of the inversion lemma is about the same length as the given derivation of $\Gamma \cup \{\sim A\} \Rightarrow \Delta$, the derivation of $\Gamma \Rightarrow \{B\} \cup \Delta$ obtained from the constructive proof of the foregoing corollary may be astronomically longer than the given derivations of $\Gamma \Rightarrow \{(A \rightarrow B)\} \cup \Delta$ and $\Gamma \Rightarrow \{A\} \cup \Delta$ combined.

So much for dropping (R9) or adding (R10). A great deal more adding and dropping of rules could be done. If enough new rules are added, some of our original rules (R0)–(R8) could then be dropped, since the effect of them could be achieved using the new rules. If we allowed & and ∀ officially, we would want rules for them, and the addition of these rules might make it possible to drop some of the rules for ∨ and ∃, if indeed we did not choose to drop ∨ and ∃ altogether from our official language, treating them as abbreviations. Similarly for → and ↔.

In all the possible variations mentioned in the preceding paragraph, we were assuming that the basic objects would still be sequents $\Gamma \Rightarrow \Delta$. But variation is possible in this respect as well. It is possible, with the right selection of rules, to get by working *only* with sequents of form $\Gamma \Rightarrow \{D\}$ (in which case one would simply write $\Gamma \Rightarrow D$), making *deduction* the central notion. It is even possible to get by working only with sequents of form $\Gamma \Rightarrow \varnothing$ (in which case one would simply write Γ), making *refutation* the central notion. Indeed, it is even possible to get by working only with sequents of form $\varnothing \Rightarrow \{D\}$ (which in one would simply write D), making *demonstration* the central notion.

Just by way of illustration, the rules for a variant approach in which $\sim$ and $\to$ and $\forall$ and $=$ are the official logical operators, and in which one works only with sequents of form $\Gamma \Rightarrow D$, are listed in Table 14-5.

Table 14-5. *Rules of a variant sequent calculus*

(Q0)	$\overline{\Gamma \Rightarrow A}$	A in Γ
(Q1)	$\dfrac{\begin{array}{c}\Gamma \Rightarrow A \to B \\ \Gamma \Rightarrow A\end{array}}{\Gamma \Rightarrow B}$	
(Q2)	$\dfrac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow A \to B}$	
(Q3)	$\dfrac{\Gamma \Rightarrow \sim\sim A}{\Gamma \Rightarrow A}$	
(Q4)	$\dfrac{\begin{array}{c}\Gamma, A \Rightarrow B \\ \Gamma, A \Rightarrow \sim B\end{array}}{\Gamma \Rightarrow \sim A}$	
(Q5)	$\dfrac{\Gamma \Rightarrow \forall x A(x)}{\Gamma \Rightarrow A(t)}$	
(Q6)	$\dfrac{\Gamma \Rightarrow A(c)}{\Gamma \Rightarrow \forall x A(x)}$	c not in Γ or $A(x)$
(Q7)	$\dfrac{\begin{array}{c}\Gamma \Rightarrow s = t \\ \Gamma \Rightarrow A(s)\end{array}}{\Gamma \Rightarrow A(t)}$	
(Q8)	$\overline{\Gamma \Rightarrow t = t}$	

This variation can be proved sound and complete in the sense that a sequent $\Gamma \Rightarrow D$ will be obtainable by these rules if and only if D is a consequence of Γ. We give one sample deduction to give some idea how the rules work.

14.21 Example. A deduction.

(1) $\sim A \to \sim B,\ B,\ \sim A \Rightarrow \sim A \to \sim B$ (Q0), (i)

(2) $\sim A \to \sim B,\ B,\ \sim A \Rightarrow \sim A$ (Q0), (iii)

(3) $\sim A \to \sim B,\ B,\ \sim A \Rightarrow \sim B$ (Q1), (1), (2)

(4) $\sim A \to \sim B,\ B,\ \sim A \Rightarrow B$ (Q0), (ii)

(5) $\sim A \to \sim B,\ B \Rightarrow \sim\sim A$ (Q4), (3), (4)

(6) $\sim A \to \sim B,\ B \Rightarrow A$ (Q3), (5)

(7) $\sim A \to \sim B \Rightarrow B \to A$ (Q2), (6)

The lowercase Roman numerals (i)–(iii) associated with (Q0) indicate whether it is the first, second, or third sentence in $\Gamma = \{\sim A \to \sim B,\ B,\ \sim A\}$ that is playing the role of A in the rule (Q0).

In addition to such substantive variations as we have been discussing, considerable variations in *style* are possible, and in particular in typographical layout. For instance, if one opens an introductory textbook, one may well encounter something like what appears in Figure 14-1.

(i)	$\sim A \to \sim B$			
(ii)		B		
(iii)			$\sim A$	
(1)			$\sim A \to \sim B$	(Q0), (i)
(2)			$\sim A$	(Q0), (iii)
(3)			$\sim B$	(Q1), (1), (2)
(4)			B	(Q0), (ii)
(5)		$\sim\sim A$		(Q4), (3), (4)
(6)		A		(Q3), (5)
(7)	$B \to A$			(Q2), (6)

Figure 14-1. A 'natural deduction'.

What appears in Figure 14-1 is really the same as what appears in Example 14.21, differently displayed. The form of display adopted in this book, as illustrated in Example 14.21, is designed for convenience when engaged in theoretical writing *about* deductions. But when engaged in the practical writing *of* deductions, as in introductory texts, the form of display in Figure 14-1 is more convenient, because it involves less rewriting of the same formula over and over again. In lines (1)–(7) in Figure 14-1, one only writes the sentence D on the right of the sequent $\Gamma \Rightarrow D$ that occurs at the corresponding line in Example 14.21. Which of the sentences (i), (ii), (iii) occur in the set Γ on the left of that sequent is indicated by the *spatial position* where D is written: if it is written in the third column, all of (i)–(iii) appear; if in the second, only (i) and (ii) appear; if in the first, only (i). Colloquially one sometimes speaks of deducing a conclusion D 'under' certain hypotheses Γ, but in the form of display illustrated in Figure 14-1, the spatial metaphor is taken quite literally.

It would take us too far afield to enter into a detailed description of the conventions of this form of display, which in any case can be found expounded in many introductory texts. The pair of examples given should suffice to make our only real point here: that what is substantively the same kind of procedure can be set forth in very different styles, and indeed appropriately so, given the different purposes of introductory texts and of more theoretical books like this one. Despite the diversity of approaches possible, the aim of any approach is to set up a system of rules with the properties that if D is deducible from Γ, then D is a consequence of Γ (*soundness*), and that if D is a consequence of Γ, then D is formally deducible from Γ (*completeness*). Clearly, *all systems of rules that achieve these aims will be equivalent to each other* in the sense that D will be deducible from Γ in the one system if and only if D is deducible from Γ in the other system. Except for one optional section at the end of the next chapter, there will be no further mention of the details of our particular proof procedure in the rest of this book.

A word may now be said about the relationship between *any* formal notion, whether ours or a variant, of deduction of a sentence from a set of sentences, and the notion in unformalized mathematics of a proof of a theorem from a set of axioms. For in later chapters we are going to be establishing results about the scope and limits of formal deducibility whose interest largely depends on their having something to do with proof in a more ordinary sense (just as results about the scope and limits of computability in one or another formal sense discussed in other chapters depend for their interest on their having something to do with computation in a more ordinary sense).

We have already mentioned towards the end of Chapter 10 that theorems and axioms in ordinary mathematics can virtually always be expressed as sentences of a formal first-order language. Suppose they are so expressed. Then if there is a deduction in the logician's formal sense of the theorem from the axioms, there will be a proof in the mathematician's ordinary sense, because, as indicated earlier, each formal rule of inference in the definition of deduction corresponds to some ordinary mode of argument as used in mathematics and elsewhere. It is the converse assertion, that if there is a proof in the ordinary sense, then there will be a deduction in our very restrictive format, that may well seem more problematic. This converse assertion is sometimes called *Hilbert's thesis*.

As the notion of 'proof in the ordinary sense' is an intuitive, not a rigorously defined one, there cannot be a rigorous proof of Hilbert's thesis. Before the completeness theorem was discovered, a good deal of evidence of two kinds had already been obtained for the thesis. On the one hand, logicians produced vast compendia of formalizations of ordinary proofs. On the other hand, various independently proposed systems of formal deducibility, each intended to capture formally the ordinary notion of provability, had been proved equivalent to each other by directly showing how to convert formal deductions in one format into formal deductions in another format; and such equivalence of proposals originally advanced independently of each other, while it does not amount to a rigorous proof that either has succeeded in capturing the ordinary notion of provability, is surely important evidence in favor of both.

The completeness theorem, however, makes possible a much more decisive argument in favor of Hilbert's thesis. The argument runs as follows. Suppose there is a proof in the ordinary mathematical sense of some theorem from some axioms. As part-time orthodox mathematicians ourselves, we presume ordinary mathematical methods of proof are sound, and if so, then the existence of an ordinary mathematical proof means that the theorem really is a consequence of the axioms. But if the theorem is a consequence of the axioms, then the completeness theorem tells us that, in agreement with Hilbert's thesis, there will be a formal deduction of the theorem from the axioms. And when in later chapters we show that there can be *no* formal deduction in certain circumstances, it will follow that there can be no ordinary proof, either.

Problems

14.1 Show that:

(**a**) Γ secures Δ if and only if $\Gamma \cup {\sim}\Delta$ is unsatisfiable.

(**b**) Γ secures Δ if and only if some finite subset of Γ secures some finite subset of Δ.

14.2 Explain why the problems following this one become more or less trivial if one is allowed to appeal to the soundness and completeness theorems.

Unless otherwise specified, 'derivable' is to mean 'derivable using (R0)–(R8)'. *All proofs should be constructive, not appealing to the soundness and completeness theorems.*

14.3 Show that if $\Gamma, A, B \Rightarrow \Delta$ is derivable, then $\Gamma, A \,\&\, B \Rightarrow \Delta$ is derivable.

14.4 Show that if $\Gamma \Rightarrow A, \Delta$ and $\Gamma \Rightarrow B, \Delta$ are derivable, then $\Gamma \Rightarrow A \,\&\, B, \Delta$ is derivable.

14.5 Show that if $\Gamma \Rightarrow A(c), \Delta$ is derivable, then $\Gamma \Rightarrow \forall x A(x), \Delta$ is derivable, provided c does not appear in Γ, Δ, or $A(x)$.

14.6 Show that if $\Gamma, A(t) \Rightarrow \Delta$ is derivable, then $\Gamma, \forall x A(x) \Rightarrow \Delta$ is derivable.

14.7 Show that $\forall x Fx \,\&\, \forall xGx$ is deducible from $\forall x(Fx \,\&\, Gx)$.

14.8 Show that $\forall x\ (Fx \,\&\, Gx)$ is deducible from $\forall xFx \,\&\, \forall xGx$.

14.9 Show that the *transitivity of identity*, $\forall x\forall y\forall z(x = y \,\&\, y = z \rightarrow x = z)$ is demonstrable.

14.10 Show that if $\Gamma, A(s) \Rightarrow \Delta$ is derivable, then $\Gamma, s = t, A(t) \Rightarrow \Delta$ is derivable.

14.11 Prove the following (*left*) *inversion lemma for disjunction*: if there is a derivation of $\Gamma \Rightarrow \{(A \vee B)\} \cup \Delta$ using rules (R0)–(R8), then there is such a derivation of $\Gamma \Rightarrow \{A, B\} \cup \Delta$.

14.12 Prove the following (*right*) *inversion lemma for disjunction*: if there is a derivation of $\Gamma \cup \{(A \vee B)\} \Rightarrow \Delta$, then there is a derivation of $\Gamma \cup \{A\} \Rightarrow \Delta$, and there is a derivation of $\Gamma \cup \{B\} \Rightarrow \Delta$.

14.13 Consider adding one or the other of the following rules to (R0)–(R8):

(R11)
$$\frac{\begin{array}{r}\Gamma \cup \{A\} \Rightarrow \Delta \\ \Gamma \Rightarrow \{A\} \cup \Delta\end{array}}{\Gamma \Rightarrow \Delta}.$$

(R12)
$$\frac{\Gamma \cup \{(A \vee {\sim}A)\} \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}.$$

Show that a sequent is derivable on adding (R11) if and only if it is derivable on adding (R12).

15

Arithmetization

In this chapter we begin to bring together our work on logic from the past few chapters with our work on computability from earlier chapters (specifically, our work on recursive functions from Chapters 6 and 7). In section 15.1 we show how we can 'talk about' such syntactic notions as those of sentence and deduction in terms of recursive functions, and draw among others the conclusion that, once code numbers are assigned to sentences in a reasonable way, the set of valid sentences is semirecursive. Some proofs are deferred to sections 15.2 and 15.3. The proofs consist entirely of showing that certain effectively computable functions are recursive. Thus what is being done in the two sections mentioned is to present still more evidence, beyond that accumulated in earlier chapters, in favor of Church's thesis that all *effectively computable functions are recursive. Readers who feel they have seen enough evidence for Church's thesis for the moment may regard these sections as optional.*

15.1 Arithmetization of Syntax

A necessary preliminary to applying our work on computability, which pertained to functions on natural numbers, to logic, where the objects of study are expressions of a formal language, is to code expressions by numbers. Such a coding of expressions is called a *Gödel numbering*. One can then go on to code finite sequences of expressions and still more complicated objects.

A set of symbols, or expressions, or more complicated objects may be called recursive in a transferred or derivative sense if and only if the set of code numbers of elements of the set in question is recursive. Similarly for functions. Officially, a language is just a set of nonlogical symbols, so a language may be called recursive if and only if the set of code numbers of symbols in the language is recursive. In what follows we tacitly assume throughout that the languages we are dealing with are recursive: in practice we are going to be almost exclusively concerned with *finite* languages, which are trivially so.

There are many reasonable ways to code finite sequences, and it does not really matter which one we choose. Almost all that matters is that, for any reasonable choice, the following *concatenation* function will be recursive: $s * t =$ the code number for the sequence consisting of the sequence with code number s followed by the sequence with code number t. This is all that is needed for the proof of the next proposition, in which, as elsewhere in this section, ‘recursive’ could actually be strengthened to ‘primitive recursive’.

So that the reader may have something definite in mind, let us offer one example of a coding scheme. It begins by assigning code numbers to symbols as in Table 15-1.

Table 15-1. *Gödel numbers of symbols (first scheme)*

Symbol	(	$\sim$	$\exists$	$=$	v_0	A^0_0	A^1_0	A^2_0	$\ldots$	f^0_0	f^1_0	f^2_0	$\ldots$
	)	$\vee$			v_1	A^0_1	A^1_1	A^2_1	$\ldots$	f^0_1	f^1_1	f^2_1	$\ldots$
	,				v_2	A^0_2	A^1_2	A^2_2	$\ldots$	f^0_2	f^1_2	f^2_2	$\ldots$
					$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	
Code	1	2	3	4	5	6	68	688	$\ldots$	7	78	788	$\ldots$
	19	29			59	69	689	6889	$\ldots$	79	789	7889	$\ldots$
	199				599	699	6899	68899	$\ldots$	799	7899	78899	$\ldots$
					$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	

Thus for the language of arithmetic $<$ or A^2_0 has code number 688, $\mathbf{0}$ or f^0_0 has code number 7′, or f^1_0 has code number 78, $+$ or f^2_0 has code number 788, and $\cdot$ or f^2_1 has code number 7889. We then extend the code numbering to all finite sequences of symbols. The principle is that if the expression E has code number e and the expression D has code number d, then the expression ED obtained by concatenating them is to have the code number whose decimal numeral is obtained by concatenating the decimal numeral for e and the decimal numeral for d. Thus $(\mathbf{0} = \mathbf{0} \vee \sim\mathbf{0} = \mathbf{0})$, the sequence of symbols with code numbers

$$1, 7, 4, 7, 29, 2, 7, 4, 7, 19$$

has code number 174 729 274 719.

In general the code number for the concatenation of the expressions with code numbers e and d can be obtained from e and d as $e * d = e \cdot 10^{\lg(d,10)+1} + d$, where lg is the logarithm function of Example 7.11. For $\lg(d, 10) + 1$ will be the least power z such that $d < 10^z$, or in other words, the number of digits in the decimal numeral for d, and thus the decimal numeral for $e \cdot 10^{\lg(d,10)+1}$ will be that for e followed by as many 0 s as there are digits in that for d, and the decimal numeral for $e \cdot 10^{\lg(d,10)+1} + d$ will be that for e followed by that for d.

15.1 Proposition. The logical operations of negation, disjunction, existential quantification, substitution of a term for free occurrences of a variable, and so on, are recursive.

Proof: Let n be the code number for the tilde, and let neg be the recursive function defined by letting $\text{neg}(x) = n * x$. Then if x is the code number for a formula, $\text{neg}(x)$ will be the code number for its negation. (We do not care what the function does with numbers that are *not* code numbers of formulas.) This is what is meant by saying that the operation of negation is recursive. Similarly, if l and d and r are the code numbers for the left parenthesis and wedge and right parenthesis, $\text{disj}(x, y) = l * x * d * y * r$ will be the code number for the disjunction of the formulas coded by

x and y. If e is the code number for the backwards E, then exquant(v, x) = $e * v * x$ will be the code number for the existential quantification with respect to the variable with code number v of the formula with code number x. And similarly for as many other logical operations as one cares to consider. For instance, if officially the conjunction (X & Y) is an abbreviation for $\sim(\sim X \vee \sim Y)$, the conjunction function is then the composition conj (x, y) = neg(disj(neg (x), neg(y))). The case of substitution is more complicated, but as we have no immediate need for this operation, we defer the proof.

Among sets of expressions, the most important for us will be simply the sets of formulas and of sentences. Among more complicated objects, the only important ones for us will be deductions, on whatever reasonable proof procedure one prefers, whether ours from the preceding chapter, or some other from some introductory textbook. Now intuitively, one can effectively decide whether or not a given sequence of symbols is a formula, and if so, whether it is a sentence. Likewise, as we mentioned when introducing our own proof procedure, one can effectively decide whether a given object D is a deduction of a given sentence from a given finite set of sentences Γ_0. If Γ is an *infinite* set of sentences, then a deduction of D from Γ is simply a deduction of D from some finite subset of Γ_0, and therefore, *so long as one can effectively decide whether a given sentence C belongs to* Γ, and hence can effectively decide whether a given finite set Γ_0 is a subset of Γ, one can also effectively decide whether a given object is a deduction of D from Γ. Church's thesis then implies the following.

15.2 Proposition. The sets of formulas and of sentences are recursive.

15.3 Proposition. If Γ is a recursive set of sentences, then the relation 'Σ is a deduction of sentence D from Γ' is recursive.

Collectively, Propositions 15.1–15.3 (and their various attendant lemmas and corollaries) are referred to by the imposing title at the head of this section. Before concerning ourselves with the proofs of these propositions, let us note a couple of implications.

15.4 Corollary. The set of sentences deducible from a given recursive set of sentences is semirecursive.

Proof: What is meant is that the set of *code numbers* of sentences deducible from a given recursive set is semirecursive. To prove this we apply Proposition 15.3. What is meant by the statement of that proposition is that if Γ is recursive, then the relation

$$\begin{aligned} Rsd \leftrightarrow\ & d \text{ is the code number of a sentence and} \\ & s \text{ is the code number of a deduction of it from } \Gamma \end{aligned}$$

is recursive. And then the set S of code numbers of sentences deducible from Γ, being given by $Sd \leftrightarrow \exists s\ Rsd$, will be semirecursive.

15.5 Corollary (Gödel completeness theorem, abstract form). The set of valid sentences is semirecursive.

Proof: By the Gödel completeness theorem, the set of valid sentences is the same as the set of demonstrable sentences, that is, as the set of sentences deducible from $\Gamma = \varnothing$. Since the empty set $\varnothing$ is certainly recursive, it follows from the preceding corollary that the set of valid sentences is semirecursive.

The preceding corollary states as much of the content of the Gödel completeness theorem as it is possible to state without mentioning any particular proof procedure. The next corollary is more technical, but will be useful later.

15.6 Corollary. Let Γ be a recursive set of sentences in the language of arithmetic, and $D(x)$ a formula of that language. Then:

(a) The set of natural numbers n such that $D(\mathbf{n})$ is deducible from Γ is semirecursive.
(b) The set of natural numbers n such that $\sim D(\mathbf{n})$ is deducible from Γ is semirecursive.
(c) If for every n either $D(\mathbf{n})$ or $\sim D(\mathbf{n})$ is deducible from Γ, then the set of n such that $D(\mathbf{n})$ is deducible from Γ is recursive.

Proof: For (a), we actually show that the set R of pairs (d, n) such that d is the code number for a formula $D(x)$ and $D(\mathbf{n})$ is deducible from Γ is semirecursive. It immediately follows that for any one fixed $D(x)$, with code number d, the set of n such that $D(\mathbf{n})$ is deducible from Γ will be semirecursive, since it will simply be the set of n such that Rdn. To avoid the need to consider substituting a term for the free occurrences of a variable (the one operation mentioned in Proposition 15.1 the proof of whose recursiveness we deferred), first note that for any n, $D(\mathbf{n})$ and $\exists x(x = \mathbf{n} \,\&\, D(x))$ are logically equivalent, and one will be a consequence of, or equivalently, deducible from, Γ if and only if the other is. Now note that the function taking a number n to the code number num(n) for the numeral $\mathbf{n}$ is (primitive) recursive, for recalling that officially s' is $'(s)$ we have

$$\mathrm{num}(0) = z \qquad \mathrm{num}(n') = a * b * \mathrm{num}(n) * c$$

where z is the code number for the cipher $\mathbf{0}$ and a, b, and c are the code numbers for the accent and the left and right parentheses. The function f taking the code number d for a formula $D(x)$ and a number n to the code number for $\exists x(x = \mathbf{n} \,\&\, D(x))$ is recursive in consequence of Proposition 15.1, since we have

$$f(d, n) = \mathrm{exquant}(v, \mathrm{conj}(i\,^*\, b\,^*\, v\,^*\, k\,^*\, \mathrm{num}(n)\,^*\, c),\ d)$$

where v is the code number for the variable, i for the equals sign, k for the comma. The set S of code numbers of sentences that are deducible from Γ is semirecursive by Corollary 15.4. The set R of pairs is then given by

$$R(d, n) \leftrightarrow S(f(d, n)).$$

In other words, R is obtained from the semirecursive set S by substituting the recursive total function f, which implies that R is itself semirecursive.

As for (b), we actually show that the set Q of pairs (d, n) such that d is the code number for a formula $D(x)$ and $\sim D(\mathbf{n})$ is deducible from Γ is semirecursive. Indeed, with R as in part (a) we have

$$Q(d, n) \leftrightarrow R(\mathrm{neg}(d), n).$$

So Q is obtained from the semirecursive R by substitution of the recursive total function neg, which implies that Q is itself semirecursive.

Obviously there is nothing special about negation as opposed to other logical constructions here. For instance, in the language of arithmetic, we could consider the operation taking $D(x)$ not to $\sim D(x)$ but to, say,

$$D(x) \,\&\, \sim\exists y < x \; D(y)$$

and since the relevant function on code numbers would still, like neg, be recursive in consequence of Proposition 15.1, so the set of pairs (d, n) such that

$$D(\mathbf{n}) \,\&\, \sim\exists y < \mathbf{n} D(y)$$

is deducible from Γ is also semirecursive. We are not going to stop, however, to try to find the most general formulation of the corollary.

As for (c), if for any n both $D(\mathbf{n})$ and $\sim D(\mathbf{n})$ are deducible from Γ, then *every* formula is deducible from Γ, and the set of n such that $D(\mathbf{n})$ is deducible from Γ is simply the set of *all* natural numbers, which is certainly recursive. Otherwise, on the assumption that for every n either $D(\mathbf{n})$ or $\sim D(\mathbf{n})$ is deducible from Γ, the set of n for which $D(\mathbf{n})$ is deducible and the set of n for which $\sim D(\mathbf{n})$ is deducible are complements of each other. Then (c) follows from (a) and (b) by Kleene's theorem (Proposition 7.16).

There is one more corollary worth setting down, but before stating it we introduce some traditional terminology. We use 'Γ *proves* D', written $\Gamma \vdash D$ or $\vdash_\Gamma D$, interchangeably with 'D is deducible from Γ'. The sentences proved by Γ we call the *theorems* of Γ. We conscript the word *theory* to mean a set of sentences *that contains all the sentences of its language that are provable from it.* Thus the theorems of a theory T are just the sentences in T, and $\vdash_T B$ and $B \in T$ are two ways of writing the same thing.

Note that we do not require that any subset of a theory T be singled out as 'axioms'. If there *is* a recursive set Γ of sentences such that T consists of all and only the sentences provable from Γ, we say T is *axiomatizable*. If the set Γ is finite, we say T is *finitely axiomatizable*. We have already defined a set Γ of sentences to be *complete* if for every sentence B of its language, either B or $\sim B$ is a consequence of Γ, or equivalently, is provable from Γ. Note that for a *theory* T, T is complete if and only if for every sentence B of its language, either B or $\sim B$ is in T. Similarly, a set Γ is *consistent* if not every sentence is a consequence of Γ, so a *theory* T is consistent if not every sentence of its language is in T. A set Γ of sentences is *decidable* if the set of sentences of its language that are consequences of Γ, or equivalently, are proved by Γ, is recursive. Note that for a *theory* T, T is decidable if and only if T is recursive. This terminology is used in stating our next result.

15.7 Corollary. Let T be an axiomatizable theory. If T is complete, then T is decidable.

Proof: Throughout, 'sentence' will mean 'sentence of the language of T'. The assumption that T is an axiomatizable theory means that T is the set of sentences provable from some recursive set of sentences Γ. We write T^* for the set of *code numbers of* theorems of T. By Corollary 15.4, T^* is semirecursive. To show that T is decidable we need to show that T^* is in fact recursive. By Proposition 15.2, T^* will be so if it is simply the set of *all* code numbers of sentences, so let us consider the case where this is not so, that is, where not every sentence is a theorem of T. Since every sentence *would be* a theorem of T if for any sentence D it happened that both D and $\sim D$ were theorems of T, for no sentence D can this happen. On the other hand, the hypothesis that T is complete means that for every sentence D, at least one of D and $\sim D$ is a theorem of T. It follows that D is not a theorem of T if and only if $\sim D$ *is* a theorem of T. Hence the complement of T^* is the union of the set X of those numbers n that are not code numbers of sentences at all, and the set Y of code numbers of sentences whose negations are theorems of T, or in other words, the set of n such that neg(n) is in T^*. X is recursive by Proposition 15.2. Y is semirecursive, since it is obtainable by substituting the recursive function neg in the semirecursive set T^*. So the complement of T^* is semirecursive, as was T^* itself. That T^* is recursive follows by Kleene's theorem (Proposition 7.16).

It 'only' remains to prove Propositions 15.2 and 15.3. In proving them we are once again going to be presenting evidence for Church's thesis: we are one more time going to be showing that certain sets and functions that must be recursive if Church's thesis is correct are indeed recursive. Many readers may well feel that by this point they have seen enough evidence, and such readers may be prepared simply to take Church's thesis on trust in future. There is much to be said for such an attitude, especially since giving the proofs of these propositions requires going into details about the Gödel numbering, the scheme of coding sequences, and the like, that we have so far largely avoided; and it is very easy to get bogged down in such details and lose sight of larger themes. (There is serious potential for a woods–trees problem, so to speak.) Readers who share the attitude described are therefore welcome to postpone *sine die* reading the proofs that fill the rest of this chapter. Section 15.2 concerns (the deferred clause of Proposition 15.1 as well as) Proposition 15.2, while section 15.3 concerns Proposition 15.3.

15.2* Gödel Numbers

We next want to indicate the proof of Proposition 15.2 (also indicating, less fully, the proof of the one deferred clause of Proposition 15.1, on the operation of substituting a term for the free occurrences of a variable in a formula). The Gödel numbering we gave by way of illustration near the beginning of this chapter is not, in fact, an especially convenient one to work with here, mainly because it is not so easy to show that such functions as the one that gives the the length (that is, number of symbols) in the expression with a given code number are primitive recursive. An alternative way of assigning code numbers to expressions begins by assigning code numbers to symbols as in Table 15-2.

Table 15-2. *Gödel numbers of symbols (second scheme)*

Symbol	(	)	,	$\sim$	$\vee$	$\exists$	$=$	v_i	A_i^n	f_i^n
Code	1	3	5	7	9	11	13	$2 \cdot 5^i$	$2^2 \cdot 3^n \cdot 5^i$	$2^3 \cdot 3^n \cdot 5^i$

Thus for the language of arithmetic $<$ or A_0^2 has code number $2^2 \cdot 3^2 \cdot 5^0 = 4 \cdot 9 = 36$, $\mathbf{0}$ or f_0^0 has code number $2^3 \cdot 3^0 \cdot 5^0 = 8$, $'$ or f_0^1 has code number $2^3 \cdot 3^1 \cdot 5^0 = 8 \cdot 3 = 24$, $+$ or f_0^2 has code number $2^3 \cdot 3^2 \cdot 5^0 = 8 \cdot 9 = 72$, and similarly $\cdot$ has code number 360. We then extend the code numbering to all finite sequences of symbols by assigning to an expression E consisting of a sequence of symbols $S_1 S_2 \cdots S_n$ the code number # (E) for the sequence $(|S_1|, |S_2|, \ldots, |S_n|)$ according to the scheme for coding finite sequences of numbers by single numbers based on prime decomposition. [In contrast to the earlier scheme, we need to distinguish, in the case of the expression consisting of a single symbol S, the code number # (S) of S *qua* expression from the code number $|S|$ of S *qua* symbol. In general the code number for a single-term sequence (n) is $2 \cdot 3^n$, so we get $\#(S) = 2 \cdot 3^{|S|}$.] Thus the code number for the sentence we have been writing $\mathbf{0} = \mathbf{0}$, which is officially $=(\mathbf{0}, \mathbf{0})$, is that for $(13, 1, 36, 5, 36, 3)$, which is $2^6 \cdot 3^{13} \cdot 5 \cdot 7^{36} \cdot 11^5 \cdot 13^{36} \cdot 17^3$. This is a number of 89 digits. Fortunately, our concern will only be with what kinds of calculations could in principle be performed with such large numbers, not with performing such calculations in practice.

The calculation of the length lh(e) of the expression with code number e is especially simple on this scheme, since $\text{lh}(e) = \text{lo}(e, 2)$, where lo is the logarithm function in Example 7.11, or in other words, the exponent on the prime 2 in the prime decomposition of e. What are not so easy to express as primitive recursive functions on this coding scheme are such functions as the one that gives the code number for the concatenation of the expressions with two given code numbers. But while such functions may not have been so easy to prove primitive recursive, they *have* been proved to be so in Chapter 7. We know from our work there that in addition to the concatenation function $*$, several further *cryptographic* or code-related functions are primitive recursive. Writing $\#(\sigma)$ for the code number of sequence σ, and $\S(s)$ for the sequence with code number s, we list these functions in Table 15-3.

Table 15-3. *Cryptographic functions*

lh (s)	= the length of $\S(s)$
ent(s, i)	= the ith entry of $\S(s)$
last(s)	= the last entry of $\S(s)$
ext(s, a)	= #($\S(s)$ with a added at the end)
pre(a, s)	= #($\S(s)$ with a added at the beginning)
sub(s, c, d)	= #($\S(s)$ with c replaced by d throughout)

More complicated objects, such as finite sequences or finite sets of expressions, can also be assigned code numbers. A code number for a finite sequence of expressions is simply a code number for a finite sequence of natural numbers, whose entries are themselves in turn code numbers for expressions. As a code number for a finite *set* of expressions, we may take the code number for the finite sequence of expressions that list the elements of the set (without repetitions) *in order of increasing code number*.

This means that a code number of a finite set of expressions will be a code number for a finite sequence of expressions *whose entries are increasing*, with later entries larger than earlier ones. A virtue of this coding is that such relations as 'the expression with code number i belongs to the set with code number s' and 'the set with code number t is a subset of the set with code number s' will all be simply definable in terms of the cryptographic functions, and hence recursive. (The first amounts to 'i is an entry of the sequence coded by s', and the second amounts to 'every entry of the sequence coded by t is an entry of the sequence coded by s'.) Similarly the coding can be extended to finite sequences or finite sets of finite sequences or finite sets, and so on.

Towards proving Proposition 15.2, the first thing to note is that one- and two-place relations like those given by 'a is the code number of a predicate' and 'a is the code number of an n-place predicate' are primitive recursive. For the former is equivalent to the existence of n and i such that $a = 2^2 \cdot 3^n \cdot 5^i$, and the latter is equivalent to the existence of i such that $a = 2^2 \cdot 3^n \cdot 5^i$. The function f given by $f(n, i) = 2^2 \cdot 3^n \cdot 5^i$ is primitive recursive, being a composition of exponentiation, multiplication, and the constant functions with values 2^2, 3, and 5. So the relation '$a = 2^2 \cdot 3^n \cdot 5^i$' is primitive recursive, being the graph relation '$a = f(n, i)$'. The two relations of interest are obtained from the relation '$a = 2^2 \cdot 3^n \cdot 5^i$' by existential quantification, and in each case the quantifiers can be taken to be *bounded*, since if $a = 2^2 \cdot 3^n \cdot 5^i$, then certainly n and i are less than a. So the first condition amounts to $\exists n < a\ \exists i < a(a = 2^2 \cdot 3^n \cdot 5^i)$ and the second to $\exists i < a(a = 2^2 \cdot 3^n \cdot 5^i)$.

Similar remarks apply to 'a codes a variable', 'a codes a function symbol', and 'a codes a constant (that is, a zero-place function symbol)', 'a codes an n-place function symbol', and 'a codes an atomic term (that is, a variable or constant)'. These all give primitive recursive relations. If we are interested only in formulas and sentences of some language L less than the full language containing all nonlogical symbols, we must add clauses 'and a is in L' to our various definitions of the items just listed. So long as L is still primitive recursive, and in particular if L is finite, the relations just listed will still be primitive recursive. (If L is only recursive and not primitive recursive, we have to change 'primitive recursive' to 'recursive' both here and below.)

Considering only the case without identity and function symbols, the relation given by 's codes an atomic formula' is also primitive recursive, being obtainable by simple operations (namely, substitution, conjunction, and bounded universal quantifications) from the relations mentioned in the preceding paragraph and the graph relations of the primitive recursive functions of some of the cryptographic functions listed earlier. Specifically, s codes an atomic formula if and only if there is an n less than lh(s) such that the following holds:

lh $(s) = 2n + 2$, and

ent $(s, 0)$ is the code number for an n-place predicate, and

ent $(s, 1) = 1$ (the code number for a left parenthesis), and

for every i with $1 < i <$ lh $(s) - 1$:

if i is odd then ent$(s, i) = 5$ (the code number for a comma), and

if i is even then ent(s, i) is the code number for an atomic term, and

last $(s) = 3$ (the code number for a right parenthesis).

Now s is the code number of a formula S if and only if there is some r that is the code number for a formation sequence for S. In general, the relation given by 'r is the code number of a formation sequence for a formula with code number s' is primitive recursive, since this relation holds if and only if the following does:

```
For all $j < \text{lh}(r)$ either:
    ent $(r, j)$ is the code number for an atomic sentence, or
    for some $k < j$,
        ent $(r, j) = \text{neg}(\text{ent}\,(r, k))$, or
    for some $k_1 < j$and some $k_2 < j$,
        ent $(r, j) = \text{disj}(\text{ent}\,(r, k_1),\ \text{ent}\,(r, k_2))$, or
    for some $k < j$ and some $i <$ ent $(r, j)$,
    ent $(r, j) =$ exquant $(2 \cdot 5^i,$ ent $(r, k))$
and last $(r) = s$.
```

Here neg, disj, and exquant are as in the proof of Proposition 15.1.

We can give a rough upper bound on the code number for a formation sequence, since we know (from the problems at the end of Chapter 9) that if S is a formula—that is, if S has any formation sequence at all—then S has a formation sequence in which every line is a substring of S, and the number of lines is less than the length of S. Thus, if there is any formation sequence at all for s, letting $n = \text{lh}(s)$, there will be a formation sequence for s of length no greater than n with each entry of size no greater than s. The code number for such a formation sequence will therefore be less than the code number for a sequence of length n all of whose entries are s, which would be $2^n \cdot 3^s \cdot \cdots \cdot \pi(n)^s$, where $\pi(n)$ is the nth prime, and this is less that $\pi(n)^{s(n+1)}$. So there is a primitive recursive function g, namely the one given by $g(x) = \pi(\text{lh}(x))^{x[\text{lh}(x)+1]}$, such that if s is the code number for a formula at all, then there will be an $r < g(s)$ such that r is a code number for a formation sequence for that formula. In other words, the relation given by 's is the code number for a formula' is obtainable by bounded quantification from a relation we showed in the preceding paragraph to be primitive recursive: $\exists r < g(s)$ (r codes a formation sequence for s). Thus the relation 's is the code number for a formula' is itself primitive recursive.

In order to define sentencehood, we need to be able to check which occurrences of variables in a formula are bound and which free. This is also what is needed in order to define the one operation in Lemma 15.1 whose proof we deferred, substitution of a term for the *free* occurrences of a variable in a formula. It is not the substitution itself that is the problem here, so much as recognizing which occurrences of the variable are to be substituted for and which not. The relation 's codes a formula and the eth symbol therein is a free occurrence of the dth variable' holds if and only if

```
$s$ codes a formula and ent $(s, e) = 2 \cdot 5^d$ and
for no $t, u, v, w < s$ is it the case that
    $s = t * v * w$ and lh $(t) < e$ and $e < \text{lh}(t) + \text{lh}(v)$ and
    $u$ codes a formula and $v =$ exquant $(2 \cdot 5^d, u)$.
```

For the first clause says that s codes a formula and the eth symbol therein is the dth variable, while the second clause says that the eth symbol does not fall within any subsequence v of the formula that is itself a formula beginning with a quantification of the dth variable. This relation is primitive recursive. Since the relation 's codes a sentence' is then simply

> s codes a formula and
>
> for no $d, e < s$ is the eth symbol therein a free occurrence of the dth variable

it is primitive recursive, too, as asserted.

So much for the proof in the case where identity and function symbols are absent. If identity is present, but not function symbols, the definition of atomic formula will be the disjunction of the clause above covering atomic formulas involving a nonlogical predicate with a second, similar but simpler, clause covering atomic formulas involving the logical predicate of identity. If function symbols are present, it will be necessary to give a preliminary definitions of *term formation sequence* and *term*. The definition for term formation sequence will have much the same gross form as the definition above of formation sequence; the definition for term will be obtained from it by a bounded existential quantification. We suppress the details.

15.3* More Gödel Numbers

We indicate the proof of Proposition 15.3, for the proof procedure used in the preceding chapter, only in gross outline. Something similar can be done for any reasonable proof procedure, though the details will be different.

We have already indicated how sets of sentences are to be coded: s is a code for a set of sentences if and only if s is a code for a sequence and for all $i < \mathrm{lh}(s)$, $\mathrm{ent}(s, i)$ is a code for a sentence, and in addition for all $j < i$, $\mathrm{ent}\,(s, j) < \mathrm{ent}(s, i)$. It follows that the set of such codes is primitive recursive. A derivation, on the approach we took in the last chapter, is a sequence of sequents $\Gamma_1 \Rightarrow \Delta_1$, $\Gamma_2 \Rightarrow \Delta_2$, and so on, subject to certain conditions. Leaving aside the conditions for the moment, a sequence of sequents is most conveniently coded by a code for $(c_1, d_1, c_2, d_2, \ldots)$, where c_i codes Γ_i and d_i codes Δ_i. The set of such codes is again primitive recursive. The sequence of sequents coded by the code for $(c_1, d_1, \ldots, c_n, d_n)$ will be a deduction of sentence D from set Γ if and only if: first, the sequence of sequents coded is a derivation; and second, c_n codes a sequences whose entries are all codes for sentences in Γ, and d_n codes the sequence of length 1 whose sole entry is the code for D. Assuming Γ is recursive, the second condition here defines a recursive relation.

The first condition defines a primitive recursive set, and the whole matter boils down to proving as much. Now the sequence of sequents coded by a code for $(c_1, d_1, \ldots, c_n, d_n)$ will be derivation if for each $i \leq n$, the presence of c_i and d_i is justified by the presence of zero, one, or more earlier pairs, such that the sequent $\Gamma_i \Rightarrow \Delta_i$ coded by c_i and d_i follows from the sequents $\Gamma_j \Rightarrow \Delta_j$ coded by these earlier $c_j$and d_j according to one or another rule. In gross form, then, the definition of coding a derivation will resemble the definition of coding a formation sequence, where the presence of any code for an expression must be justified by the presence of zero, one, or more earlier codes for expressions from which the given expression 'follows' by or another 'rule' of formation. The rules of formation are just the rules the zero-'premiss' rule allowing atomic formulas to appear, the one-'premiss' rule allowing a negation to be 'inferred' from the expression it negates, the two-'premiss' rule allowing a disjunction to be 'inferred' from the two expressions it disjoins—and so on. Definitions of this gross form define primitive recursive relations, provided the individual rules in them do.

So, going back to derivations, let us look at a typical one-premiss rule. (The zero-premiss rule would be a bit simpler, a two-premiss rule a bit more complicated.) Take

$$\text{(R2}a\text{)} \qquad \frac{\Gamma \cup \{A\} \Rightarrow \Delta}{\Gamma \Rightarrow \{\sim A\} \cup \Delta}.$$

The relation we need to show to be primitive recursive is the relation 'e and f code a sequent that follows from the sequent coded by c and d according to (R2a)'. But this can be defined as follows:

$$\begin{aligned}
&c, d, e, f \text{ code sets of formulas, and } \exists a < \mathrm{lh}(c) \quad \exists b < \mathrm{lh}(f)\\
&\quad \mathrm{ent}(f, b) = \mathrm{neg}(\mathrm{ent}(c, a)), \text{ and}\\
&\quad \forall i < \mathrm{lh}(c)\ (i = a \text{ or } \exists j < \mathrm{lh}(e)\ \mathrm{ent}(c, i) = \mathrm{ent}(e, j)), \text{ and}\\
&\quad \forall i < \mathrm{lh}(e) \quad \exists j < \mathrm{lh}(c)\ \mathrm{ent}(e, i) = \mathrm{ent}(c, j), \text{ and}\\
&\quad \forall i < \mathrm{lh}(d) \quad \exists j < \mathrm{lh}(f)\ \mathrm{ent}(d, i) = \mathrm{ent}(f, j), \text{ and}\\
&\quad \forall i < \mathrm{lh}(f)(i = b \text{ or } \exists j < \mathrm{lh}(d)\ \mathrm{ent}(f, i) = \mathrm{ent}(d, j)).
\end{aligned}$$

Here the last four clauses just say that the only difference between the sets coded by c and e is the presence of the sentence A coded by ent(c, a) in the former, and the only difference between the sets coded by d and f is the presence of the sentence B coded by ent(f, b) in the latter. The second clause tells us that $B = \sim A$. This is a primitive recursive relation, since we known neg is a primitive recursive function.

To supply a full proof, each of the rules would have to be analyzed in this way. In general, the analyses would be very similar, the main difference being in the second clauses, stating how the 'exiting' and 'entering' sentences are related. In the case we just looked at, the relationship was very simple: one sentence was the negation of the other. In the case of some other rules, we would need to know that the function taking a formula $B(x)$ and a closed term t to the result $B(t)$ of substituting t for all the free occurrences of x is recursive, or rather, that the corresponding function on codes is. We suppress the details.

Problems

15.1 On the first scheme of coding considered in this chapter, show that the *length* of, or number of symbols in, the expression with code number e is obtainable by a primitive recursive function from e.

15.2 Let Γ be a set of sentences, and T the set of sentences in the language of Γ that are deducible from Γ. Show that T is a theory.

15.3 Suppose an axiomatizable theory T has only infinite models. If T has only one isomorphism type of denumerable models, we know that it will be complete by Corollary 12.17, and decidable by Corollary 15.7. But suppose T is *not* complete, though it has only *two* isomorphism types of denumerable models. Show that T is still decidable.

15.4 Give examples of theories that are decidable though not complete.

15.5 Suppose $A_1, A_2, A_3, \ldots$ are sentences such that no A_n is provable from the conjunction of the A_m for $m < n$. Let T be the theory consisting of all sentences provable from the A_i. Show that T is not finitely axiomatizable, or in other words, that there are not some other, finitely many, sentences $B_1, B_2, \ldots, B_m$ such that T is the set of consequences of the B_j.

15.6 For a language with, say, just two nonlogical symbols, both two-place relation symbols, consider interpretations where the domain consists of the positive integers from 1 to n. How many such interpretations are there?

15.7 A sentence D is *finitely* valid if every finite interpretation is a model of D. Outline an argument *assuming Church's thesis* for the conclusion that the set of sentences that are *not* finitely valid is semirecursive. (It follows from Trakhtenbrot's theorem, as in the problems at the end of chapter 11, that the set of such sentences is *not* recursive.)

15.8 Show that the function taking a pair consisting of a code number a of a sentence A and a natural number n to the code number for the conjunction $A \,\&\, A \,\&\, \cdots \,\&\, A$ of n copies of A is recursive.

15.9 *The Craig reaxiomatization lemma* states that any theory T whose set of theorems is semirecursive is axiomatizable. Prove this result.

15.10 Let T be an axiomatizable theory in the language of arithmetic. Let f be a one-place total or partial function f of natural numbers, and suppose there is a formula $\phi(x, y)$ such that for any a and b, $\phi(\mathbf{a}, \mathbf{b})$ is a theorem of T if and only if $f(a) = b$. Show that f is a recursive total or partial function.

16
Representability of Recursive Functions

In the preceding chapter we connected our work on recursion with our work on formulas and proofs in one way, by showing that various functions associated with formulas and proofs are recursive. In this chapter we connect the two topics in the opposite way, by showing how we can 'talk about' recursive functions using formulas, and prove things about them in theories formulated in the language of arithmetic. In section 16.1 we show that for any recursive function f, we can find a formula ϕ_f such that for any natural numbers a and b, if $f(a) = b$ then $\forall y(\phi_f(\mathbf{a}, y) \leftrightarrow y = \mathbf{b})$ will be true in the standard interpretation of the language of arithmetic. In section 16.2 we strengthen this result, by introducing a theory **Q** *of* minimal arithmetic*, and showing that for any recursive function f, we can find a formula ψ_f such that for any natural numbers a and b, if $f(a) = b$ then $\forall y(\psi_f(\mathbf{a}, y) \leftrightarrow y = \mathbf{b})$ will be not merely true, but provable in* **Q**. *In section 16.3 we briefly introduce a stronger theory* **P** *of* Peano arithmetic*, which includes axioms of* mathematical induction*, and explain how these axioms enable us to prove results not obtainable in* **Q**. *The brief, optional section 16.4 is an appendix for readers interested in comparing our treatment of these matters here with other treatments in the literature.*

16.1 Arithmetical Definability

In Chapter 9, we introduced the language L^* of arithmetic and its standard interpretation $\mathcal{N}^*$. We now abbreviate 'true in the standard interpretation' to *correct*. Our goal in this chapter is to show that we can 'talk about' recursive functions in the language of arithmetic, and we begin by making talk about 'talking about' precise. We say a formula $F(x)$ of the language of arithmetic *arithmetically defines* a set S of natural numbers if and only if for all natural numbers a we have Sa if and only if $F(\mathbf{a})$ is correct. We say the set S is *arithmetically definable*, or *arithmetical* for short, if some formula arithmetically defines it. These notions naturally extend to two-place or many-place relations. A formula $F(x, y)$ arithmetically defines a relation R on natural numbers if and only if for all natural numbers a and b we have Rab if and only if $F(\mathbf{a}, \mathbf{b})$ is correct. The notions also naturally extend to functions, a function being counted as arithmetical if and only if its graph relation is arithmetical. Thus a one-place function f is arithmetical if and only if there is a formula $F(x, y)$ of the language of arithmetic such that for all a and b we have $f(a) = b$ if and only if $F(\mathbf{a}, \mathbf{b})$ is correct.

16.1 Examples (Basic functions). To give the most trivial example, the identity function $\mathrm{id} = \mathrm{id}^1_1$is arithmetically defined by the formula $y = x$, and more generally, id^n_i is arithmetically defined by the formula $y = x_i$, or if we want the other x_j to be mentioned, by the formula

$$x_1 = x_1 \,\&\ldots\&\, x_n = x_n \,\&\, y = x_i.$$

The zero function $\mathrm{const}_0(x) = 0$ is also arithmetically definable, by the formula $y = \mathbf{0}$, or if we want x to be mentioned, by the formula $x = x \;\&\; y = \mathbf{0}$. The successor, addition, and multiplication functions are arithmetically definable by the formulas $y = x'$, $y = x_1 + x_2$, and $y = x_1 \cdot x_2$.

16.2 Examples (Other arithmetical functions). Of course, it is no surprise that the functions we have just been considering are arithmetically definable, since they are 'built in': we have included in the language special symbols expressly for them. But their inverses, for which we have not built in symbols, are also arithmetical. The predecessor function is arithmetically definable by the following formula $F_{\mathrm{pred}}(x_1, y)$:

$$(x_1 = \mathbf{0} \,\&\, y = \mathbf{0}) \vee x_1 = y'.$$

The difference function $x_1 \dot{-} x_2$ is arithmetically defined by the following formula $F_{\mathrm{dif}}(x_1, x_2, y)$:

$$(x_1 < x_2 \,\&\, y = \mathbf{0}) \vee (x_1 = x_2 + y)$$

and the quotient and remainder functions $\mathrm{quo}(x_1, x_2)$ and $\mathrm{rem}(x_1, x_2)$ are arithmetically defined by the following formulas $F_{\mathrm{quo}}(x_1, x_2, y)$ and $F_{\mathrm{rem}}(x_1, x_2, y)$:

$$(x_2 = \mathbf{0} \,\&\, y = \mathbf{0}) \vee \exists u < x_2 x_1 = y \cdot x_2 + u$$
$$(x_2 = \mathbf{0} \,\&\, y = x_1) \vee (y < x_2 \,\&\, \exists u \leq x_1\; x_1 = u \cdot x_2 + y).$$

On the other hand, it is not obvious how to define exponentiation, and as a temporary expedient we now expand the language of arithmetic by adding a symbol ↑, thus obtaining the language of *exponential arithmetic*. Its standard interpretation is like that of the original language arithmetic, with the denotation of ↑ being the usual exponentiation function. In terms of this expansion we define *↑-arithmetical definability* in the obvious way. (The expression '↑-arithmetical' may be pronounced 'exponential-arithmetical' or 'exp-arithmetical' for short.)

16.3 Examples (↑-arithmetical functions). Examples of ↑-arithmetical functions include the exponential function itself, its inverses the logarithm functions (lo and lg of Example 7.11), and, what will be more significant for our present purposes, any number of functions pertaining to the coding of finite sequences of numbers by single numbers or pairs of numbers. For instance, in section 1.2 we found one serviceable if not especially elegant way of coding sequences by pairs for which the ith entry of the sequence coded by the pair (s, t) could be recovered using the function

$$\mathrm{entry}(i, s, t) = \mathrm{rem}(\mathrm{quo}(s, t^i), t)$$

This function is $\uparrow$-arithmetically definable by the following formula $F_{\text{ent}}(x_1, x_2, x_3, y)$:

$$\exists z \leq x_3 \uparrow x_1 (F_{\text{quo}}\ (x_2, x_3 \uparrow x_1, z)\ \&\ F_{\text{rem}}\ (z, x_2, y)).$$

For this just says that there is something that is the quotient on dividing x_2 by $x_3^{x_1}$, and whose remainder on dividing by x_2 is y, adding that it will be less than or equal to x_2 (as any quotient on dividing x_2 by anything must be).

Even after helping ourselves to exponentiation, it is still not obvious how to define *super*-exponentiation, but though not obvious, it is possible—in fact *any* recursive function can now be defined, as we next show.

16.4 Lemma. Every recursive function f is $\uparrow$-arithmetical.

Proof: Since we have already shown the basic functions to be definable, we need only show that if any of the three processes of composition, primitive recursion, or minimization is applied to $\uparrow$-arithmetical functions, the result is an $\uparrow$-arithmetical function. We begin with composition, the idea for which was already encountered in the last example. Suppose that f and g are one-place functions and that h is obtained from them by composition. Then clearly $c = h(a)$ if and only if

$$c = g(f(a))$$

which may be more long-windedly put as

there is something such that it is $f(a)$ and g(it) is c.

It follows that if f and g are $\uparrow$-arithmetically defined by ϕ_f and ϕ_g, then h is $\uparrow$-arithmetically defined by the following formula $\phi_h(x, z)$:

$$\exists y\ (\phi_f(x, y)\ \&\ \phi_g(y, z)).$$

[To be a little more formal about it, given any a, let $b = f(a)$ and let $c = h(a) = g(f(a)) = g(b)$. Since ϕ_f and ϕ_g define f and g, $\phi_f(\mathbf{a}, \mathbf{b})$ and $\phi_g(\mathbf{b}, \mathbf{c})$ are correct, so $\phi_f(\mathbf{a}, \mathbf{b})\ \&\ \phi_g(\mathbf{b}, \mathbf{c})$ is correct, so $\exists y(\phi_f(\mathbf{a}, y)\ \&\ \phi_g(y, \mathbf{c}))$ is correct, which is to say $\phi_h(\mathbf{a}, \mathbf{c})$ is correct. Conversely, if $\phi_h(\mathbf{a}, \mathbf{c})$ is correct, $\phi_f(\mathbf{a}, \mathbf{b})\ \&\ \phi_g(\mathbf{b}, \mathbf{c})$ and hence $\phi_f(\mathbf{a}, \mathbf{b})$ and $\phi_g(\mathbf{b}, \mathbf{c})$ must be correct for some b, and since ϕ_f defines f, this b can only be $f(a)$, while since ϕ_g defines g, c then can only be $g(b) = g(f(a)) = h(a)$.]

For the composition of a two-place function f with a one-place function g the formula would be

$$\exists y(\phi_f(x_1, x_2, y)\ \&\ \phi_g(y, z)).$$

For the composition of two one-place functions f_1 and f_2 with a two-place function g, the formula would be

$$\exists y_1 \exists y_2 (\phi_{f_1}(x, y_1)\ \&\ \phi_{f_2}(x, y_2)\ \&\ \phi_g(y_1, y_2, z))$$

and so on. The construction is similar for functions of more places.

Recursion is just a little more complicated. Suppose that f and g are one-place and three-place functions, respectively, and that the two-place function h is obtained from them by primitive recursion. Writing i' for the successor of i, clearly $c = h(a, b)$ if and only if there exists a sequence σ with the following three properties:

entry 0 of σ is $h(a, 0)$
for all $i < b$, if entry i of σ is $h(a, i)$, then entry i' of σ is $h(a, i')$
entry b of σ is c.

These conditions may be restated equivalently thus:

entry 0 of σ is $f(a)$
for all $i < b$, entry i' of σ is $g(a, i,$ entry i of $\sigma)$
entry b of σ is c.

These conditions may be restated more long-windedly thus:

there is something that is entry 0 of σ and is $f(a)$
for all $i < b$, there is something that is entry i of σ, and
there is something which is entry i' of σ, and
the latter is $g(a, i,$ the former)
entry b of σ is c.

Moreover, instead of saying 'there is a sequence' we may say 'there are two numbers coding a sequence'. It follows that if f and g are $\uparrow$-arithmetically defined by ϕ_f and ϕ_g, then h is $\uparrow$-arithmetically defined by the formula $\phi_h(x, y, z) = \exists s \exists t \phi$, where ϕ is the conjunction of the following three formulas:

$$\exists u(F_{\text{ent}}(\mathbf{0}, s, t, u) \,\&\, \phi_f(x, u))$$
$$\forall w < y\, \exists u \exists v(F_{\text{ent}}(w, s, t, u) \,\&\, F_{\text{ent}}(w', s, t, v) \,\&\, \phi_g(x, w, u, v))$$
$$F_{\text{ent}}(y, s, t, z).$$

The construction is exactly the same for functions of more places.

Minimization is a little simpler. Suppose that f is a two-place function, and that the one-place function g is obtained from it by minimization. Clearly $g(a) = b$ if and only if

$f(a, b) = 0$ and
for all $c < b$, $f(a, c)$ is defined and is not 0.

These conditions may be restated more long-windedly thus:

$f(a, b) = 0$ and
for all $c < b$, there is something that is $f(a, c)$, and it is not 0.

It follows that if f is $\uparrow$-arithmetically defined by ϕ_f, then g is $\uparrow$-arithmetically defined by the following formula $\phi_g(x, y)$:

$$\phi_f(x, y, \mathbf{0}) \,\&\, \forall z < y\, \exists u(\phi_f(x, z, u) \,\&\, u \neq \mathbf{0}).$$

The construction is exactly the same for functions of more places.

On reviewing the above construction, it will be seen that the presence of the exponential symbol $\uparrow$ in the language was required only for the formula F_{ent}. If we could find some *other* way to code sequences by pairs for which the entry function could be defined *without* using exponentiation, then we could forget about $\uparrow$. And in fact, a coding is possible for which

$$\text{entry}(i, s, t) = \text{rem}(s, t(i+1)+1)$$

so that for F_{ent} we may take

$$F_{\text{rem}}(x_2, x_3 \cdot (x_1 \mathbf{+} \mathbf{1}) \mathbf{+} \mathbf{1}, y).$$

That such a coding is possible is the content of the following lemma.

16.5 Lemma (β-function lemma). For every k and every $a_0, a_1, \ldots, a_k$ there exist s and t such that for all i with $0 \le i \le k$ we have $a_i = \text{rem}(s, t(i+1)+1)$.

Proof: This result follows directly from the proofs of two ancient and famous theorems of number theory, to be found in a prominent place in any textbook on that subject. Since this is not a textbook on number theory, we are not going to develop the whole subject from the foundations, but we do give an indication of the proof. The first ingredient is the *Chinese remainder theorem*, so called from the appearance (at least of special cases) of the theorem in the ancient *Mathematical Classic* of Sun Zi and the medieval *Mathematical Treatise in Nine Sections* of Qin Jiushao. This theorem states that given any numbers $t_0, t_1, \ldots, t_n$ no two of which have a common prime factor, and given any numbers $a_i < t_i$, there is a number s such that $\text{rem}(s, t_i) = a_i$ for all i from 0 to n. The proof is sufficiently illustrated by the case of two numbers t and u with no common prime factor, and two numbers $a < t$ and $b < u$. Every one of the tu numbers i with $0 \le i < tu$ produces one of the tu pairs (a, b) with $a < t$ and $b < u$ on taking the remainders rem (s, t) and rem (s, u). To show that, as asserted by the theorem, every pair (a, b) is produced by some number s, it suffices to show that no two distinct numbers $0 \le s < r < tu$ produce the same pair. If s and r do produce the same pair, then they leave the same remainder when divided by t, and leave the same remainder when divided by u. In that case, their difference $q = r - s$ leaves remainder zero when divided by either t or u. In other words, t and u both divide q. But when numbers with no common prime factor both divide a number, so does their product. Hence tu divides q. But this is impossible, since $0 < q < tu$.

The second ingredient comes from the proof in Euclid's *Elements of Geometry* that there exist infinitely many primes. Given any number n, we want to find a prime $p > n$. Well, let $N = n!$, so that in particular N is divisible by every prime $\le n$. Then $N + 1$, like any number >1, has a prime factor p. (Possibly N is itself prime, in which case we have $p = N$.) But we cannot have $p \le n$, since when N is divided by any number $\le n$, there is a remainder of 1. A slight extension of the argument shows that any two distinct numbers $N \cdot i + 1$ and $N \cdot j + 1$ with $0 < i < j \le n$ have no common prime factor. For if a prime p divides both numbers, it divides their difference $N(j - i)$. This is a product of factors $\le n$, and when a prime divides a product of several factors, it must divide one of the factors; so p itself must be a number $\le n$. But then p can*not* divide $N \cdot i + 1$ or $N \cdot j + 1$. Now given k and every $a_0, a_1, \ldots, a_k$, taking n larger than all of them, and letting t be a number divisible by every prime $\le n$, no two of the numbers $t_i = t(i+1)+1$ will have a common prime factor, and we will of course have $a_i < t_i$, so there will be an s such that $\mathrm{rem}(s, t_i) = a_i$ for all i with $0 \le i \le k$.

Thus we have proved part (a) of the following.

16.6 Lemma

(a) Every recursive function f is arithmetical.
(b) Every recursive set is arithmetical.

Proof: As remarked just before the statement of the lemma, we already have (a). For (b), if R is an n-place recursive relation and f its characteristic function, then apply (a) to get a formula $\phi(x_1, \ldots, x_n, y)$ arithmetically defining f. Then the formula $\phi(x_1, \ldots, x_n, \mathbf{1})$ arithmetically defines R.

Further sharpening of the result depends on distinguishing different kinds of formulas. By a *rudimentary formula* of the language of arithmetic we mean a formula built up from atomic formulas using only negation, conjunction, disjunction, and bounded quantifications $\forall x < t$ and $\exists x < t$, where t may be any term of the language (not involving x). (Conditionals and biconditionals are allowed, too, since these officially are just abbreviations for certain constructions involving negation, conjunction, and disjunction. So are the bounded quantifiers $\forall x \le t$ and $\exists x \le t$, since these are equivalent to $\forall x < t'$ and $\exists x < t'$). By an $\exists$-*rudimentary formula* we mean a formula of form $\exists x F$ where F is rudimentary, and similarly for an $\forall$-*rudimentary formula*. (The negation of an $\exists$-rudimentary formula is equivalent to an $\forall$-rudimentary formula, and conversely.) Many major theorems of number theory are naturally expressible by $\forall$-rudimentary formulas.

16.7 Examples (Theorems of number theory). *Lagrange's theorem* that every natural number is the sum of four squares is naturally expressible by an $\forall$-rudimentary sentence as follows:

$$\forall x\, \exists y_1 < x\, \exists y_2 < x\, \exists y_3 < x\, \exists y_4 < x\;\, x = y_1 \cdot y_1 + y_2 \cdot y_2 + y_3 \cdot y_3 + y_4 \cdot y_4.$$

Bertrand's postulate, or *Chebyshev's theorem*, that there is a prime between any number greater than one and its double, is naturally expressible by an $\forall$-rudimentary sentence as follows:

$$\forall x(\mathbf{1} < x \to \exists y < 2 \cdot x(x < y \;\&\; {\sim}\exists u < y\, \exists v < y\;\, y = u \cdot v)).$$

Our present concern, however, will be with $\exists$-rudimentary formulas and with *generalized* $\exists$-rudimentary formulas, which include all formulas obtainable from rudimentary formulas by conjunction, disjunction, bounded universal quantification, bounded existential quantification, and unbounded existential quantification. Reviewing the proof of Lemma 16.6, one finds that the formulas defining the basic functions and the formula F_{ent} are rudimentary, and that the formula defining a composition of functions is obtained by conjunction, bounded quantification, and existential quantification from rudimentary formulas and the formulas defining the original functions, and similarly for recursion and minimization. Hence we have proved:

16.8 Lemma. Every recursive function is arithmetically definable by a generalized $\exists$-rudimentary formula.

The next refinement will be to get rid of the word 'generalized' here. Two formulas with, say, two free variables, $\phi(x, y)$ and $\psi(x, y)$, are called *arithmetically equivalent* if for all numbers a and b, $\phi(\mathbf{a}, \mathbf{b})$ is correct if and only if $\psi(\mathbf{a}, \mathbf{b})$ is correct. Clearly arithmetically equivalent formulas define the same relation or function. The condition that ϕ and ψ are arithmetically equivalent is equivalent to the condition that the biconditional

$$\forall x \forall y(\phi(x, y) \leftrightarrow \psi(x, y))$$

is correct. In particular, if ϕ and ψ are logically equivalent—in which case the biconditional is true not just in the standard interpretation, but in *any* interpretation—then they are arithmetically equivalent. The following lemma bears more than a passing resemblance to Corollary 7.15.

16.9 Lemma (Closure properties of $\exists$-rudimentary formulas).

(a) Any rudimentary formula is arithmetically equivalent to an $\exists$-rudimentary formula.
(b) The conjunction of two $\exists$-rudimentary formulas is arithmetically equivalent to an $\exists$-rudimentary formula.
(c) The disjunction of two $\exists$-rudimentary formulas is arithmetically equivalent to an $\exists$-rudimentary formula.
(d) The result of applying bounded universal quantification to an $\exists$-rudimentary formula is arithmetically equivalent to an $\exists$-rudimentary formula.
(e) The result of applying bounded existential quantification to an $\exists$-rudimentary formula is arithmetically equivalent to an $\exists$-rudimentary formula.
(f) The result of applying (unbounded) existential quantification to an $\exists$-rudimentary formula is arithmetically equivalent to an $\exists$-rudimentary formula.

Proof: For (a), ϕ is logically equivalent to $\exists w(w = w \,\&\, \phi)$ (and if ϕ is rudimentary, so is $w = w \,\&\, \phi$).

For (b), $\exists u \phi(u) \,\&\, \exists v \psi(v)$ is arithmetically equivalent to

$$\exists w\, \exists u < w\, \exists v < w\, (\phi(u) \,\&\, \psi(v)).$$

[and if $\phi(u)$ and $\psi(v)$ are rudimentary, so is $\exists u < w\, \exists v < w\, (\phi(u) \,\&\, \psi(v))$]. The implication in one direction is logical, and in the other direction we use the fact that for any two natural numbers u and v, there is always a natural number w greater than both.

For (c), $\exists u \phi(u) \vee \exists v \psi(v)$ is logically equivalent to $\exists w(\phi(w) \vee \psi(w))$.

For (d), $\forall z < y\, \exists u \phi(u, z)$ is arithmetically equivalent to

$$\exists w\, \forall z < y\, \exists u < w\, \phi(u, z).$$

The implication in one direction is logical, and in the other direction we use the fact that for any finitely many natural numbers $u_0, u_1, \ldots, u_{y-1}$ there is a number w that is greater than all the u_z.

For (e), $\exists z < y\, \exists u\, \phi(u, z)$ is logically equivalent to $\exists u\, \exists z < y\, \phi(u, z)$.

For (f), $\exists u\, \exists v\, \phi(u, v)$ is arithmetically equivalent to $\exists w\, \exists u < w\, \exists v < w\, \phi(u, v)$, much as in part (b).

Repeated application of Lemma 16.9, followed by combination with Lemma 16.8 give the following:

16.10 Proposition. Every generalized ∃-rudimentary formula is arithmetically equivalent to an ∃-rudimentary formula.

16.11 Lemma. Every recursive function is arithmetically definable by an ∃-rudimentary formula.

Call a function that is arithmetically definable by a rudimentary formula a *rudimentary function*. Can we go further and show every recursive function to be rudimentary? Not quite. The next lemma tells us how far we *can* go. It bears more than a passing resemblance to Proposition 7.17.

16.12 Lemma. Every recursive function is obtainable by composition from rudimentary functions.

Proof: Let f be a recursive function of, say, one place. (The proof for many-place functions is exactly the same.) We know f is arithmetically definable by an ∃-rudimentary formula $\exists z \phi(x, y, z)$. Let S be the relation arithmetically defined by ϕ, so that we have

$$Sabc \leftrightarrow \phi(\mathbf{a}, \mathbf{b}, \mathbf{c}) \text{ is correct.}$$

We have

$$f(a) = b \leftrightarrow \exists c\; Sabc.$$

We now introduce two auxiliary functions:

$$g(a) = \begin{cases} \text{the least } d \text{ such that} & \\ \quad \exists b < d\, \exists c < d\; Sabc & \text{if such a } d \text{ exists} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$h(a, d) = \begin{cases} \text{the least } b < d \text{ such that} & \\ \quad \exists c < d\; Sabc & \text{if such a } b \text{ exists} \\ 0 & \text{otherwise.} \end{cases}$$

(Note that if f is total, then g is total, while h is always total.) These functions are rudimentary, being arithmetically definable by the following formulas $\phi_g(x, w)$ and $\phi_h(x, w, y)$:

$$\exists y < w\, \exists z < w\, \phi(x, y, z)\ \&\ \forall v < w\, \forall y < v\, \forall z < v \sim\phi(x, y, z)$$

$$\exists z < w\, \phi(x, y, z)\ \&\ \forall u < y\, \forall z < w \sim\phi(x, u, z)$$

and a little thought shows that $f(x) = h(x, g(x)) = h(\mathrm{id}\,(x), g(x))$, so $f = \mathrm{Cn}[h, \mathrm{id},\ g]$ is a composition of rudimentary functions.

If T is a consistent theory in the language of arithmetic, we say a set S is *defined* in T by $D(x)$ if for all n, if n is in S, then $D(\mathbf{n})$ is a theorem of T, and if n is not in S, then $\sim D(\mathbf{n})$ is a theorem of T. S is *definable* in T if S is defined by some formula. Arithmetical definability is simply the special case where T is *true arithmetic*, the set of all correct sentences. The general notion of definability in a theory extends to relations, but definability of a function turns out to be less useful than a related notion. For the remainder of this chapter, unless otherwise noted, 'function' will mean 'total function'. Let f be a one-place function. (The definition we are about to give extends easily to many-place functions.) We say f is *representable* in T if there is a formula $F(x, y)$ such that whenever $f(a) = b$, the following is a theorem of T:

$$\forall y(F(\mathbf{a}, y) \leftrightarrow y = \mathbf{b}).$$

This is logically equivalent to the conjunction of the positive assertion

$$F(\mathbf{a}, \mathbf{b})$$

and the general negative assertion

$$\forall y(y \neq \mathbf{b} \rightarrow \sim F(\mathbf{a}, y)).$$

By contrast, definability would only require that we have the positive assertion and for each particular $c \neq b$ the relevant particular instance of the general negative assertion, namely, $\sim F(\mathbf{a}, \mathbf{c})$.

Now in the special case where T is true arithmetic, of course if each particular numerical instance is correct, then the universal generalization is correct as well, so representability and definability come to the same thing. But for other theories, each particular numerical instance may be a theorem without the universal generalization being a theorem, and representability is in general a stronger requirement than definability. Note that if T is a *weaker* theory than T^* (that is, if the set of theorems of T is a subset of the set of theorems of T^*), then the requirement that a function be representable in T is a *stronger* requirement than that it be representable in T^* (that is, representability in T implies representability in T^*). Thus far we have proved all recursive functions to be representable in true arithmetic. If we are to strengthen our results, we must consider weaker theories than that.

16.2 Minimal Arithmetic and Representability

We now introduce a finite set of *axioms of minimal arithmetic* **Q**, which, though not strong enough to prove major theorems of number theory, at least are correct and strong enough to prove all correct ∃-rudimentary sentences. By themselves, the axioms of **Q** would not be adequate for number theory, but any set of adequate axioms would have to include them, or at least to prove them (in which case the set might as well include them). Our main theorems (Theorems 16.13 and 16.15) apply to any theory T that contains **Q**, and since **Q** is weak, the theorems are correspondingly strong.

In displaying the list of axioms we make use of a traditional convention, according to which when displaying sentences of the language of arithmetic that begin with a string of one or more universal quantifiers, one may omit to write the quantifiers and write only the open formula that comes after them.

(Q1) $\mathbf{0} \neq x'$

(Q2) $x' = y' \rightarrow x = y$

(Q3) $x + \mathbf{0} = x$

(Q4) $x + y' = (x + y)'$

(Q5) $x \cdot \mathbf{0} = \mathbf{0}$

(Q6) $x \cdot y' = (x \cdot y) + x$

(Q7) $\sim x < \mathbf{0}$

(Q8) $x < y' \leftrightarrow (x < y \vee x = y)$

(Q9) $\mathbf{0} < y \leftrightarrow y \neq \mathbf{0}$

(Q10) $x' < y \leftrightarrow (x < y \,\&\, y \neq x')$

Thus axiom (Q1) is really $\forall x\ \mathbf{0} \neq x'$, axiom (Q2) is really $\forall x \forall y\ (x' = y' \rightarrow x = y)$, and so on. As is said, the real axioms are the *universal closures* of the formulas displayed. The theory **Q** of *minimal arithmetic* is the set of all sentences of the language of arithmetic that are provable from (or, equivalently, are true in all models of) these axioms. The significance of the various axioms will become clear as we work through the steps of the proof of the main theorem of this section.

16.13 Theorem. An ∃-rudimentary sentence is correct if and only if it is a theorem of **Q**.

Proof: Since every axiom of **Q** is correct, so is every theorem of **Q**, and hence any ∃-rudimentary sentence provable from the axioms of **Q** is correct. All the work will go into proving the converse. To begin with zero and sucessor, for any natural number m, of course $\mathbf{m} = \mathbf{m}$ (where $\mathbf{m}$ is as always the numeral for m, that is, is the term $\mathbf{0}'^{\cdots\prime}$ with m accents $'$) is provable even without any axioms, by pure logic.

All of $\mathbf{0} \neq \mathbf{1}, \mathbf{0} \neq \mathbf{2}, \mathbf{0} \neq \mathbf{3}, \ldots,$ are provable by (Q1) (since the numerals $\mathbf{1}, \mathbf{2}, \mathbf{3}, \ldots$ all end in accents). Then $\mathbf{1} = \mathbf{2} \rightarrow \mathbf{0} = \mathbf{1}, \mathbf{1} = \mathbf{3} \rightarrow \mathbf{0} = \mathbf{2}, \ldots$ are provable using (Q2), and since $\mathbf{0} \neq \mathbf{1}, \mathbf{0} \neq \mathbf{2}, \ldots$ are provable, it follows by pure logic that $\mathbf{1} \neq \mathbf{2}, \mathbf{1} \neq \mathbf{3}, \ldots,$ are provable. Then $\mathbf{2} = \mathbf{3} \rightarrow \mathbf{1} = \mathbf{2}, \mathbf{2} = \mathbf{4} \rightarrow \mathbf{1} = \mathbf{3}, \ldots$ are provable, again using (Q2), and since $\mathbf{1} \neq \mathbf{2}, \mathbf{1} \neq \mathbf{3}, \ldots,$ are provable, it follows by pure logic that $\mathbf{2} \neq \mathbf{3}, \mathbf{2} \neq \mathbf{4}, \ldots$ are provable. Continuing in the same way, if $m < n$, then $\mathbf{m} \neq \mathbf{n}$ is provable.

It follows by pure logic (the symmetry of identity) that if $m < n$, then $\mathbf{n} \neq \mathbf{m}$ is provable also. Since in general if $m \neq n$ we have either $m < n$ or $n < m$, it follows that if $m \neq n$ then both $\mathbf{m} \neq \mathbf{n}$ and $\mathbf{n} \neq \mathbf{m}$ are provable.

Turning now to order, note that using (Q8), $x < \mathbf{1} \leftrightarrow (x < \mathbf{0} \vee x = \mathbf{0})$ is provable, and (Q7) is $\sim x < \mathbf{0}$. By pure logic $x < \mathbf{1} \leftrightarrow x = \mathbf{0}$ is provable from these, so that $\mathbf{0} < \mathbf{1}$ is provable, and since we already know that $\mathbf{1} \neq \mathbf{0}, \mathbf{2} \neq \mathbf{0}, \ldots$ are provable, it follows that $\sim\mathbf{1} < \mathbf{1}, \sim\mathbf{2} < \mathbf{1}, \ldots$ are provable. Then using (Q8) again, $x < \mathbf{2} \leftrightarrow (x < \mathbf{1} \vee x = \mathbf{1})$ is provable, from which, given what we already know to be provable, it follows that $x < \mathbf{2} \leftrightarrow (x = \mathbf{0} \vee x = \mathbf{1})$ is provable, from which it follows that $\mathbf{0} < \mathbf{2}$, $\mathbf{1} < \mathbf{2}$, and also $\sim\mathbf{2} < \mathbf{2}$, $\sim\mathbf{3} < \mathbf{2}$, ... are all provable. Continuing in the same way, for any m the following is provable:

(1) $$x < \mathbf{m} \leftrightarrow (x = \mathbf{0} \vee x = \mathbf{1} \vee \ldots \vee x = \mathbf{m} - \mathbf{1}).$$

Moreover, whenever $n < m$, $\mathbf{n} < \mathbf{m}$ is provable, and whenever $m \geq n$, $\sim\mathbf{m} < \mathbf{n}$ is provable.

Turning now to addition and multiplication, let us show how (Q3) and (Q4), which are of course just the formal versions of the recursion equations for addition, can be used to prove, for instance, $\mathbf{2} + \mathbf{3} = \mathbf{5}$, or $\mathbf{0}'' + \mathbf{0}''' = \mathbf{0}'''''$. Using (Q4), the following are all provable:

$$\mathbf{0}'' + \mathbf{0}''' = (\mathbf{0}'' + \mathbf{0}'')'$$
$$\mathbf{0}'' + \mathbf{0}'' = (\mathbf{0}'' + \mathbf{0}')'$$
$$\mathbf{0}'' + \mathbf{0}' = (\mathbf{0}'' + \mathbf{0})'.$$

Using (Q3), $\mathbf{0}'' + \mathbf{0} = \mathbf{0}''$ is provable. Working backwards, by pure logic the following are all provable from what we have so far:

$$\mathbf{0}'' + \mathbf{0}' = \mathbf{0}'''$$
$$\mathbf{0}'' + \mathbf{0}'' = \mathbf{0}''''$$
$$\mathbf{0}'' + \mathbf{0}''' = \mathbf{0}'''''$$

This is, in fact, just the formal calculation exhibited in section 6.1. Obviously this method is perfectly general, and whenever $a + b = c$ we can prove $\mathbf{a} + \mathbf{b} = \mathbf{c}$. Then also, again as in section 6.1, the recursion equations (Q5) and (Q6) for multiplication can be used to prove $\mathbf{2} \cdot \mathbf{3} = \mathbf{6}$ and more generally, whenever $a \cdot b = c$ to prove $\mathbf{a} \cdot \mathbf{b} = \mathbf{c}$.

If we next consider more complex terms involving $'$ and $+$ and $\cdot$, their correct values are also provable. For example, consider $(\mathbf{1} + \mathbf{2}) \cdot (\mathbf{3} + \mathbf{4})$. By what we have already said, $\mathbf{1} + \mathbf{2} = \mathbf{3}$ and $\mathbf{3} + \mathbf{4} = \mathbf{7}$, as well as $\mathbf{3} \cdot \mathbf{7} = \mathbf{21}$, are provable. From these it is provable by pure logic that $(\mathbf{1} + \mathbf{2}) \cdot (\mathbf{3} + \mathbf{4}) = \mathbf{21}$, and similarly for other complex terms. Thus for any closed term t built up from $\mathbf{0}$ using $'$, $+$, $\cdot$, it is provable what is the correct value of the term. Suppose then we have two terms s, t that have the same value m. Since by what we have just said $s = \mathbf{m}$ and $t = \mathbf{m}$ are provable, by pure logic $s = t$ is also provable. Suppose instead the two terms have different values m and n. Then since $s = \mathbf{m}$ and $t = \mathbf{n}$ and $\mathbf{m} \neq \mathbf{n}$ are provable, again by pure logic $s \neq t$ is also provable. A similar argument applies to order, so all correct formulas of types $s = t$, $s \neq t$, $s < t$, $\sim s < t$ are provable. Thus all correct closed atomic and negated atomic sentences are provable.

Now we move beyond atomic and negation-atomic sentences. First, by pure logic the double negation of a sentence is provable if and only if the sentence itself is, and a conjunction is provable if both its conjuncts are, a disjunction is provable if either of its disjuncts is, a negated conjunction is provable if the negation of one of its conjuncts is, and a negated disjunction is provable if the negations of both of its disjuncts are. Since all correct atomic and negated atomic closed sentences are provable, so are all correct sentences of types $\sim S$, $\sim\sim S$, S_1 & S_2, $\sim(S_1$ & $S_2)$, $S_1 \vee S_2$, $\sim(S_1 \vee S_2)$, where S, S_1, S_2 are atomic or negated atomic sentences. Continuing in this way, all correct closed formulas built up from atomic formulas by negation, conjunction, and disjunction are provable: All correct closed formulas without quantifiers are provable.

As for bounded quantifiers, using (1), for any formula $A(x)$ and any m, the following are provable:

$$\forall x < \mathbf{m} A(x) \leftrightarrow (A(\mathbf{0}) \,\&\, A(\mathbf{1}) \,\&\, \ldots \,\&\, A(\mathbf{m}-\mathbf{1})),$$
$$\exists x < \mathbf{m} A(x) \leftrightarrow (A(\mathbf{0}) \vee A(\mathbf{1}) \vee \ldots \vee A(\mathbf{m}-\mathbf{1})).$$

More generally, if t is a closed term whose correct value is m, since $t = \mathbf{m}$ is provable, so are the following:

$$\forall x < t A(x) \leftrightarrow (A(\mathbf{0}) \,\&\, A(\mathbf{1}) \,\&\, \ldots \,\&\, A(\mathbf{m}-\mathbf{1})),$$
$$\exists x < t A(x) \leftrightarrow (A(\mathbf{0}) \vee A(\mathbf{1}) \vee \ldots \vee A(\mathbf{m}-\mathbf{1})).$$

Thus any bounded universal or existential quantification of formulas without quantifiers can be proved equivalent to a conjunction or disjunction of sentences without quantifiers, which is of course itself then a sentence without quantifiers, so that we already know it can be proved if it is correct. Thus any correct sentence obtained by applying bounded universal or existential quantification to formulas without quantifiers is provable, and repeating the argument, so is any correct sentence built up from atomic formulas using negation, conjunction, disjunction, and bounded universal and bounded existential quantification: Any correct rudimentary sentence is provable.

Finally, consider now a correct $\exists$-rudimentary sentence $\exists x A(x)$. Since it is correct, there is some a such that $A(\mathbf{a})$ is correct. Being correct and rudimentary, $A(\mathbf{a})$ is provable, and hence so is $\exists x A(x)$, completing the proof.

Note that for a correct $\forall$-rudimentary sentence $\forall x A(x)$, we can conclude that each numerical instance $A(\mathbf{0})$, $A(\mathbf{1})$, $A(\mathbf{2})$, ... is provable from the axioms of **Q**, but this is not to say that $\forall x A(x)$ itself is provable from the axioms of **Q**, and in general it is not. There are nonstandard interpretations of the language of arithmetic on which all the axioms of **Q** come out true, but some very simple $\forall$-universal sentences that are correct or true on the standard interpretation come out false. Works on set theory develop an extremely natural nonstandard model of **Q**, called the system of *ordinal numbers,* for which, among others, laws as simple as $\mathbf{1} + x = x + \mathbf{1}$ fail. It would take us too far afield to stop to develop this model here, but some of its features are hinted at by the nonstandard interpretations of **Q** indicated in the problems at the end of the chapter. As we have already said, the fact that **Q** is a weak theory makes the following theorem (which automatically applies to any theory T containing **Q**) a strong theorem.

16.14 Lemma. Every rudimentary function is representable in **Q** (and by a rudimentary formula).

Proof: Inspection of the proof of the preceding lemma shows that it actually did not require any use of (Q9) and (Q10), but the proof of the present lemma does. An argument exactly like that used in the earlier proof to derive

(1) $$x < \mathbf{m} \leftrightarrow (x = \mathbf{0} \vee x = \mathbf{1} \vee \ldots \vee x = \mathbf{m}-\mathbf{1})$$

from (Q7) and (Q8) can be used to derive

$$\mathbf{m} < y \leftrightarrow (y \neq \mathbf{0} \,\&\, y \neq \mathbf{1} \,\&\ldots\&\, y \neq \mathbf{m}) \tag{2}$$

from (Q9) and (Q10). An immediate consequence of (1) and (2) together is the following:

$$z < \mathbf{m} \vee z = \mathbf{m} \vee \mathbf{m} < z. \tag{3}$$

Now let f be a one-place rudimentary function. (The proof for many-place functions is exactly the same.) Let $\phi(x, y)$ be a rudimentary formula arithmetically defining f. We do *not* claim that ϕ represents f in **Q**, but we do claim that ϕ can be used to build *another* rudimentary formula ψ that *does* represent f in **Q**. The formula $\psi(x, y)$ is simply

$$\phi(x, y) \,\&\, \forall z < y \sim \phi(x, z).$$

To show this formula represents f we must do two things. First, we must show that if $f(a) = b$, then $\psi(\mathbf{a}, \mathbf{b})$ is a theorem of **Q**. But indeed, since ϕ arithmetically defines f, if $f(a) = b$, then $\phi(\mathbf{a}, \mathbf{b})$ is correct, and $\sim\phi(\mathbf{a}, \mathbf{c})$ is correct for every $c \neq b$, and in particular for every $c < b$. Therefore $\forall z < \mathbf{b} \sim\phi(\mathbf{a}, z)$ is correct and $\psi(\mathbf{a}, \mathbf{b})$ is correct, and being rudimentary, it is a theorem of **Q** by Theorem 16.13.

Second, we must show that the following is a theorem of **Q**:

$$y \neq \mathbf{b} \rightarrow \sim\psi(\mathbf{a}, y),$$

which is to say

$$y \neq \mathbf{b} \rightarrow \sim(\phi(\mathbf{a}, y) \,\&\, \forall z < y \sim \phi(\mathbf{a}, z))$$

or, what is logically equivalent,

$$\phi(\mathbf{a}, y) \rightarrow (y = \mathbf{b} \vee \exists z < y\, \phi(\mathbf{a}, z)). \tag{4}$$

It will be sufficent to show that the following is a theorem of **Q**, since together with $\phi(\mathbf{a}, \mathbf{b})$, which we know to be a theorem of **Q**, it logically implies (4):

$$\phi(\mathbf{a}, y) \rightarrow (y = \mathbf{b} \vee \mathbf{b} < y). \tag{5}$$

But (3), together with $\forall y < \mathbf{b} \sim \phi(\mathbf{a}, y)$, which we know to be a theorem of **Q**, logically implies (5), to compete the proof.

16.15 Lemma. Any composition of rudimentary functions is representable in **Q** (and by an $\exists$-rudimentary formula).

Proof: We consider the composition of two one-place functions, the proof for many-place functions being similar. Suppose f and g are rudimentary functions, represented in **Q** by the rudimentary formulas ϕ_f and ϕ_g respectively. Let $h(x) = g(f(x))$, and consider the ($\exists$-rudimentary) formula $\phi_h$we get from the proof of Lemma 16.4:

$$\exists y(\phi_f(x, y) \;\&\; \phi_g(y, z))\,.$$

We claim ϕ_h represents h in $\mathbf{Q}$. For let a be any number, $b = f(a)$, and $c = h(a) = g(f(a)) = g(b)$. Since ϕ_f represents f and $f(a) = b$, the following is a theorem of $\mathbf{Q}$:

$$\forall y(\phi_f(\mathbf{a}, y) \leftrightarrow y = \mathbf{b})\,. \tag{1}$$

Since ϕ_g represents g and $g(b) = c$, the following is a theorem of $\mathbf{Q}$:

$$\forall z(\phi_g(\mathbf{b}, z) \leftrightarrow z = \mathbf{c})\,. \tag{2}$$

What we need to show in order to establish that ϕ_h represents h in $\mathbf{Q}$ is that the following is a theorem of $\mathbf{Q}$:

$$\forall z(\exists y(\phi_f(\mathbf{a}, y)\,\&\,\phi_g(y, z)) \leftrightarrow z = \mathbf{c})\,. \tag{3}$$

But (3) is logically implied by (1) and (2)!

16.16 Theorem

(a) Every recursive function is representable in $\mathbf{Q}$ (and by an $\exists$-rudimentary formula).
(b) Every recursive relation is definable in $\mathbf{Q}$ (and by an $\exists$-rudimentary formula).

Proof: (*a*) is immediate from Lemmas 16.12, 16.14, and 16.15. For (*b*), we consider the case of a one-place relation or set, many-place relations being similar. Let P be the recursive set, f its characteristic function, and $\exists w\phi(x, y, w)$ an $\exists$-rudimentary formula representing f in $\mathbf{Q}$. If n is in P, then $f(n) = 1$, and $\mathbf{Q}$ proves $\exists w\phi(\mathbf{n}, \mathbf{1}, w)$. If n is not in P, then $f(n) = 0$, and $\mathbf{Q}$ proves $\forall y(y \neq \mathbf{0} \rightarrow {\sim}\exists w\phi(\mathbf{n}, y, w))$ and in particular ${\sim}\exists w\phi(\mathbf{n}, \mathbf{0}, w)$. So the formula $\exists w\phi(x, \mathbf{1}, w)$ defines P in $\mathbf{Q}$.

Careful review of the proof of Theorem 16.16(a) shows that it actually applies to any recursive total *or partial* function f and gives a formula that *both arithmetically defines and* represents f in $\mathbf{Q}$. This refinement will not be needed, however, for our work in the next chapter.

We now have all the machinery we need for the proof of the *first Gödel incompleteness theorem*, and readers impatient to see that famous result may skip ahead to the next chapter. They should then return to the next brief section of this one before going on to the *second Gödel incompleteness theorem* in the chapter after next.

16.3 Mathematical Induction

The most immediate reason for the inadequacy of the axioms of minimal arithmetic to prove many correct $\forall$-universal sentences is that they make no provision for proof by mathematical induction, a method ubiquitously used in number theory and mathematics generally, according to which we can prove that every number has some property by proving that zero has it (the *zero* or *basis* step), and proving that, assuming a number x has it (an assumption called the *induction hypothesis*) then the successor of x also has it (the *successor* or *induction* step).

16.17 Example (Dichotomy). As the most trivial example, we can prove by mathematical induction that every x is either 0 or the successor of some number. *Basis*. 0 is 0. *Induction*. x' is the successor of x.

Another example is the proof of the law

$$0+1+2+\cdots+x = x(x+1)/2.$$

Basis. $0 = 0\cdot 1/2$. *Induction*. Assuming the result for x, we have

$$\begin{aligned} 0+1+2+\cdots+x+(x+1) &= x(x+1)/2+(x+1) \\ &= [x(x+1)+2(x+1)]/2 \\ &= (x+1)(x+2)/2. \end{aligned}$$

The algebraic manipulations in this proof depend on basic laws of arithmetic (associative, commutative, distributive) which can be proved using mathematical induction.

16.18 Example (Additive identity). By mathematical induction one can prove (from the recursion equations defining addition) $0+x = x+0$. *Zero* or *basis* step: for $x = 0$ we have $0+0 = 0+0$ by pure logic. *Successor* or *induction* step: assuming $0+x = x+0$, we have

$$\begin{aligned} 0+x' &= (0+x)' && \text{by the second recursion equation for addition} \\ (0+x)' &= (x+0)' && \text{by our assumption} \\ (x+0)' &= x' = x'+0 && \text{by the first recursion equation for addition.} \end{aligned}$$

16.19 Example (First case of the commutativity of addition). Similarly, we can prove $1+x = x+1$, or $0'+x = x+0'$. *Basis*: $0'+0 = 0+0'$ by the preceding example. *Induction*: assuming $0'+x = x+0'$, we have

$$\begin{aligned} 0'+x' &= (0'+x)' && \text{by the second recursion equation for addition} \\ (0'+x)' &= (x+0')' && \text{by assumption} \\ (x+0')' &= (x+0)'' && \text{by the second recursion equation for addition} \\ (x+0)'' &= x'' && \text{by the first recursion equation for addition} \\ x'' &= (x'+0)' && \text{by the first recursion equation for addition} \\ (x'+0)' &= x'+0' && \text{by the second recursion equation for addition.} \end{aligned}$$

We relegate further examples of this kind to the problems at the end of the chapter.

Once we have the basic laws of arithmetic, we can go on to prove various elementary lemmas of number theory such as the facts that a divisor of a divisor of a number is a divisor of that number, that every number has a prime factor, that if a prime divides a product it divides one of its factors, and that if two numbers with no common prime factor both divide a number, then so does their product. (The reader may recognize these as results we took for granted in the proof of Lemma 16.5.) Once we have enough elementary lemmas, we can go on to prove more substantial theorems of number theory, such as Lagrange's theorem from Example 16.7.

Closely related to the principle of mathematical induction as stated above is the principle of *complete induction*, according to which we can prove that every number has some property P by proving that zero has P, and proving that, assuming every number $\leq x$ has P, then the successor of x also has P. Indeed, complete induction for a property P follows on applying mathematical induction to the related property 'every number $\leq x$ has P,' using the facts (Q7) that 0 is the only number ≤ 0, and (Q8) that the only numbers $\leq x'$ are the numbers $\leq x$ and x' itself.

Another related principle is the *least-number principle*, according to which, if there is some number that has a given property, then there is a *least* number having the property, one such that no lesser number has it. This principle follows from the principle of mathematical induction as follows. Consider some property P such that there is *no* least number with the property P. Then we can use induction to show that in fact no number has the property P. We do this a bit indirectly, showing first by induction that for any number x, there is no number less than x with the property P. *Basis*: there is no number less than zero with the property P, because by (Q7) there is no number less than zero at all. *Induction*: supposing there is no number less than x with the property P, there can be no number less than the successor of x with the property P, since by (Q8) the only numbers less than the successor of x are the numbers less than x, which by assumption do not have the property, and x itself, which if it had the property would be the *least* number having the property. Now that we know that for any number x there is no number y less than x with the property, it follows that there is no number y with the property, since, taking x to be the successor of y, y is less than x and therefore cannot have the property.

(Conversely, the least-number principle together with the dichotomy of Example 16.17 yields the principle of mathematical induction. For if zero has a property and the successor of any number having the property has it also, then neither zero nor any successor can be the *least* number failing to have the property.)

All our argumentation in this section so far has been informal. A more adequate set of formal axioms for number theory is provided by the set of *axioms of Peano arithmetic*—an infinite (but primitive recursive) set of axioms consisting of the axioms of **Q** plus all sentences of the following form:

$$(A(\mathbf{0})\ \&\ \forall x(A(x) \rightarrow A(x'))) \rightarrow \forall x A(x).$$

[Here $A(x)$ may contain other free variables $y_1, \ldots, y_n$, and what is really meant is

$$\forall y_1 \ldots \forall y_n((A(\mathbf{0}, y_1, \ldots, y_n)\ \&\ \forall x(A(x, y_1, \ldots, y_n) \rightarrow A(x', y_1, \ldots, y_n))) \rightarrow \\ \forall x A(x, y_1, \ldots, y_n))$$

in accordance with the traditional convention of suppressing initial universal quantifiers in displayed formulas.]

The theory **P** of *Peano arithmetic* is the set of all sentences of the language of arithmetic that are provable from (or equivalently, are consequences of) these axioms. A rule to the effect that all sentences of a certain kind are to be taken as axioms is called an *axiom scheme*. With this terminology it would be said that the axioms of Peano arithmetic **P** consist of finitely many individual axioms (those of minimal arithmetic **Q**) plus a single axiom scheme (the *induction scheme* as above). In practice, the sets of axioms of most interest to logicians tend to consist of at most a dozen or so individual axioms and at most a very few axiom schemes, and so in particular are primitive recursive.

Among the axioms of **P** are for instance the following:

$$\begin{aligned}&(\mathbf{0} + \mathbf{0} = \mathbf{0} + \mathbf{0}\ \&\\ &\ \forall x(\mathbf{0} + x = x + \mathbf{0} \rightarrow \mathbf{0} + x' = x' + \mathbf{0})) \rightarrow\\ &\qquad \forall x\, \mathbf{0} + x = x + \mathbf{0}\end{aligned}$$

and

$$\begin{aligned}&(\mathbf{0}' + \mathbf{0} = \mathbf{0} + \mathbf{0}'\ \&\\ &\ \forall x(\mathbf{0}' + x = x + \mathbf{0}' \rightarrow \mathbf{0}' + x' = x' + \mathbf{0}')) \rightarrow\\ &\qquad \forall x\, \mathbf{0}' + x = x + \mathbf{0}'.\end{aligned}$$

And using these axioms in addition to the axioms of **Q**, the laws $\mathbf{0} + x = x + \mathbf{0}$ and $\mathbf{1} + x = x + \mathbf{1}$ are provable from the axioms of **P**, by 'formalizing' the proof of these laws given above as Examples 16.18 and 16.19. Also, for any formula $F(x)$ the least-number principle for F, namely

$$\exists x F(x) \rightarrow \exists x(F(x)\ \&\ \forall y < x \sim F(y))$$

is provable from the axioms of **P**, again by 'formalizing' the proof given above; and similarly for complete induction. Eventually the usual proofs of, say, Lagrange's theorem in textbooks on number theory can be 'formalized' to give proofs from the axioms of **P**.

The method of proof by mathematical induction is indeed an ingredient in the proofs of essentially all major theorems in mathematics, but it is perhaps especially common in *metamathematics*, the branch of mathematics concerned with giving proofs *about* what can be proved in mathematics—the branch to which the present book belongs. We have been using this method of proof all along, often in disguise. Consider, for instance, the proof by induction on complexity of formulas, of which we have made considerable use. What one does with this method is, not to put too fine a point on it, prove (as base step) that any atomic formula, which is to say, any formula containing 0 occurrences of the logical symbols (negation, junctions, quantifiers), has a certain property, and then prove (as inductive step) that if all formulas containing no more than n occurrences of the logical symbols have the property, then so does any formula containing n' such occurrences. The proof of the latter assertion is broken down into cases according as the one extra symbol is a negation, a junction, or a quantifier. This method of proof is really a special form of proof by mathematical induction.

And in our proof of Theorem 16.13 in the preceding section, for instance, every step involved some sort of induction, though we have expressed it very casually, using such phrases as 'continuing in the same way'. A less casual way of putting the second paragraph of the proof, for instance, would be as follows:

It can be proved by mathematical induction that if $m < n$, then $\mathbf{m} \neq \mathbf{n}$ is provable from the axioms of **Q**. *Basis*: if $0 < n$ then $\mathbf{0} \neq \mathbf{n}$ is provable by (Q0) (since the numeral **n** ends in an accent). *Induction*: assuming $\mathbf{m} \neq \mathbf{n}$ is provable whenever $m < n$, if $m' < n$, then we show $\mathbf{m}' \neq \mathbf{n}$ is provable as follows. Let $n = k'$. Then $m < k$, and by assumption $\mathbf{m} \neq \mathbf{k}$ is provable. But $\mathbf{m}' = \mathbf{k}' \rightarrow \mathbf{m} = \mathbf{k}$, which is to say $\mathbf{m}' = \mathbf{n} \rightarrow \mathbf{m} = \mathbf{k}$, is provable by (Q1). It follows by pure logic that $\mathbf{m} \neq \mathbf{n}$ is provable.

In this example we are using induction ('in the metalanguage') to prove something about a theory that does not have induction as an axiom ('in the object language'): we prove that something is a theorem of **Q** for every m by proving it is a theorem for 0, and that if it is a theorem for m, then it is a theorem for m'. Again, this sort of proof can be 'formalized' in **P**.

16.4* Robinson Arithmetic

This optional section is addressed to readers who wish to compare our treatment of the matters with which we have been concerned in this chapter with other treatments in the literature. In the literature, the label **Q** is often used to refer not to our minimal arithmetic but to another system, called *Robinson arithmetic*, for which we use the label **R**. To obtain the axioms of **R** from those of **Q**, add

$$\text{(Q0)} \qquad x = \mathbf{0} \vee \exists y\, x = y'$$

and replace (Q7)–(Q10) by

$$\text{(Q11)} \qquad x < y \leftrightarrow \exists z(z' + x = y).$$

We have already mentioned an extremely natural nonstandard model for **Q**, called the system of ordinal numbers, in which (Q0) fails. There is also an extremely natural nonstandard model for **R**, called the system of *cardinal numbers*, in which (Q10) fails; though it would take us too far afield to develop this model here, a simplified version suffices to show that some theorems of **Q** are not theorems of **R**. Thus **Q** is in some respects weaker and in some respects stronger than **R**, and vice versa.

By Theorem 16.16, every recursive function is representable in **Q**. Careful rereading of the proof reveals that all the facts it required about order are these, that the following are theorems:

$$\text{(1)} \qquad \mathbf{a} < \mathbf{b}, \text{ whenever } a < b$$

$$\text{(2)} \qquad \sim x < \mathbf{0}$$

$$\text{(3)} \qquad \mathbf{0} < y \leftrightarrow y \neq \mathbf{0}$$

and for any b the following:

$$\text{(4)} \qquad x < \mathbf{b}' \rightarrow x < \mathbf{b} \vee x = \mathbf{b}$$

$$\text{(5)} \qquad \mathbf{b} < y \,\&\, y \neq \mathbf{b}' \rightarrow \mathbf{b}' < y.$$

Clearly (1) is a theorem of **R**, since if $a < b$, then for some c, $c' + a = b$, and then $\mathbf{c}' + \mathbf{a} = \mathbf{b}$ is a consequence of (Q1)–(Q4). Also (2), which is axiom (Q7), is a theorem of **R**. For first $z' + \mathbf{0} = z' \neq \mathbf{0}$ by (Q3) and (Q1), which gives us $\sim\mathbf{0} < \mathbf{0}$, and then second $z' + y' = (z' + y') \neq \mathbf{0}$ by (Q4) and (Q1), which gives us $\sim y' < \mathbf{0}$. But these two, together with (Q0), give us (2). Also (3), which is axiom (Q9), is a theorem of **R**. For $\mathbf{0} < y \ \rightarrow\ y \neq \mathbf{0}$ follows from $\sim\mathbf{0} < \mathbf{0}$, and for the opposite direction, (Q0) gives $y \neq \mathbf{0} \rightarrow \exists z(y = z')$, while (Q3) gives $y = z' \rightarrow z' + \mathbf{0} = y$, and (Q11) gives $\exists z(z' + \mathbf{0} = y) \rightarrow \mathbf{0} < y$, and (3) is a logical consequence of these three.

It turns out that (4) and (5) are also theorems of **R**, and hence every recursive function is representable in **R**. Proofs have been relegated to the problems at the end of the chapter because we do not need any results about **R** for our later work. All we need for the purposes of proving the celebrated Gödel incompleteness theorems and their attendant lemmas and corollaries in the next chapter is that there is *some* correct, finitely axiomatizable theory in the language of arithmetic in which all recursive functions are representable. We chose minimal arithmetic because it is easier to prove representability for it; except in this regard Robinson arithmetic would really have done no worse and no better.

Problems

16.1 Show that the class of arithmetical relations is closed under substitution of recursive total functions. In other words, if P is an arithmetical set and f a recursive total function, and if $Q(x) \leftrightarrow P(f(x))$, then Q is an arithmetical set, and similarly for n-place relations and functions.

16.2 Show that the class of arithmetical relations is closed under negation, conjunction, disjunction, and universal and existential quantification, and in particular that every semirecursive relation is arithmetical.

16.3 A theory T is inconsistent if for some sentence A, both A and $\sim A$ are theorems of T. A theory T in the language of arithmetic is called *ω-inconsistent* if for some formula $F(x)$, $\exists x F(x)$ is a theorem of T, but so is $\sim F(\mathbf{n})$ for each natural number n. Let T be a theory in the language of arithmetic extending **Q**. Show:

(a) If T proves any incorrect ∀-rudimentary sentence, then T is inconsistent.

(b) If T proves any incorrect ∃-rudimentary sentence, then T is ω-inconsistent.

16.4 Extend Theorem 16.3 to generalized ∃-rudimentary sentences.

16.5 Let R be the set of triples (m, a, b) such that m codes a formula $\phi(x, y)$ and **Q** proves

$$\forall y(\phi(\mathbf{a}, y) \leftrightarrow y = \mathbf{b}).$$

Show that R is semirecursive.

16.6 For R as in the preceding problem, show that R is the graph of a two-place partial function.

16.7 A *universal function* is a two-place recursive partial function F such that for any one-place recursive total or partial function f there is an m such that $f(a) = F(m, a)$ for all a. Show that a universal function exists.

The result of the preceding problem was already proved in a completely different way (using the theory of Turing machines) in Chapter 8 *as Theorem* 8.5. *After completing the preceding problem, readers who skipped section* 8.3 *may turn to it, and to the related problems at the end of Chapter* 8.

16.8 A set P is (*positively*) *semidefinable* in a theory T by a formula $\phi(x)$ if for every n, $\phi(\mathbf{n})$ is a theorem of T if and only if n is in P. Show that every semirecursive set is (positively) semidefinable in **Q** and any ω-consistent extension of **Q**.

16.9 Let T be a consistent, axiomatizable theory containing **Q**. Show that:
(**a**) Every set (positively) semi-definable in T is semirecursive.
(**b**) Every set definable in T is recursive.
(**c**) Every total function representable in T is recursive.

16.10 Using the recursion equations for addition, prove:
(**a**) $x + (y + 0) = (x + y) + 0$.
(**b**) $x + (y + z) = (x + y) + z \rightarrow x + (y + z') = (x + y) + z'$.
The *associative law* for addition,

$$x + (y + z) = (x + y) + z$$

then follows by mathematical induction (‘on z’). (You may argue informally, as at the beginning of section 16.3. The proofs can be ‘formalized’ in **P**, but we are not asking you to do so.)

16.11 Continuing the preceding problem, prove:
(**c**) $x' + y = (x + y)'$,
(**d**) $x + y = y + x$.
The latter is the *commutative law* for addition.

16.12 Continuing the preceding problems, prove the *associative* and *distributive* and *commutative laws* for multiplication:
(**e**) $x \cdot (y + z) = x \cdot y + x \cdot z$,
(**f**) $x \cdot (y \cdot z) = (x \cdot y) \cdot z$,
(**g**) $x \cdot y = y \cdot x$.

16.13 (**a**) Consider the following nonstandard order relation on the natural numbers: $m <_1 n$ if and only if m is odd and n is even, or m and n have the same parity (are both odd or both even) and $m < n$. Show that if there is a natural number having a property P then there is a $<_1$-least such natural number.
(**b**) Consider the following order on pairs of natural numbers: $(a, b) <_2 (c, d)$ if and only if either $a < c$ or both $a = c$ and $b < d$. Show that if there is a pair of natural numbers having a property P then there is a $<_2$-least such pair.
(**c**) Consider the following order on finite sequences of natural numbers: $(a_0, \ldots, a_m) <_3 (b_0, \ldots, b_n)$ if and only if either $m < n$ or both $m = n$ and the following condition holds: that either $a_m < b_m$ or else for some $i < m$, $a_i < b_i$ while for $j > i$ we have $a_j = b_j$. Show that if there is a sequence of natural numbers having a property P then there is a $<_3$-least such sequence.

16.14 Consider a nonstandard interpretation of the language $\{\mathbf{0}, ', <\}$ in which the domain is the set of natural numbers, but the denotation of $<$ is taken to be the relation $<_1$ of Problem 16.13(*a*). Show that by giving suitable denotations to **0** and $'$, axioms (Q1)–(Q2) and (Q7)–(Q10) of **Q** can be made true, while the sentence $\forall x(x = \mathbf{0} \vee \exists y\ x = y')$ is made false.

16.15 Consider a nonstandard interpretation of the language $\{\mathbf{0}, ', <, +\}$ in which the domain is the set of pairs of natural numbers, and the denotation of $<$ is taken to be the relation $<_2$ of Problem 16.13(*b*). Show that by giving suitable denotations to **0** and $'$ and $+$, axioms (Q1)–(Q4) and (Q7)–(Q10) of **Q** can be made true, while both the sentence of the preceding problem and the sentence $\forall y(\mathbf{1} + y = y + \mathbf{1})$ are made false.

16.16 Consider an interpretation of the language $\{\mathbf{0}, ', +, \cdot, <\}$ in which the domain is the set of natural numbers plus one additional object called ∞, where the relations and operations on natural numbers are as usual, $\infty' = \infty$, $x + \infty = \infty + x = \infty$ for any x, $0 \cdot \infty = \infty \cdot 0 = 0$ but $x \cdot \infty = \infty \cdot x = \infty$ for any $x \neq 0$, and $x < \infty$ for all x, but not $\infty < y$ for any $y \neq \infty$. Show that axioms (Q0)–(Q9) and (Q11) are true on this interpretation, but not axiom (Q10).

16.17 Show that, as asserted in the proof of Lemma 16.14, for each m the following is a theorem of **Q**:

$$\mathbf{m} < y \leftrightarrow (y \neq \mathbf{0} \,\&\, y \neq \mathbf{1} \,\&\, \ldots \,\&\, y \neq \mathbf{m}).$$

16.18 Show that if the induction axioms are added to (Q1)–(Q8), then (Q9) and (Q10) become theorems. *The following problems pertain to the optional section 16.4.*

16.19 Show that the following are theorems of **R** for any b:

(a) $x' + \mathbf{b} = x + \mathbf{b}'$.

(b) $\mathbf{b} < x \rightarrow \mathbf{b}' < x'$.

(c) $x' < y' \rightarrow x < y$.

16.20 Show that the following are theorems of **R** for any b:

(a) $x < \mathbf{b}' \rightarrow x < \mathbf{b} \vee x = \mathbf{b}$.

(b) $\mathbf{b} < y \,\&\, y \neq \mathbf{b}' \rightarrow \mathbf{b}' < y$.

16.21 Show that adding induction to **R** produces the same theory (Peano arithmetic **P**) as adding induction to **Q**.

17

Indefinability, Undecidability, Incompleteness

We are now in a position to give a unified treatment of some of the central negative results of logic: Tarski's theorem *on the indefinability of truth,* Church's theorem *on the undecidability of logic, and* Gödel's first incompleteness theorem, *according to which, roughly speaking, any sufficiently strong formal system of arithmetic must be incomplete (if it is consistent). These theorems can all be seen as more or less direct consequences of a single exceedingly ingenious lemma, the* Gödel diagonal lemma. *This lemma, and the various negative results on the limits of logic that follow from it, will be presented in section 17.1. This presentation will be followed by a discussion in section 17.2 of some classic particular examples of sentences that can be neither proved nor disproved in theories of arithmetic like* **Q** *or* **P**. *Further such examples will be presented in the optional section 17.3. According to* Gödel's second incompleteness theorem, *the topic of the next chapter, such examples also include the sentence stating that* **P** *is consistent.*

17.1 The Diagonal Lemma and the Limitative Theorems

By the results in the preceding chapter on the representability of recursive functions, we can 'talk about' such functions within a formal system of arithmetic. By the results of the chapter before that on the arithmetization of syntax, we can 'talk about' sentences and proofs in a formal system of arithmetic in terms of recursive functions. Putting the two together, we can 'talk about' sentences and proofs in a formal system of arithmetic *within the formal system of arithmetic itself*. This is the key to the main lemma of this section, the diagonal lemma.

Until further notice, all formulas, sentences, theories, and so on, will be formulas, sentences, theories, or whatever in the language of arithmetic. Given any expression A of the language of arithmetic, we have introduced in Chapter 15 a code number for A, called the Gödel number of A. If a is this number, then the numeral $\mathbf{a}$ for a, consisting of $\mathbf{0}$ followed by a accents $'$, is naturally called the *Gödel numeral* for A. We write $\ulcorner A \urcorner$ for this code numeral for A. In what follows, $\ulcorner A \urcorner$ will be seen to function somewhat like a name for A.

We define the *diagonalization* of A to be the expression $\exists x(x = \ulcorner A \urcorner \,\&\, A)$. While this notion makes sense for arbitrary expressions, it is of most interest in the case of a formula $A(x)$ with just the one variable x free. Since in general $F(t)$ is equivalent to $\exists x(x = t \,\&\, F(x))$, in case A is such a formula, the diagonalization of A is a sentence equivalent to $A(\ulcorner A \urcorner)$, the result of substituting the code numeral for A itself for the free variable in A. In this case the diagonalization 'says that' A is satisfied by its own Gödel number, or more precisely, the diagonalization will be true in the standard interpretation if and only if A is satisfied by its own Gödel number in the standard interpretation.

17.1 Lemma (Diagonal lemma). Let T be a theory containing **Q**. Then for any formula $B(y)$ there is a sentence G such that $\vdash_T G \leftrightarrow B(\ulcorner G \urcorner)$.

Proof: There is a (primitive) recursive function, diag, such that if a is the Gödel number of an expression A, then diag(a) is the Gödel number of the diagonalization of A. Indeed, we have seen almost exactly the function we want before, in the proof of Corollary 15.6. Recalling that officially $x = y$ is supposed to be written $=(x, y)$, it can be seen to be

$$\mathrm{diag}(y) = \mathrm{exquant}(v, \mathrm{conj}(i * l * v * c * \mathrm{num}(y) * r, y))$$

where v is the code number for the variable, i, l, r, and c for the equals sign, left and right parentheses, and comma, and exquant, conj, and num are as in Proposition 15.1 and Corollary 15.6.

If T is a theory extending **Q**, then diag is representable in T by Theorem 16.16. Let Diag(x, y) be a formula representing diag, so that for any m and n, if diag $(m) = n$, then $\vdash_T \forall y(\mathrm{Diag}(\mathbf{m}, y) \leftrightarrow y = \mathbf{n})$.

Let $A(x)$ be the formula $\exists y(\mathrm{Diag}(x, y) \,\&\, B(y))$. Let a be the Gödel number of $A(x)$, and **a** its Gödel numeral. Let G be the sentence $\exists x(x = \mathbf{a} \,\&\, \exists y(\mathrm{Diag}(x, y) \,\&\, B(y)))$.

Thus G is $\exists x(x = \mathbf{a} \,\&\, A(x))$, and is logically equivalent to $A(\mathbf{a})$ or $\exists y(\mathrm{Diag}(\mathbf{a}, y) \,\&\, B(y))$. The biconditional $G \leftrightarrow \exists y(\mathrm{Diag}(\mathbf{a}, y) \,\&\, B(y))$ is therefore valid, and as such provable in any theory, so we have

$$\vdash_T G \leftrightarrow \exists y(\mathrm{Diag}(\mathbf{a}, y) \,\&\, B(y)).$$

Let g be the Gödel number of G, and **g** its Gödel numeral. Since G is the diagonalization of $A(x)$, diag$(a) = g$ and so we have

$$\vdash_T \forall y(\mathrm{Diag}(\mathbf{a}, y) \leftrightarrow y = \mathbf{g}).$$

It follows that

$$\vdash_T G \leftrightarrow \exists y(y = \mathbf{g} \,\&\, B(y)).$$

Since $\exists y(y = \mathbf{g} \,\&\, B(y))$ is logically equivalent to $B(\mathbf{g})$, we have

$$\vdash_T \exists y(y = \mathbf{g} \,\&\, B(y)) \leftrightarrow B(\mathbf{g}).$$

It follows that

$$\vdash_T G \leftrightarrow B(\mathbf{g})$$

or in other words, $\vdash_T G \leftrightarrow B(\ulcorner G \urcorner)$, as required.

17.2 Lemma. Let T be a consistent theory extending **Q**. Then the set of Gödel numbers of theorems of T is not definable in T.

Proof: Let T be an extension of **Q**. Suppose $\theta(y)$ defines the set Θ of Gödel numbers of sentences in T. By the diagonal lemma there is a sentence G such that

$$\vdash_T G \leftrightarrow {\sim}\theta(\ulcorner G \urcorner).$$

In other words, letting g be the Gödel number of G, and **g** its Gödel numeral, we have

$$\vdash_T G \leftrightarrow {\sim}\theta(\mathbf{g}).$$

Then G is a theorem of T. For if we assume G is not a theorem of T, then g is not in Θ, and since $\theta(y)$ defines Θ, we have $\vdash_T {\sim}\theta(\mathbf{g})$; but then since $\vdash_T G \leftrightarrow {\sim}\theta(\mathbf{g})$, we have $\vdash_T G$ and G is a theorem of T after all. But since G is a theorem of T, g is in Θ, and so we have $\vdash_T \theta(\mathbf{g})$; but then, since $\vdash_T G \leftrightarrow {\sim}\theta(\mathbf{g})$, we have $\vdash_T {\sim}G$, and T is inconsistent.

Now the 'limitative theorems' come tumbling out in rapid succession.

17.3 Theorem (Tarski's theorem). The set of Gödel numbers of sentences of the language of arithmetic that are correct, or true in the standard interpretation, is not arithmetically definable.

Proof: The set T in question is the theory we have been calling true arithmetic. It is a consistent extension of **Q**, and arithmetic definability is simply definability in this theory, so the theorem is immediate from Lemma 17.2.

17.4 Theorem (Undecidability of arithmetic). The set of Gödel numbers of sentences of the language of arithmetic that are correct, or true in the standard interpretation, is not recursive.

Proof: This follows from Theorem 17.3 and the fact that all recursive sets are definable in arithmetic.

Assuming Church's thesis, this means that the set in question is not effectively decidable: there are no rules—of a kind requiring only diligence and persistence, not ingenuity and insight, to execute—with the property that applied to any sentence of the language of arithmetic they will eventually tell one whether or not it is correct.

17.5 Theorem (Essential undecidability theorem). No consistent extension of **Q** is decidable (and in particular, **Q** itself is undecidable).

Proof: Suppose T is a consistent extension of **Q** (in particular, T could just be **Q** itself). Then by Lemma 17.2, the set Θ of Gödel numbers of theorems of T is not definable in T. Now, again as in the proof of Theorem 17.4, we invoke the fact that every recursive set is definable in T. So the set Θ is not recursive, which is to say T is not decidable.

17.6 Theorem (Church's theorem). The set of valid sentences is not decidable.

> *Proof*: Let C be the conjunction of all the axioms of **Q**. Then a sentence A is a theorem of **Q** if and only if A is a consequence of C, hence if and only if $(\sim C \vee A)$ is valid. The function f taking the Gödel number of A to that of $(\sim C \vee A)$ is recursive [it being simply $f(y) = \text{disj}(\text{neg}(c), y)$, in the notation of the proof of Proposition 15.1]. If the set Λ of logically valid sentences were recursive, the set K of Gödel numbers of theorems of **Q** would be obtainable from it by substitution of the recursive function f, since a is in K if and only if $f(n)$ is in Λ, and so would be recursive, which it is not by Theorem 17.4.

The sets of valid sentences, and of theorems of any axiomatizable theory, are semirecursive by Corollaries 15.4 and 15.5, and intuitively, of course, both are *positively* effectively decidable: in principle, if not in practice, just by searching through all demonstrations (or all proofs from the axioms of the theory), *if* a sentence is valid (or a theorem of the theory), one will eventually find that out. But Theorems 17.5 and 17.6 tell us these sets are not recursive, and so by Church's thesis are not effectively decidable.

17.7 Theorem (Gödel's first incompleteness theorem). There is no consistent, complete, axiomatizable extension of **Q**.

> *Proof*: Any complete axiomatizable theory is decidable by Corollary 15.7, but no consistent extension of **Q** is decidable by Theorem 17.5 above.

The import of Gödel's first incompleteness theorem is sometimes expressed in the words 'any sufficiently strong formal system of arithmetic (or mathematics) is incomplete, unless it is inconsistent'. Here by 'formal system' is meant a theory whose theorems are derivable by the rules of logical derivation from a set of axioms that is effectively decidable, and hence (assuming Church's thesis) recursive. So 'formal system' amounts to 'axiomatizable theory', and 'formal system of arithmetic' to 'axiomatizable theory in the language of arithmetic'. Gödel's first incompleteness theorem in the version in which we have given it indicates a sufficient condition for being 'sufficiently strong', namely, being an extension of **Q**. Since **Q** is a comparatively weak theory, this version of Gödel's first incompleteness theorem is a correspondingly strong result.

Now a formal system of mathematics might well be such that the domain of its intended interpretation was a more inclusive set than the set of natural numbers, and it might well be such that it did not have symbols specifically for 'less than' and the other items for which there are symbols in the language of arithmetic. So the principle that any two natural numbers are comparable as to ordering might not be expressed, as it is in the axioms of **Q**, by the sentence

$$\forall x \forall y (x < y \vee x = y \vee y < x).$$

Still, it is reasonable to understand ‘sufficiently strong’ as implying that this principle can be *somehow* expressed in the language of the theory, perhaps by a sentence

$$\forall x(N(x) \to \forall y(N(y) \to (L(x, y) \vee x = y \vee L(y, x))))$$

where $N(x)$ appropriately expresses ‘x is a natural number’ and $L(x, y)$ appropriately expresses ‘x is less than y’. Moreover, the sentence that thus ‘translates’ this or any axiom of **Q** should be a theorem of the theory. Such is the case, for instance, with the formal systems considered in works on set theory, such as the one known as **ZFC**, which are adequate for formalizing essentially all accepted mathematical proofs. When the notion of ‘translation’ is made precise, it can be shown that any ‘sufficiently strong’ formal system of mathematics in the sense we have been indicating is still subject to the limitative theorems of this chapter. In particular, if consistent, it will be incomplete.

Perhaps the most important implication of the incompleteness theorem is what is says about the notions of *truth* (in the standard interpretation) and *provability* (in a formal system): *that they are in no sense the same*.

17.2 Undecidable Sentences

A sentence in the language of a theory T is said to be *disprovable in T* if its negation is provable in T, and is said to be *undecidable in* or *by* or *for T* if it is neither provable nor disprovable in T. (Do not confuse the notion of an undecidable sentence with that of an undecidable theory. True arithmetic, for example, is an undecidable theory with no undecidable sentences: the sentences of its language that are true in the standard interpretation all being provable, and those that are false all being disprovable.) If T is a theory in the language of arithmetic that is consistent, axiomatizable, and an extension of **Q**, then T is an undecidable theory by Theorem 17.4, and there exist undecidable sentences for T by Theorem 17.7. Our proof of the latter theorem did not, however, exhibit any explicit example of a sentence that is undecidable for T. An immediate question is: can we find any such specific examples?

In order to do so, we use the fact that the set of sentences that are provable and the set of sentences that are disprovable from any recursive set of axioms is semirecursive, and that all recursive sets are definable by ∃-rudimentary formulas. It follows that there are formulas $\mathrm{Prv}_T(x)$ and $\mathrm{Disprv}_T(x)$ of forms $\exists y\, \mathrm{Prf}_T(x, y)$ and $\exists y\, \mathrm{Disprf}_T(x, y)$ respectively, with Prf and Disprf rudimentary, such that $\vdash_T A$ if and only if the sentence $\mathrm{Prv}_T(\ulcorner A \urcorner)$ is correct or true in the standard interpretation, and hence if and only if for some b the sentence $\mathrm{Prf}_T(\ulcorner A \urcorner, \mathbf{b})$ is correct or—what is equivalent for rudimentary sentences—provable in **Q** and in T; and similarly for disprovability. $\mathrm{Prf}_T(x, y)$ could be read ‘y is a witness to the provability of x in T’.

By the diagonalization lemma, there is a sentence G_T such that

$$\vdash_T G_T \leftrightarrow \sim\exists y\, \mathrm{Prf}_T(\ulcorner G_T \urcorner, y)$$

and a sentence R_T such that

$$\vdash_T R_T \leftrightarrow \forall y(\mathrm{Prf}_T(\ulcorner R_T \urcorner, y) \to \exists z < y\, \mathrm{Disprf}(\ulcorner R_T \urcorner, z)).$$

Such a G_T is called a *Gödel sentence* for T, and such an R_T a *Rosser sentence* for T. Thus a Gödel sentence 'says of itself that' it is unprovable, and a Rosser sentence 'says of itself that' if there is a witness to its provability, then there is an earlier witness to its disprovability.

17.8 Theorem. Let T be a consistent, axiomatizable extension of **Q**. Then a Rosser sentence for T is undecidable for T.

Proof: Suppose the Rosser sentence R_T is provable in T. Then there is some a that witnesses the provability of R_T. Since T is consistent, $\sim R_T$ is not also provable, and so no m witnesses the disprovability of R_T, and in particular, no $m < n$ does so. It follows that the rudimentary sentence

$$\mathrm{Prf}_T(\ulcorner R_T \urcorner, \mathbf{n})\,\&\sim\exists z < \mathbf{n}\ \mathrm{Disprf}_T(\ulcorner R_T \urcorner, z)$$

is correct and as such is provable from the axioms of **Q**, and hence from T. In other words, we have

$$\vdash_T \mathrm{Prf}_T(\ulcorner R_T \urcorner, \mathbf{n})\,\&\sim\exists z < \mathbf{n}\ \mathrm{Disprf}_T(\ulcorner R_T \urcorner, z)$$

while, since R_T is a Rosser sentence, we also have

$$\vdash_T R_T \leftrightarrow \forall y(\mathrm{Prf}_T(\ulcorner R_T \urcorner, y) \to \exists z < y\ \mathrm{Disprf}(\ulcorner R_T \urcorner, z)).$$

By pure logic it follows that

$$\vdash_T \sim R_T.$$

But then T is inconsistent, both R_T and $\sim R_T$ being provable, contrary to assumption. This contradiction shows that R_T cannot be provable.

Suppose the Rosser sentence R_T is disprovable in T. Then there is some m that witnesses the disprovability of R_T. Since T is consistent, R_T is not also provable, and so no n witnesses the provability of R_T, and in particular, no $n \leq m$ does so. It follows that the rudimentary formulas

$$\mathrm{Disprf}_T(\ulcorner R_T \urcorner, \mathbf{m})$$
$$\forall x((x < \mathbf{m} \vee x = \mathbf{m}) \to \sim\mathrm{Prf}_T(\ulcorner R_T \urcorner, x))$$

are correct and hence provable in T. In other words we have

$$\vdash_T \mathrm{Disprf}_T(\ulcorner R_T \urcorner, \mathbf{m}),$$
$$\vdash_T \forall y((y < \mathbf{m} \vee y = \mathbf{m}) \to \sim\mathrm{Prf}_T(\ulcorner R_T \urcorner, y)).$$

By pure logic it follows from the former of these that

$$\vdash_T \forall y(\mathbf{m} < y \to \exists z < y\ \mathrm{Disprf}_T(\ulcorner R_T \urcorner, z)).$$

As a theorem of **Q** we also have

$$\vdash_T \forall y(y < \mathbf{m} \vee y = \mathbf{m} \vee \mathbf{m} < y).$$

It follows by pure logic that

$$\vdash_T \forall y(\mathrm{Prf}_T(\ulcorner R_T \urcorner, y) \rightarrow \exists z < y\ \mathrm{Disprf}(\ulcorner R_T \urcorner, z))$$

and hence $\vdash_T R_T$, and T is inconsistent, a contradiction that shows that R_T cannot be disprovable in T.

A theory T is called *ω-inconsistent* if and only if for some formula $F(x)$, $\vdash_T \exists x F(x)$ but $\vdash_T \sim F(\mathbf{n})$ for every natural number n, and is called *ω-consistent* if and only if it is not ω-inconsistent. Thus an ω-inconsistent theory 'affirms' there is some number with the property expressed by F, but then 'denies' that zero is such a number, that one is such a number, than two is such a number, and so on. Since $\exists x F(x)$ and $\sim F(\mathbf{0})$, $\sim F(\mathbf{1})$, $\sim F(\mathbf{2})$, . . . cannot all be correct, any ω-inconsistent theory must have some incorrect theorems. But an ω-inconsistent theory need not be inconsistent. (An example of a consistent but ω-inconsistent theory will be given shortly.)

17.9 Theorem. Let T be a consistent, axiomatizable extension of **Q**. Then a Gödel sentence for T is unprovable in T, and if T is ω-consistent, it is also undisprovable in T.

Proof: Suppose the Gödel sentence G_T is provable in T. Then the $\exists$-rudimentary sentence $\exists y\ \mathrm{Prf}_T(\ulcorner G_T \urcorner, y)$ is correct, and so provable in T. But since G_T is a Gödel sentence, $G_T \leftrightarrow \sim\exists y\ \mathrm{Prf}_T(\ulcorner G_T \urcorner, y)$ is also provable in T. By pure logic it follows that $\sim G_T$ is provable in T, and T is inconsistent, a contradiction, which shows that G_T is not provable in T.

Suppose the sentence G_T is disprovable in T. Then $\sim\sim\exists y\ \mathrm{Prf}_T(\ulcorner G_T \urcorner, y)$ and hence $\exists y\ \mathrm{Prf}_T(\ulcorner G_T \urcorner, y)$ is provable in T. But by consistency, G_T is not provable in T, and so for any n, n is not a witness to the provability of G_T, and so the rudimentary sentence $\sim\mathrm{Prf}_T(\ulcorner G_T \urcorner, \mathbf{n})$ is correct and hence provable in **Q** and hence in T. But this means T is ω-inconsistent, a contradiction, which shows that G_T is not disprovable in T.

For an example of a consistent but ω-inconsistent theory, consider the theory $T = \mathbf{Q} + \sim G_{\mathbf{Q}}$ consisting of all consequences of the axioms of **Q** together with $\sim G_{\mathbf{Q}}$ or $\sim\sim\exists y\ \mathrm{Prf}_{\mathbf{Q}}(G_{\mathbf{Q}}, y)$. Since $G_{\mathbf{Q}}$ is not a theorem of **Q**, this theory T is consistent. Of course $\exists y\ \mathrm{Prf}_{\mathbf{Q}}(G_{\mathbf{Q}}, y)$ is a theorem of T. But for any particular n, the rudimentary sentence $\sim\mathrm{Prf}_{\mathbf{Q}}(G_{\mathbf{Q}}, \mathbf{n})$ is correct, and therefore provable in any extension of **Q**, including T.

Historically, Theorem 17.9 came first, and Theorem 17.8 was a subsequent refinement. Accordingly, the Rosser sentence is sometimes called the *Gödel–Rosser sentence*. Subsequently, many other examples of undecidable sentences have been brought forward. Several interesting examples will be discussed in the following, optional section, and the most important example in the next chapter.

17.3* Undecidable Sentences without the Diagonal Lemma

The diagonal lemma, which was used to construct the Gödel and Rosser sentences, is in some sense the cleverest idea in the proof of the first incompleteness theorem, and is heavily emphasized in popularized accounts. However, the possibility of implementing the idea of this lemma, of constructing a sentence that says of itself that it is unprovable, depends on the apparatus of the arithmetization of syntax and the representability of recursive functions. Once that apparatus is in place, a version of the incompleteness theorem, showing the existence of a true but unprovable sentence, can be established *without* the diagonal lemma. One way to do so is indicated in the first problem at the end of this chapter. (This way uses the fact that there exist semirecursive sets that are not recursive, and though it does not use the diagonal *lemma*, does involve a diagonal *argument*, buried in the proof of the fact just cited.) Some other ways will be indicated in the present section.

Towards describing one such way, recall the *Epimenides* or *liar paradox*, involving the sentence 'This sentence is untrue'. A contradiction arises when we ask whether this sentence is true: it seems that it is if and only if it isn't. The Gödel sentence in effect results from this paradoxical sentence on substituting 'provable' for 'true' (a substitution that is crucial for establishing that we can actually construct a Gödel sentence in the language of arithmetic). Now there are other *semantic paradoxes*, paradoxes in the same family as the liar paradox, involving other semantic notions related to truth. One famous one is the *Grelling* or *heterological* paradox. Call an adjective *autological* if it is true of itself, as 'short' is short, 'polysyllabic' is polysyllabic, and 'English' is English, and call it *heterological* if it is untrue of itself, as 'long' is not long, 'monosyllabic' is not monosyllabic, and 'French' is not French. A contradiction arises when we ask whether 'heterological' is heterological: it seems that it is if and only if it isn't.

Let us modify the definition of heterologicality by substituting 'provable' for 'true'. We then get the notion of *self-applicability*: a number m is self-applicable in **Q** if it is the Gödel number of a formula $\mu(x)$ such that $\mu(\mathbf{m})$ is provable in **Q**. Now the same apparatus that allowed us to construct the Gödel sentence allows us to construct what may be called the *Gödel–Grelling formula* GG(x) expressing 'x is not self-applicable'. Let m be its Gödel number. If m were self-applicable, then GG($\mathbf{m}$) would be provable, hence true, and since what it expresses is that m is *not* self-applicable, this is impossible. So m is not self-applicable, and hence GG($\mathbf{m}$) is true but unprovable.

Another semantic paradox, *Berry's paradox*, concerns the least integer not namable in fewer than nineteen syllables. The paradox, of course, is that the integer in question appears to have been named just now in eighteen syllables. This paradox, too, can be adapted to give an example of a sentence undecidable in **Q**. Let us say that a number n is *denominable* in **Q** by a formula $\phi(x)$ if $\forall x(\phi(x) \leftrightarrow x = \mathbf{n})$ is (not just true but) provable in **Q**.

Every number n is denominable in **Q**, since if worse comes to worst, it can always be denominated by the formula $x = \mathbf{n}$, a formula with $n + 3$ symbols. Some numbers n are denominable in **Q** by formulas with far fewer than n symbols. For example, the number $10 \Uparrow 10$ is denominable by the formula $\phi(\mathbf{10}, \mathbf{10}, x)$, where ϕ is a formula representing the super-exponential function $\Uparrow$. We have not actually written out this formula, but instructions for doing so are implicit in the proof that all recursive functions are representable, and review of that proof reveals that writing out the formula would not take more time or more paper than an ordinary homework assignment. By contrast, $10 \Uparrow 10$ is larger than the number of particles in the visible universe. But while big numbers can thus be denominated by comparatively short formulas, for any fixed k, only finitely many numbers can be denominated by formulas with fewer than

k symbols. For logically equivalent formulas denominate the same number (if they denominate any number at all), and every formula with fewer than k symbols is logically equivalent, by relettering bound variables, to one with only the first k variables on our official list of variables, and there are only finitely many of those.

Thus, there will be numbers not denominable using fewer than $10 \Uparrow 10$ symbols. The usual apparatus allows us to construct a *Gödel–Berry formula* $\mathrm{GB}(x, y)$, expressing 'x is the least number not denominable by a formula with fewer than $y \Uparrow y$ symbols'. Writing out this formula would involve writing out not just the formula representing the super-exponential function $\Uparrow$, but also the formulas relating to provability in **Q**. Again we have not actually written out these formulas, but only given an outline of how to do so in our proofs of the arithmetizability of syntax and the representability of recursive functions in **Q**. Review of those proofs reveals that writing out the formula $\mathrm{GB}(x, y)$ or $\mathrm{GB}(x, \mathbf{10})$, though it would require more time and paper than any reasonable homework assignment, would not require more symbols than appear in an ordinary encyclopedia, which is far fewer than the astronomical figure $10 \Uparrow 10$. Now there is some number not denominable by a formula with fewer symbols than that astronomical figure, and among such numbers there is one and only one least, call it n. Then $\mathrm{GB}(\mathbf{n}, \mathbf{10})$ and $\forall x(\mathrm{GB}(x, \mathbf{10}) \leftrightarrow x = \mathbf{n})$ are true. But if the latter were provable, the formula $\mathrm{GB}(x, \mathbf{10})$ would denominate n, whereas n is not denominable except by formulas much longer than that. Hence we have another example of an unprovable truth.

This example is worth pressing a little further. The length of the shortest formula denominating a number may be taken as a measure of the *complexity* of that number. Just as we could construct the Gödel–Berry formula, we can construct a formula $C(x, y, z)$ expressing 'the complexity of x is y and y is greater than $z \Uparrow z$', and using it the *Gödel–Chaitin* formula $\mathrm{GC}(x)$ or $\exists y C(x, y, \mathbf{10})$, expressing that x has complexity greater than $10 \Uparrow 10$. Now for all but finitely many n, $\mathrm{GC}(\mathbf{n})$ is true. *Chaitin's theorem* tells us that no sentence of form $\mathrm{GC}(\mathbf{n})$ is provable.

The reason may be sketched as follows. Just as 'y is a witness to the provability of x in **Q**' can be expressed in the language of arithmetic by a formula $\mathrm{Prf}_{\mathbf{Q}}(x,y)$, so can '$y$ is a witness to the provability of the result of subsituting the numeral for x for the variable in GC' be expressed by a formula $\mathrm{PrfGC}_{\mathbf{Q}}(x,y)$. Now if any sentence of form $\mathrm{GC}(\mathbf{n})$ can be proved, there is a least m such that m witnesses the provability of $\mathrm{GC}(\mathbf{n})$ for some n. Let us call m the 'lead witness' for short. And of course, since any one number witnesses the provability of at most one sentence, there will be a least n—in fact, there will be one and only one n—such that the lead witness is a witness to the provability of $\mathrm{GC}(\mathbf{n})$. Call n the number 'identified by the lead witness' for short.

If one is careful, one can arrange matters so that the sentences $K(\mathbf{m})$ and $L(\mathbf{n})$ expressing 'm is the lead witness' and 'n is the number identified by the lead witness' will be $\exists$-rudimentary, so that, being true, $K(\mathbf{m})$ and $L(\mathbf{n})$ will be provable. Moreover, since it can be proved in **Q** that there is at most *one* least number fulfilling the condition expressed by any formula, $\forall x(x \neq \mathbf{m} \rightarrow \sim K(x))$ and $\forall x(x \neq \mathbf{n} \rightarrow \sim L(x))$ will also be provable. But this means that n is denominated by the formula $L(x)$, and hence has complexity less than the number of symbols in that formula. And though it might take an encyclopedia's worth of paper and ink to write the formula down, the number of symbols in any encyclopedia remains far less than $10 \Uparrow 10$. So if n is denominated by the formula $L(x)$, its complexity is less than $10 \Uparrow 10$. Since this is impossible, it follows that no sentence of form GC(**n**) can be proved: no specific number n can be proved to have complexity greater than $10 \Uparrow 10$. This reasoning can be adapted to any other reasonable measure of complexity.

(For example, suppose we take the complexity of a number to be the smallest number of states needed for a Turing machine that will produce that number as output given zero as input. To establish that 'the complexity of x is y' and related formulas can be expressed in the language of arithmetic we now need the fact that Turing machines can be coded by recursive functions in addition to the fact that recursive functions are representable. And to show that if there is any proof that some number has complexity greater than $10 \Uparrow 10$, then the number n identified by the lead witness can be generated as the output for input zero by some Turing machine, we need in addition to the arithmetizability of syntax the fact also of the Turing computability of recursive functions. Almost the whole of this book up to this point is involved just in *outlining* how one would go about writing down the relevant formula and designing the relevant Turing machine. But while filling in the details of this outline might fill an encyclopedia, still it would not require anything approaching $10 \Uparrow 10$ symbols, and that is all that is essential to the argument. In the literature, the label *Chaitin's theorem* refers especially to this Turing-machine version, but as we have said, similar reasoning applies to any reasonable notion of complexity.)

Thus on any reasonable measure of complexity, there is an upper bound b—we have used $10 \Uparrow 10$, though a closer analysis would show that a much smaller number would do, its exact value depending on the particular measure of complexity being used—such that *no* specific number n can be proved in **Q** to have complexity greater than b. Moreover, this applies not just to **Q** but to any stronger true theory, such as **P** or the theories developed in works on set theory that are adequate for formalizing essentially all ordinary mathematical proofs. Thus Chaitin's theorem, whose proof we have sketched, tells us that *there is an upper bound such that no specific number can be proved by ordinary mathematical means to have complexity greater than that bound.*

Problems

17.1 Show that the existence of a semirecursive set that is not recursive implies that any consistent, axiomatizable extension of **Q** fails to prove some correct ∀-rudimentary sentence.

17.2 Let T be a consistent, axiomatizable theory extending **Q**. Consider the set P^{yes} of (code numbers of) formulas that are provable in T, and the set P^{no} of (code numbers of) formulas that are *dis*provable in P. Show that there is no recursive set R such that P^{yes} is a subset of R while no element of R is an element of P^{no}.

17.3 Let $B_1(y)$ and $B_2(y)$ be two formulas of the language of arithmetic. Generalizing the diagonal lemma, show that there are sentences G_1 and G_2 such that

$$\vdash_{\mathbf{Q}} G_1 \leftrightarrow B_2(\ulcorner G_2 \urcorner)$$
$$\vdash_{\mathbf{Q}} G_2 \leftrightarrow B_1(\ulcorner G_1 \urcorner).$$

For instance, there are a pair of sentences such that the first says the second is provable, while the second says the first is unprovable.

The set of (code numbers of) sentences of the language of arithmetic $\{<, \mathbf{0}, {}', +, \cdot\}$ *that are* correct, *or true in the standard interpretation, is not recursive. Actually, it can be shown that the set of (code numbers of) sentences of the language* $\{+, \cdot\}$ *that are true in the standard interpretation is not recursive. The next few problems are pieces of the proof.*

17.4 Explain why, to establish the stronger result just stated, it would suffice to associate in a recursive way to every sentence A of the language $\{<, \mathbf{0}, {}', +, \cdot\}$ a sentence $A^\dagger$ of the language $\{+, \cdot\}$ such that A is correct if and only if $A^\dagger$ is correct.

17.5 Continuing the preceding problem, show that there is a formula $D_0(x)$ of the language $\{+, \cdot\}$ such that the following is correct: $\forall x(x = \mathbf{0} \leftrightarrow D_0(x))$. Then explain how to associate in an effective (and therefore, assuming Church's thesis, a recursive) way to every sentence A of the language $\{<, \mathbf{0}, {}', +, \cdot\}$ a sentence $A^\dagger$ of the language $\{<, {}', +, \cdot\}$ such that A is correct if and only if $A^\dagger$ is correct.

17.6 Continuing the preceding series of problems, exhibit a formula $D_s(x, y)$ of the language $\{+, \cdot\}$ such that

$$\forall x \forall y(x' = y \leftrightarrow D_s(x, y))$$

is correct, and a formula $D_<(x, y)$ of the language $\{+, \cdot\}$ such that

$$\forall x \forall y(x < y \leftrightarrow D_<(x, y))$$

is correct. Then show how, say, the statements in Example 16.7 can be naturally expressed in the language $\{+, \cdot\}$.

17.7 Let $T = \mathbf{Q}$, let R be the Rosser sentence of T, let T_0 be $T + \{R\}$, the set of consequences of $T \cup \{R\}$, and let $T_1 = T + \{\sim R\}$; then $\{T_0, T_1\}$ is a set of two consistent, axiomatizable extensions of $\mathbf{Q}$ that are inconsistent with each other in the sense that their union is inconsistent. Show that for every n there is a set of 2^n consistent, axiomatizable extensions of $\mathbf{Q}$ that are *pairwise inconsistent* in the sense that any two of them are inconsistent with each other.

17.8 Show that there is a nonenumerable set of consistent extensions of $\mathbf{Q}$ that are pairwise inconsistent.

17.9 Let L_1 and L_2 be finite or recursive languages, and T a theory in L_2. A *translation* of L_1 into T is an assignment to each sentence S of L_1 of a sentence $S^\dagger$ of L_2 such that:

(T0) $(\sim A)^\dagger$ is logically equivalent to $\sim(A)^\dagger$.

(T1) The function taking the code number of a sentence of L_1 to the code number of its translation is recursive.

(T2) Whenever $A_1, \ldots, A_k, B$ are sentences of L_1 and B is a consequence of $A_1, \ldots, A_k$, then $B^\dagger$ is a consequence of $T \cup \{A_1^\dagger, \ldots, A_k^\dagger\}$.

Show that if T is a consistent, axiomatizable theory in a language L, and if there is a translation of the language of arithmetic into T such that the translation of every axiom of **Q** is a theorem of T, then the set of sentences of the language of arithmetic whose translations are theorems of T is a consistent, axiomatizable extension of **Q**.

17.10 Show that under the hypotheses of the preceding problem, T is incomplete and undecidable.

17.11 Let L be a language, $N(u)$ a formula of L. For any sentence F of L, let the *relativization* F^N be the result of replacing each universal quantifier $\forall x$ in F by $\forall x(N(x) \rightarrow \cdots)$ and each existential quantifier $\exists x$ by $\exists x(N(x) \,\&\, \ldots)$. Let T be a theory in L such that for every name c, $N(c)$ is a theorem of T and for every function symbol f the following is a theorem of T:

$$\forall x_1 \ldots \forall x_k((N(x_1) \,\&\, \ldots \,\&\, N(x_k)) \rightarrow N(f(x_1, \ldots, x_k))).$$

Show that for any model $\mathcal{M}$ of T, the set of a in $|\mathcal{M}|$ that satisfies $N(x)$ is the domain of an interpretation $\mathcal{N}$ such that any sentence S of L is true in $\mathcal{N}$ if and only if its relativization S^N is true in $\mathcal{M}$.

17.12 Continuing the preceding series of problem, show that the function assigning each sentence S of L its relativization S^N is a translation. (You may appeal to Church's thesis.)

17.13 Consider the interpretation $\mathcal{Z}$ of the language $\{<, \mathbf{0}, ', +, \cdot\}$ in which the domain is the set of all integers (including the negative ones), and the denotation of **0** is zero, of $'$ is the function that adds one to a number, of $+$ and $\cdot$ are the usual addition and multiplication functions, and of $<$ is the usual order relation. Show that the set of all sentences that are true in $\mathcal{Z}$ is undecidable, and that this is still so if $<$ is dropped.

18

The Unprovability of Consistency

According to Gödel's second incompleteness theorem, *the sentence expressing that a theory like* **P** *is consistent is undecidable by* **P**, *supposing* **P** *is consistent. The full proof of this result is beyond the scope of a book on the level of the present one, but the overall structure of the proof and main ingredients that go into the proof will be indicated in this short chapter. In place of problems there are some historical notes at the end.*

Officially we defined T to be inconsistent if every sentence is provable from T, though we know this is equivalent to various other conditions, notably that for some sentence S, both S and $\sim S$ are provable from T. If T is an extension of **Q**, then since $\mathbf{0} \neq \mathbf{1}$ is the simplest instance of the first axiom of **Q**, $\mathbf{0} \neq \mathbf{1}$ is provable from T, and if $\mathbf{0} = \mathbf{1}$ is also provable from T, then T is inconsistent; while if T is inconsistent, then $\mathbf{0} = \mathbf{1}$ is provable from T, since every sentence is. Thus T is consistent if and only if $\mathbf{0} = \mathbf{1}$ is *not* provable from T. We call $\sim\mathrm{Prv}_T(\ulcorner\mathbf{0} = \mathbf{1}\urcorner)$, which is to say $\sim\exists y\ \mathrm{Prf}_T(\ulcorner\mathbf{0} = \mathbf{1}\urcorner, y)$, the *consistency sentence* for T. Historically, the original paper of Gödel containing his original version of the first incompleteness theorem (corresponding to our Theorem 17.9) included towards the end a statement of a version of the following theorem.

18.1 Theorem* (Gödel's second incompleteness theorem, concrete form). Let T be a consistent, axiomatizable extension of **P**. Then the consistency sentence for T is not provable in T.

We have starred this theorem because we are not going to give a full proof of it. In gross outline, Gödel's idea for the proof of this theorem was as follows. The proof of Theorem 17.9 shows that if the absurdity $\mathbf{0} = \mathbf{1}$ is not provable in T then the Gödel sentence G_T is not provable in T either, so the following is true: $\sim\mathrm{Prv}_T(\ulcorner\mathbf{0} = \mathbf{1}\urcorner) \rightarrow \sim\mathrm{Prv}_T(\ulcorner G_T\urcorner)$. Now it turns out that the theory **P** of inductive arithmetic, and hence any extension T thereof, is strong enough to 'formalize' the proof of Theorem 17.9, so we have

$$\vdash_T \sim\mathrm{Prv}_T(\ulcorner\mathbf{0} = \mathbf{1}\urcorner) \rightarrow \sim\mathrm{Prv}_T(\ulcorner G_T\urcorner).$$

But G_T was a Gödel sentence, so we have also

$$\vdash_T G_T \leftrightarrow \sim\mathrm{Prv}_T(\ulcorner G_T\urcorner).$$

And so we have

$$\vdash_T \sim\mathrm{Prv}_T(\ulcorner \mathbf{0} = \mathbf{1} \urcorner) \to G_T.$$

So if we had $\vdash_T \sim\mathrm{Prv}_T(\ulcorner \mathbf{0} = \mathbf{1} \urcorner)$, then we would have $\vdash_T G_T$, which by Proposition 17.9 we do not.

Of course, the key step here, of which we have not given and are not going to be giving the proof, is the claim that a theory like **P** is strong enough to 'formalize' the proof of a result like Theorem 17.9. Gödel's successors, beginning with Paul Bernays, have analyzed just what properties of Prv_T are actually essential to get the second incompleteness theorem, finding that one does not really have to 'formalize' the whole proof of Theorem 17.9, but only certain key facts that serve as lemmas in that proof. We summarize the results of the analysis in the next two propositions.

18.2 Lemma*. Let T be a consistent, axiomatizable extension of **P**, and let $B(x)$ be the formula $\mathrm{Prv}_T(x)$. Then the following hold for all sentences:

(P1) If $\vdash_T A$ then $\vdash_T B(\ulcorner A \urcorner)$

(P2) $\vdash_T B(\ulcorner A_1 \to A_2 \urcorner) \to (B(\ulcorner A_1 \urcorner) \to B(\ulcorner A_2 \urcorner))$

(P3) $\vdash_T B(\ulcorner A \urcorner) \to B(\ulcorner B(\ulcorner A \urcorner) \urcorner)$.

Again we have starred the lemma because we are not going to give a full proof. First we note a property not on the above list:

(P0) If $\vdash_T A_1 \to A_2$ and $\vdash_T A_1$, then $\vdash_T A_2$.

This is a consequence of the Gödel *completeness* theorem, according to which the theorems of T are just the sentences implied by T, since if a conditional $A_1 \to A_2$ and its antecedent A_1 are both implied by a set of sentences, then so is its consequent A_2. Whatever notion of proof one starts with, so long as it is sound and complete, (P0) will hold. One might therefore just as well build it into one's notion of proof, adding some appropriate version of it to the rules of one's proof procedure. Of course, once it is thus built in, the proof of (P0) no longer requires the completeness theorem, but becomes comparatively easy. [For the particular proof procedure we used in Chapter 14, we discussed the possibility of doing this in section 14.3, where the version of (P0) appropriate to our particular proof procedure was called rule (R10).]

(P1) holds for any extension of **Q**, since if $\vdash_T A$, then $\mathrm{Prv}_T(\ulcorner A \urcorner)$ is correct, and being an ∃-rudimentary sentence, it is therefore provable in **Q**. (P2) is essentially the assertion that the proof of (P0) (which we have just said can be made comparatively easy) can be 'formalized' in **P**. (P3) is essentially the assertion that the (by no means so easy) proof of (P1) can also be 'formalized' in **P**. The proofs of the assertions (P2) and (P3) of 'formalizability' are omitted from virtually all books on the level of this one, not because they involve any terribly difficult new ideas, but because the innumerable routine verifications they—and especially the latter of them—require would take up too much time and patience. What we can and do include is the proof *that the starred lemma implies the starred theorem.* More generally, we have the following:

18.3 Theorem (Gödel's second incompleteness theorem, abstract form). Let T be a consistent, axiomatizable extension of **P**, and let $B(x)$ be a formula having properties (P1)–(P3) above. Then not $\vdash_T \sim B(\ulcorner \mathbf{0} = \mathbf{1} \urcorner)$.

The proof will occupy the remainder of this chapter. Throughout, let T be an extension (not necessarily consistent) of **Q**. A formula $B(x)$ with properties (P1)–(P3) of Lemma 18.2 we call a *provability predicate* for T. We begin with a few words about this notion. The formula $\mathrm{Prv}_T(x)$ considered so far we call the *traditional* provability predicate for T, though, as we have indicated, we are not going to give the proof of Lemma 18.2, and so are not going to be giving the proof that the 'traditional provability predicate' *is* a 'provability predicate' in the sense of our official definition of the latter term.

If T is ω-consistent, taking the traditional $\mathrm{Prv}_T(x)$ for $B(x)$, we have also the following property, the converse of (P1):

(P4) If $\vdash_T B(\ulcorner A \urcorner)$ then $\vdash_T A$.

[For if we had $\vdash_T \mathrm{Prv}_T(\ulcorner A \urcorner)$, or in other words $\vdash_T \exists y\ \mathrm{Prf}_T(\ulcorner A \urcorner, y)$, but did not have $\vdash_T A$, then for each b, $\sim\mathrm{Prf}_T(\ulcorner A \urcorner, \mathbf{b})$ would be correct and hence provable in **Q** and hence in T, and we would have an ω-inconsistency in T.] We do not, however, include ω-consistency in our assumptions on T, or (P4) in our definition of the technical term 'provability predicate'. Without the assumption of (P4), which is not part of our official definition, a 'provability predicate' need not have much to do with provability. In fact, the formula $x = x$ is easily seen to be a 'provability predicate' in the sense of our definition.

On the other hand, a formula may arithmetically define the set of Gödel numbers of theorems of T without being a provability predicate for T. If T is consistent and $\mathrm{Prv}_T(x)$ is the traditional provability predicate for T, then not only does $\mathrm{Prv}_T(x)$ arithmetically define the set of Gödel numbers of theorems of T, but so does the formula $\mathrm{Prv}^*{}_T(x)$, which is the conjunction of $\mathrm{Prv}_T(x)$ with $\sim\mathrm{Prv}_T(\ulcorner \mathbf{0} = \mathbf{1} \urcorner)$, since the second conjunct is true. But notice that, in contrast to Theorem 18.1, $\sim\mathrm{Prv}^*{}_T(\ulcorner \mathbf{0} = \mathbf{1} \urcorner)$ *is* provable in T. For it is simply

$$\sim(\mathrm{Prv}_T(\ulcorner \mathbf{0} = \mathbf{1} \urcorner) \,\&\, \sim\mathrm{Prv}_T(\ulcorner \mathbf{0} = \mathbf{1} \urcorner))$$

which is a *valid* sentence and hence a theorem of any theory. The formula $\mathrm{Prv}^*{}_T(x)$, however, lacks property (P1) in the definition of provability predicate. That is, it is *not* the case that if $\vdash_T A$ then $\vdash_T \mathrm{Prv}^*{}_T(\ulcorner A \urcorner)$. Indeed, it is *never* the case that $\vdash_T \mathrm{Prv}^*{}_T(\ulcorner A \urcorner)$, since it is not the case that $\vdash_T \sim\mathrm{Prv}_T(\ulcorner \mathbf{0} = \mathbf{1} \urcorner)$, by Theorem 18.1. The traditional provability predicate $\mathrm{Prv}_T(x)$ has the further important, if nonmathematical, property beyond (P0)–(P4), that intuitively speaking $\mathrm{Prv}(x)$ can plausibly be regarded as *meaning* or *saying* (on the standard interpretation) that x is the Gödel number of a sentence that is provable in T. This is conspicuously not the case for $\mathrm{Prv}^*{}_T(x)$, which means or says that x the Gödel number of a sentence that is provable in T *and T is consistent*.

The thought that whatever is provable had better be true might make it surprising that a further condition was not included in the definition of provability predicate, namely, that for every sentence A we have

(P5) $$\vdash_T B(\ulcorner A \urcorner) \to A.$$

But in fact, as we also show below, no provability predicate fulfills condition (P5) unless T is inconsistent.

Our next theorem will provide answers to three questions. First, just as the diagonal lemma provides a sentence, the Gödel sentence, that 'says of itself' that it is unprovable, so also it provides a sentence, the *Henkin sentence*, that 'says of itself' that it is provable. In other words, given a provability predicate $B(x)$, there is a sentence H_T such that $\vdash_T H_T \leftrightarrow B(\ulcorner H_T \urcorner)$. Gödel's theorem was that, if T is consistent, then the Gödel sentence is indeed unprovable. *Henkin's question* was whether the Henkin sentence is indeed provable. This is the first question our next theorem will answer. Second, call a formula Tr(x) a *truth predicate* for T if and only if for every sentence A of the language of T we have $\vdash_T A \leftrightarrow \text{Tr}(\ulcorner A \urcorner)$. Another question is whether, if T is consistent, there can exist a truth predicate for T. (The answer to this question is going to be negative. Indeed, the negative answer can actually be obtained directly from the diagonal lemma of the preceding chapter.) Third, if $B(x)$ is a provability predicate, call $\sim B(\ulcorner \mathbf{0} = \mathbf{1} \urcorner)$ the *consistency sentence* for T [relative to $B(x)$]. Yet another question is whether, if T is consistent, the consistency sentence for T can be provable in T. (We have already indicated in Theorem 18.3 that the answer to this last question is going to be negative.)

The proof of the next theorem, though elementary, is somewhat convoluted, and as warm-up we invite the reader to ponder the following paradoxical argument, by which we seem to be able to prove from pure logic, with no special assumptions, the existence of Santa Claus. (The argument would work equally well for Zeus.) Consider the sentence 'if this sentence is true, then Santa Claus exists'; or to put the matter another way, let S be the sentence 'if S is true, then Santa Claus exists'.

Assuming

(1) S is true

by the logic of identity it follows that

(2) 'If S is true, then Santa Claus exists' is true.

From (2) we obtain

(3) If S is true, then Santa Claus exists.

From (1) and (3) we obtain

(4) Santa Claus exists.

Having derived (4) from the assumption (1) we infer that *without* the assumption (1), indeed without any special assumption, that we at least have the conditional conclusion that if (1), then (4), or in other words

(5) If S is true, then Santa Claus exists.

From (5) we obtain

(6) ‘If S is true, then Santa Claus exists’ is true.

By the logic of identity again it follows that

(7) S is true.

And from (5) and (7) we infer, without any special assumptions, the conclusion that

(8) Santa Claus exists.

18.4 Theorem (Löb’s theorem). If $B(x)$ is a provability predicate for T, then for any sentence A, if $\vdash_T B(\ulcorner A\urcorner) \to A$, then $\vdash_T A$.

Proof: Suppose that B is a provability predicate for T and that

(1) $\vdash_T B(\ulcorner A\urcorner) \to A.$

Let $D(y)$ be the formula $(B(y) \to A)$, and apply the diagonal lemma to obtain a sentence C such that

(2) $\vdash_T C \leftrightarrow (B(\ulcorner C\urcorner) \to A).$

So

(3) $\vdash_T C \to (B(\ulcorner C\urcorner) \to A).$

By virtue of property (P1) of a provability predicate,

(4) $\vdash_T B(\ulcorner C \to (B(\ulcorner C\urcorner) \to A)\urcorner).$

By virture of (P2),

(5) $\vdash_T B(\ulcorner C \to (B(\ulcorner C\urcorner) \to A)\urcorner) \to (B(\ulcorner C\urcorner) \to B(\ulcorner B(\ulcorner C\urcorner) \to A\urcorner)).$

From (4) and (5) it follows that

(6) $\vdash_T B(\ulcorner C\urcorner) \to B(\ulcorner B(\ulcorner C\urcorner) \to A\urcorner).$

By virtue of (P2) again,

(7) $\vdash_T B(\ulcorner B(\ulcorner C\urcorner) \to A\urcorner) \to (B(\ulcorner B(\ulcorner C\urcorner)\urcorner) \to B(\ulcorner A\urcorner)).$

From (6) and (7) it follows that

(8) $\vdash_T B(\ulcorner C\urcorner) \to (B(\ulcorner B(\ulcorner C\urcorner)\urcorner) \to B(\ulcorner A\urcorner)).$

By virtue of (P3),

(9) $\vdash_T B(\ulcorner C\urcorner) \to B(\ulcorner B(\ulcorner C\urcorner)\urcorner).$

From (8) and (9) it follows that

(10) $\vdash_T B(\ulcorner C\urcorner) \to B(\ulcorner A\urcorner).$

From (1) and (10) it follows that

(11) $\vdash_T B(\ulcorner C\urcorner) \to A.$

From (2) and (11) it follows that

(12) $$\vdash_T C.$$

By virtue of (P1) again,

(13) $$\vdash_T B(\ulcorner C \urcorner).$$

And so finally, from (11) and (13), we have

(14) $$\vdash_T A.$$

Since the converse of Löb's theorem is trivial (if $\vdash_T A$, then $\vdash_T F \rightarrow A$ for any sentence F), a necessary and sufficient condition for A to be a theorem of T is that $B(\ulcorner A \urcorner) \rightarrow A$ is a theorem of T. Now for the promised derivation of the three results mentioned earlier.

18.5 Corollary. Suppose that $B(x)$ is a provability predicate for T. Then if $\vdash_T H \leftrightarrow B(\ulcorner H \urcorner)$, then $\vdash_T H$.

Proof: Immediate from Löb's theorem.

18.6 Corollary. If T is consistent, then T has no truth predicate.

Proof: Suppose that Tr(x) is a truth predicate for T. Then a moment's thought shows that Tr(x) is also a provability predicate for T. Moreover, since Tr(x) is a truth predicate, for every A we have $\vdash_T \mathrm{Tr}(\ulcorner A \urcorner) \rightarrow A$. But then by Löb's theorem, for every A we have $\vdash_T A$, and T is inconsistent.

And finally, here is the proof of Theorem 18.3.

Proof: Suppose $\vdash_T {\sim}B(\ulcorner \mathbf{0} = \mathbf{1} \urcorner)$. Then $\vdash_T B(\ulcorner \mathbf{0} = \mathbf{1} \urcorner) \rightarrow F$ for any sentence F, and in particular $\vdash_T B(\ulcorner \mathbf{0} = \mathbf{1} \urcorner) \rightarrow \mathbf{0} = \mathbf{1}$, and hence $\vdash_T \mathbf{0} = \mathbf{1}$, and since T is an extension of $\mathbf{Q}$, T is inconsistent.

It is characteristic of important theorems to raise new questions even as they answer old ones. Gödel's theorems (as well as some of the major recursion-theoretic and model-theoretic results we have passed on our way to Gödel's theorems) are a case in point. Several of the new directions of research they opened up will be explored in the remaining chapters of this book. One such question is that of how far one can go working just with the abstract properties (P1)–(P3), without getting involved in the messy details about a particular predicate $\mathrm{Prv}_T(x)$. That question will be explored in the last chapter of this book.

Historical Remarks

We alluded in passing in an earlier chapter to the existence of heterodox mathematicians who reject certain principles of logic. More specifically, in the late nineteenth and early twentieth centuries there were a number of mathematicians who rejected 'nonconstructive' as opposed to 'constructive' existence proofs and were led by this rejection to reject the method of proof by contradiction, which has been ubiquitously used in orthodox mathematics since Euclid (and has been repeatedly used in this book). The most extreme critics, the 'finitists', rejected the whole of established 'infinitistic' mathematics, declaring not only that the proofs of its theorems were fallacious, but that the very statements of those theorems were meaningless. Any mathematical assertion going beyond generalizations whose every instance can be checked by direct computation (essentially, anything beyond ∀-rudimentary sentences) was rejected.

In the 1920s, David Hilbert, the leading mathematician of the period, devised a program he hoped would provide a decisive answer to these critics. On the plane of philosophical principle, he in effect conceded that sentences going beyond ∀-rudimentary sentences are 'ideal' additions to 'contentful' mathematics. He compared this addition to the addition of 'imaginary' numbers to the system of real numbers, which had also raised doubts and objections when it was first introduced. On the plane of mathematical practice, Hilbert insisted, a detour through the 'ideal' is often the shortest route to a 'contentful' result. (For example, Chebyshev's theorem that there is a prime between any number and its double was proved not in some 'finitistic', 'constructive', directly computational way, but by an argument involving applying calculus to functions whose arguments and values are imaginary numbers.) Needless to say, this reply wouldn't satisfy a critic who doubted the *correctness* of 'contentful' results arrived at by such a detour. But Hilbert's program was precisely to *prove* that any 'contentful' result provable by orthodox, infinitistic mathematics is indeed correct. Needless to say, such a proof wouldn't satisfy a critic if the proof *itself* used the methods whose legitimacy was under debate. But more precisely Hilbert's program was to prove *by 'finitistic' means* that every ∀-rudimentary sentence proved by 'infinitistic' means is correct.

An important reduction of the problem was achieved. Suppose a mathematical theory T proves some *incorrect* ∀-rudimentary sentence $\forall x F(x)$. If this sentence is incorrect, then some specific *numerical instance* $F(\mathbf{n})$ for some specific number n must be incorrect. Of course, if the theory proves $\forall x F(x)$ it also proves each instance $F(\mathbf{n})$, since the instances follow from the generalization by pure logic. But if $F(\mathbf{n})$ is incorrect, then $\sim F(\mathbf{n})$ is a correct rudimentary sentence, and as such will be provable in T, for any 'sufficiently strong' T. Hence if such a T proves an ∀-rudimentary sentence $\forall x F(x)$, it will prove an outright contradiction, proving both $F(\mathbf{n})$ and $\sim F(\mathbf{n})$. So the problem of proving T yields only correct ∀-rudimentary theorems reduces to the problem of showing T is *consistent*. Hilbert's program was, then, *to prove finitistically the consistency of infinitistic mathematics*.

It can now be appreciated how Gödel's theorems derailed this program in its original form just described. While it was never made completely explicit what 'finitistic' mathematics does and does not allow, its assumptions amounted to *less than* the assumptions of inductive or Peano arithmetic **P**. On the other hand, the assumptions of 'infinitistic' mathematics amount to *more than* the assumptions of **P**. So what Hilbert was trying to do was prove, using a theory *weaker than* **P**, the consistency of a theory *stronger than* **P**, whereas what Gödel proved was that, even using the full strength of **P**, one cannot prove the consistency of **P** itself, let alone anything stronger.

In the course of this essentially philosophically motivated work, Gödel introduced the notion of primitive recursive function, and established the arithmetization of syntax by primitive recursive functions and the representability in formal arithmetic of primitive recursive functions. But though primitive recursive functions were thus originally introduced merely as a tool for the proof of the incompleteness theorems, it was not long before logicians, Gödel himself included, began to wonder how far beyond the class of primitive recursive functions one had to go before one arrived at a class of functions that could plausibly be supposed to *include all effectively computable functions*. Alonzo Church was the first to publish a definite proposal. A. M. Turing's proposal, involving his idealized machines, followed shortly thereafter, and with it the proof of the existence of a universal machine, another intellectual landmark of the last century almost on the level of the incompleteness theorems themselves.

Gödel and others went on to show that various other mathematically interesting statements, besides the consistency statement, are undecidable by **P**, assuming it to be consistent, and even by stronger theories, such as are introduced in works on set theory. In particular, Gödel and Paul Cohen showed that the accepted formal set theory of their day and ours could not decide an old conjecture of Georg Cantor, the creator of the theory of enumerable and nonenumberable sets, which Hilbert in 1900 had placed first on a list of problems for the coming century. The conjecture, called the *continuum hypothesis*, was that any nonenumerable set of real numbers is equinumerous with the *whole* set of real numbers. Mathematicians would be, according to the results of Gödel and Cohen, wasting their time attempting to settle this conjecture on the basis of currently accepted set-theoretic axioms, in the same way people who try to trisect the angle or square the circle are wasting their time. They must *either* find some way to justify adopting new set-theoretic axioms, *or else* give up on the problem. (Which they should do is a philosophical question, and like other philosophical questions, it has been very differently answered by different thinkers. Gödel and Cohen, in particular, arrayed themselves on opposite sides of the question: Gödel favored the search for new axioms, while Cohen was for giving up.)

19

Normal Forms

A normal form theorem *of the most basic type tells us that for every formula A there is a formula A* of some special syntactic form such that A and A* are logically equivalent. A* normal form theorem for satisfiability *tells us that for every set* Γ *of sentences there is a set* Γ* *of sentences of some special syntactic form such that* Γ *and* Γ* *are* equivalent for satisfiability*, meaning that one will be satisfiable if and only if the other is. In section 19.1 we establish the* prenex *normal form theorem, according to which every formula is logically equivalent to one with all quantifiers at the beginning, along with some related results. In section 19.2 we establish the* Skolem *normal form theorem, according to which every set of sentences is equivalent for satisfiability to a set of sentences with all quantifiers at the beginning and all quantifiers universal. We then use this result to give an alternative proof of the Löwenheim–Skolem theorem, which we follow with some remarks on implications of the theorem that have sometimes been thought 'paradoxical'. In the optional section 19.3 we go on to sketch alternative proofs of the compactness and Gödel completeness theorems, using the Skolem normal form theorem and an auxiliary result known as* Herbrand's theorem*. In section 19.4 we establish that every set of sentences is equivalent for satisfiability to a set of sentences not containing identity, constants, or function symbols. Section 19.1 presupposes only Chapters 9 and 10, while the rest of the chapter presupposes also Chapter 12. Section 19.2 (with its pendant 19.3) on the one hand, and section 19.4 on the other hand, are independent of each other. The results of section 19.4 will be used in the next two chapters.*

19.1 Disjunctive and Prenex Normal Forms

This chapter picks up where the problems at the end of Chapter 10 left off. There we asked the reader to show that that every formula is logically equivalent to a formula having no subformulas in which the same variable occurs both free and bound. This result is a simple example of a *normal form* theorem, a result asserting that every sentence is logically equivalent to one fulfilling some special syntactic requirement. Our first result here is an almost equally simple example. We say a formula is *negation-normal* if it is built up from atomic and negated atomic formulas using $\vee$, &, $\exists$, and $\forall$ alone, without further use of $\sim$.

19.1 Proposition (Negation-normal form). Every formula is logically equivalent to one that is negation-normal.

Proof: The proof is by induction on complexity. The base step is trivial, since an atomic formula is already negation-normal. Most cases of the induction step are trivial as well. For instance, if A and B are equivalent respectively to negation-normal formulas A^* and B^*, then $A \,\&\, B$ and $A \vee B$ are equivalent respectively to $A^* \,\&\, B^*$ and $A^* \vee B^*$, which are also negation-normal. The nontrivial case is to prove that if A is equivalent to the negation-normal A^* then $\sim A$ is equivalent to some negation-normal $A^\dagger$. This divides into six subcases according to the form of A^*. The case where A^* is atomic is trivial, since we may simply let $A^\dagger$ be $\sim A^*$. In case A^* is of form $\sim B$, so that $\sim A^*$ is $\sim\sim B$, we may let $A^\dagger$ be B. In case A^* is of form $(B \vee C)$, so that $\sim A^*$ is $\sim(B \vee C)$, which is logically equivalent to $(\sim B \,\&\, \sim C)$, by the induction hypothesis the simpler formulas $\sim B$ and $\sim C$ are equivalent to formulas $B^\dagger$ and $C^\dagger$ of the required form, so we may let $A^\dagger$ be $(B^\dagger \,\&\, C^\dagger)$. The case of conjunction is similar. In case A^* is of form $\exists x B$, so that $\sim A^*$ is $\sim\exists x B$, which is logically equivalent to $\forall x \sim B$, by the induction hypothesis the simpler formula $\sim B$ is equivalent to a formula $B^\dagger$ of the required form, so we may let $A^\dagger$ be $\forall x B^\dagger$. The case of universal quantification is similar.

In the foregoing proof we have used such equivalences as that of $\sim(B \vee C)$ to $\sim B \,\&\, \sim C$, to show 'from the bottom up' that there exists a negation-normal equivalent for any formula. What we show at the induction step is that if there exist negation-normal equivalents for the simpler formulas $\sim B$ and $\sim C$, then there exists a negation-normal equivalent for the more complex formula $\sim(B \vee C)$. If we actually want to *find* a negation-normal equivalent for a given formula, we use the same equivalences, but work 'from the top down'. We *reduce* the problem of finding a negation-normal equivalent for the more complex formula to that of finding such equivalents for simpler formulas. Thus, for instance, if P, Q, and R are atomic, then

$$\sim(P \vee (\sim Q \,\&\, R))$$

can be successively converted to

$$\begin{gathered}\sim P \,\&\, \sim(\sim Q \,\&\, R)\\ \sim P \,\&\, (\sim\sim Q \vee \sim R)\\ \sim P \,\&\, (Q \vee \sim R)\end{gathered}$$

the last of which is negation-normal. In this process use such equivalences as that of $\sim(B \vee C)$ to $\sim B \,\&\, \sim C$ to 'bring junctions out' or 'push negations in' until we get a formula equivalent to the original in which negation is applies only to atomic subformulas.

The above result on negation-normal form can be elaborated in two different directions. Let $A_1, A_2, \ldots, A_n$ be any formulas. A formula built up from them using only $\sim$, $\vee$, and &, without quantifiers, is said to be a *truth-functional compound* of the given formulas. A truth-functional compound is said to be in *disjunctive normal form* if it is a disjunction of conjunctions of formulas from among the A_i and their negations. (A notion of *conjunctive normal form* can be defined exactly analogously.)

19.2 Proposition (Disjunctive normal form). Every formula is logically equivalent to one that is in disjunctive normal form.

Proof: Given any formula, first replace it by a negation-normal equivalent. Then, using the *distributive laws*, that is, the equivalence of $(B \,\&\, (C \vee D))$ to $((B \,\&\, C) \vee (B \,\&\, D))$ and of $((B \vee C) \,\&\, D)$ to $((B \vee D) \,\&\, (C \vee D))$, 'push conjunction inside' and 'pull disjunction outside' until a disjunctive normal equivalent is obtained. (It would be a tedious but routine task to rewrite this 'top down' description of the process of finding a disjunctive normal equivalent as a 'bottom up' proof the existence of such an equivalent.)

If in a formula that is in disjunctive normal form each disjunction contains each A_i exactly once, plain or negated, then the compound is said to be in *full* disjunctive normal form. (A notion of *full* conjunctive normal form can be defined exactly analogously.) In connection with such forms it is often useful to introduce, in addition to the two-place connectives $\vee$ and $\&$, and the one-place connective $\sim$, the *zero-place connectives* or *constant truth* $\top$ and *constant falsehood* $\bot$, counting respectively as true in every interpretation and false in every interpretation. The disjunction of zero disjuncts may by convention be understood to be $\bot$, and the conjunction of zero conjuncts to be $\top$ (rather as, in mathematics, the sum of zero summands is understood to be 0, and the product of zero factors to be 1).

In seeking a full disjunctive normal equivalent of a given disjunctive normal formula, first note that conjunctions (and analogously, disjunctions) can be reordered and regrouped at will using the *commutative* and *associative* laws, that is, the equivalence of $(B \,\&\, C)$ to $(C \,\&\, B)$, and of $(B \,\&\, C \,\&\, D)$, which officially is supposed to be an abbreviation of $(B \,\&\, (C \,\&\, D))$, with grouping to the right, to $((B \,\&\, C) \,\&\, D)$, with grouping to the left. Thus for instance $(P \,\&\, (Q \,\&\, P))$ is equivalent to $(P \,\&\, (P \,\&\, Q))$ and to $((P \,\&\, P) \,\&\, Q)$. Using the *idempotent* law, that is, the equivalence of $B \,\&\, B$ to B, this last is equivalent to $P \,\&\, Q$. This illustrates how *repetitions* of the same A_i (or $\sim A_i$) within a conjunction can be eliminated. To eliminate the occurrence of the same A_i twice, once plain and once negated, we can use the equivalence of $B \,\&\, \sim B$ to $\bot$ and of $\bot \,\&\, C$ to $\bot$, and of $\bot \vee D$ to D, so that, for instance, $(B \,\&\, \sim B \,\&\, C) \vee D$ is equivalent simply to D: *contradictory disjuncts can be dropped*. These reductions will convert a given formula to one that, like our earlier example $(\sim P \,\&\, Q) \vee (\sim P \,\&\, R)$, is a disjunction of conjunctions in which each basic formula occurs *at most* once, plain or negated, in each conjunct.

To ensure that each occurs *at least* once in each conjunction, we use the equivalence of B to $(B \,\&\, C) \vee (B \,\&\, \sim C)$. Thus our example is equivalent to

$$(\sim P \,\&\, Q \,\&\, R) \vee (\sim P \,\&\, Q \,\&\, \sim R) \vee (\sim P \,\&\, \sim R)$$

and to

$$(\sim P \,\&\, Q \,\&\, R) \vee (\sim P \,\&\, Q \,\&\, \sim R) \vee (\sim P \& Q \,\&\, \sim R) \vee (\sim P \,\&\, \sim Q \,\&\, \sim R)$$

and, eliminating repetition, to

$$(\sim P \,\&\, Q \,\&\, R) \vee (\sim P \,\&\, Q \,\&\, \sim R) \vee (\sim P \,\&\, \sim Q \,\&\, \sim R)$$

which is in full disjunctive normal form. The foregoing informal description can be converted into a formal proof of the following result.

19.3 Theorem (Full disjunctive normal form). Every truth-functional compound of given formulas is logically equivalent to one in full disjunctive normal form.

The theorem on negation-normal forms can be elaborated in another direction. A formula A is said to be in *prenex* form if it is of the form

$$\mathsf{Q}_1 x_1 \mathsf{Q}_2 x_2 \ldots \mathsf{Q}_n x_n B$$

where each Q is either $\exists$ or $\forall$, and where B contains no quantifiers. The sequence of quantifiers and variables at the beginning is called the *prefix*, and the quantifier-free formula that follows the *matrix*.

19.4 Example (Finding a prenex equivalent for a given formula). Consider ($\forall x\, Fx \leftrightarrow Ga$), where F and G are one-place predicates. This is officially an abbreviation for

$$(\sim\forall x Fx \vee Ga) \,\&\, (\sim Ga \vee \forall x Fx).$$

Let us first put this in negation-normal form

$$(\exists x{\sim}Fx \vee Ga) \,\&\, (\sim Ga \vee \forall x Fx).$$

The problem now is to 'push junctions in'. This may be done by noting that the displayed negation-normal form is equivalent successively to

$$\exists x(\sim Fx \vee Ga) \,\&\, (\sim Ga \vee \forall x Fx)$$
$$\exists x(\sim Fx \vee Ga) \,\&\, \forall x(\sim Ga \vee Fx)$$
$$\exists y(\sim Fy \vee Ga) \,\&\, \forall x(\sim Ga \vee Fx)$$
$$\forall x(\exists y(\sim Fy \vee Ga) \,\&\, (\sim Ga \vee Fx))$$
$$\forall x \exists y((\sim Fy \vee Ga) \,\&\, (\sim Ga \vee Fx)).$$

If we had 'pulled quantifiers out' in a different order, a different prenex equivalent would have been obtained.

19.5 Theorem (Prenex normal form). Every formula is logically equivalent to one in prenex normal form.

Proof: By induction on complexity. Atomic formulas are trivially prenex. The result of applying a quantifier to a prenex formula is prenex (and hence the result of applying a quantifier to a formula equivalent to a prenex formula is equivalent to a prenex formula). The equivalence of the negation of a prenex formula (or a formula equivalent to one) to a prenex formula follows by repeated application of the equivalence of $\sim\forall x$ and $\sim\exists x$ to $\exists x{\sim}$ and $\forall x{\sim}$, respectively. The equivalence of a conjunction (or disjunction) of prenex formulas to a prenex formula follows on first relettering bound variables as in Problem 10.13, so the conjuncts or disjuncts have no variables in common, and then repeatedly applying the equivalence of $\mathsf{Q}xA(x) \,\S\, B$, where x does not occur in B, to $\mathsf{Q}x(A(x) \,\S\, B)$, where Q may be $\forall$ or $\exists$ and $\S$ may be $\&$ or $\vee$.

In the remainder of this chapter our concern is less with finding a logical equivalent of a special kind for a given sentence or formula than with finding equivalents for satisfiability of a special kind for a given sentence or set of sentences. Two sets of sentences Γ and Γ^* are *equivalent for satisfiability* if and only if they are either both satisfiable or both unsatisfiable, though generally when we prove the existence of such equivalents our proof will actually provide some additional information, indicating a stronger relationship between the two sets. Two different results on the existence of equivalents for satisfiability will be established in sections 19.2 and 19.4. In each case, Γ will be shown to have an equivalent for satisfiability Γ^* whose sentences will be of a simpler type syntactically, but which will involve new nonlogical symbols.

In this connection some terminology will be useful. Let L be any language, and L^+ any language containing it. Let $\mathcal{M}$ be an interpretation of L, and $\mathcal{M}^+$ an interpretation of L^+. If the interpretations have the same domain and assign the same denotations to nonlogical symbols in L (so that the only difference is that the one assigns denotations to symbols of L^+ not in L, while the other does not), then $\mathcal{M}^+$ is said to be an *expansion* of $\mathcal{M}$ to L^+, and $\mathcal{M}$ to be the *reduct* of $\mathcal{M}^+$ to L. Note that the notions of expansion and reduct pertain to changing the language while keeping the domain fixed.

19.2 Skolem Normal Form

A formula in prenex form with all quantifies universal (respectively, existential) may be called a *universal* or $\forall$-formula (respectively, an *existential* or $\exists$-formula). Consider a language L and a sentence of that language in prenex form, say

$$(1) \qquad \forall x_1 \exists y_1 \forall x_2 \exists y_2 R(x_1, y_1, x_2, y_2).$$

Now for each existential quantifier, let us introduce a new function symbol with as many places as there are universal quantifiers to its left, to obtain an expanded language L^+. Thus in our example there would be two new function symbols, say f_1 and f_2, corresponding to $\exists y_1$ and $\exists y_2$, the former having one place corresponding to $\forall x_1$, and the latter two places corresponding to $\forall x_1$ and $\forall x_2$. Let us replace each existentially quantified variable by the term that results on applying the corresponding function symbol to the universally quantified variable(s) to its left. The resulting $\forall$-formula, which in our example would be

$$(2) \qquad \forall x_1 \forall x_2 R(x_1, f_1(x_1), x_2, f_2(x_1, x_2))$$

is called the *Skolem normal form* of the original sentence, and the new function symbols occurring in it the *Skolem function symbols*.

A little thought shows that (2) logically implies (1). In any interpretation of the expanded language L^+ with the new function symbols, it is the case that for every element a_1 of the domain there is an element b_1, such that for every element a_2 there is an element b_2, such that a_1, b_1, a_2, b_2 satisfy $R(x_1, y_1, x_2, y_2)$: namely, take for b_1 the result of applying to a_1 the function denoted by f_1, and for b_2 the result of applying to a_1 and a_2 the function denoted by f_2.

We cannot, of course, say that conversely (1) implies (2). What is true is that (2) is implied by (1) together with the following:

$$(3.1) \qquad \forall x_1(\exists y_1 \forall x_2 \exists y_2 R(x_1, y_1, x_2, y_2) \rightarrow \forall x_2 \exists y_2 R(x_1, f_1(x_1), x_2, y_2))$$

$$(3.2) \qquad \forall x_1 \forall x_2(\exists y_2 R(x_1, f_1(x_1), x_2, y_2) \rightarrow R(x_1, f_1(x_1), x_2, f_2(x_1, x_2))).$$

For (1) and (3.1) imply

$$\forall x_1 \forall x_2 \exists y_2 R(x_1, f_1(x_1), x_2, y_2)$$

which with (3.2) implies (2). The sentences (3) are called the *Skolem axioms*.

For a prenex formula of a different kind, with different numbers of universal and existential quantifiers, the number and number of places of the required Skolem functions would be different, and the Skolem axioms correspondingly so. For instance, for

$$(1') \qquad \exists y_0 \forall x_1 \forall x_2 \exists y_1 \exists y_2 \forall x_3 Q(y_0, x_1, x_2, y_1, y_2, x_3)$$

we would need one zero-place function symbol (which is to say, one constant) f_0 and two two-place function symbols f_1 and f_2. The Skolem normal form would be

$$(2') \qquad \forall x_1 \forall x_2 \forall x_3 Q(f_0, x_1, x_2, f_1(x_1, x_2), f_2(x_1, x_2), x_3)$$

and the Skolem axioms would be

$$\begin{aligned}
(3.0') \qquad & \exists y_0 \forall x_1 \forall x_2 \exists y_1 \exists y_2 \forall x_3 Q(y_0, x_1, x_2, y_1, y_2, x_3) \rightarrow \\
& \qquad \forall x_1 \forall x_2 \exists y_1 \exists y_2 \forall x_3 Q(f_0, x_1, x_2, y_1, y_2, x_3) \\
(3.1') \qquad & \forall x_1 \forall x_2 \ \ (\exists y_1 \exists y_2 \forall x_3 Q(f_0, x_1, x_2, y_1, y_2, x_3) \rightarrow \\
& \qquad \exists y_2 \forall x_3 Q(f_0, x_1, x_2, f_1(x_1, x_2), y_2, x_3)) \\
(3.2') \qquad & \forall x_1 \forall x_2 \ \ (\exists y_2 \forall x_3 Q(f_0, x_1, x_2, f_1(x_1, x_2), y_2, x_3) \rightarrow \\
& \qquad \forall x_3 Q(f_0, x_1, x_2, f_1(x_1, x_2), f_2(x_1, x_2), x_3)).
\end{aligned}$$

But in exactly the same way in any example, the Skolem normal form will imply the original formula, and the original formula together with the Skolem axioms will imply the Skolem normal form.

If L is a language and L^+ is the result of adding Skolem functions for some or all of its sentences, then an expansion $\mathcal{M}^+$ of an interpretation $\mathcal{M}$ of L to an interpretation of L^+ is called a *Skolem* expansion if it is a model of the Skolem axioms.

19.6 Lemma (Skolemization). Every interpretation of L has a Skolem expansion.

Proof: The essential idea of the proof is sufficiently illustrated by the case of our original example (1) above. The proof uses a set-theoretic principle known as the *axiom of choice*. According to this principle, given any family of nonempty sets, there is a function ε whose domain is that family of sets, and whose value $\varepsilon(X)$ for any set Y in the family is some element of Y. Thus ε 'chooses' an element out of each Y in the family. We apply this assumption to the family of nonempty subsets of $|\mathcal{M}|$ and use ε to define a Skolem expansion $\mathcal{N} = \mathcal{M}^+$ of $\mathcal{M}$.

We first want to assign a denotation $f_1^{\mathcal{N}}$ that will make the Skolem axiom (3.1) come out true. To this end, for any element a_1 in $|\mathcal{M}|$ consider the set B_1 of all $b_1$in $|\mathcal{M}|$ such that a_1 and b_1 satisfy $\forall x_2 \exists y_2 R(x_1, y_1, x_2, y_2)$ in $\mathcal{M}$. If B_1 is empty, then no matter what we take $f_1^{\mathcal{N}}(a_1)$ to be, a_1 will satisfy the conditional

$$\exists y_1 \forall x_2 \exists y_2 \ R(x_1, y_1, x_2, y_2) \rightarrow \forall x_2 \exists y_2 R(x_1, f_1(x_1), x_2, y_2)$$

since it will not satisfy the antecedent. But for definiteness, let us say that if B_1 is empty, then we are to take $f_1^{\mathcal{N}}(a_1)$ to be $\varepsilon(|\mathcal{M}|)$. If B_1 is nonempty, then we take $f_1^{\mathcal{N}}(a_1)$ to be $\varepsilon(B_1)$: we use ε to choose one particular element b_1 such that a_1 and b_1 satisfy $\forall x_2 \exists y_2 R(x_1, y_1, x_2, y_2)$. Then since a_1 and $f_1^{\mathcal{N}}(a_1)$ will satisfy $\forall x_2 \exists y_2 R(x_1, y_1, x_2, y_2)$, it follows that a_1 will satisfy the foregoing conditional, and since this will be the case for any a_1, it follows that (3.1) will be true.

We next want to assign a $f_2^{\mathcal{N}}$ that will make the Skolem axiom (3.2) come out true. We proceed in exactly the same way. For any a_1 and a_2, consider the set B_2 of all b_2 such that a_1, a_2, and b_2 satisfy $R(x_1, f_1(x_1), x_2, y_2)$. If B_1 is empty, we take $f_2^{\mathcal{N}}(a_1, a_2)$ to be $\varepsilon(|\mathcal{M}|)$, and otherwise take it to be $\varepsilon(B_2)$. The procedure would be the same no matter how many Skolem function symbols we needed to assign denotations to, and how many Skolem axioms we needed to make true.

Let Γ be any set of sentences of any language L, and for each sentence A in Γ, first associate to it a logically equivalent prenex sentence A^* as in the preceding section, and then associate to A^* its Skolem form $A^\#$ as above, and let $\Gamma^\#$ be the set of all these sentences $A^\#$ for A in Γ. Then $\Gamma^\#$ is a set of $\forall$-sentences equivalent for satisfiability to the original set Γ. For if $\Gamma^\#$ is satisfiable, there is an interpretation $\mathcal{N}$ in which each $A^\#$ in $\Gamma^\#$ comes out true, and since $A^\#$ implies A^* and A^* is equivalent to A, we thus have an interpretation in which each A in Γ comes out true, so Γ is satisfiable. Conversely, if Γ is satisfiable, there is an interpretation $\mathcal{M}$ of the original language in which each A in Γ and hence each A^* comes out true. By the preceding lemma, $\mathcal{M}$ has an expansion $\mathcal{N}$ to an interpretation in which each A^* remains true and all Skolem axioms come out true. Since A^* together with the Skolem axioms implies $A^\#$, each $A^\#$ in $\Gamma^\#$ comes out true in $\mathcal{N}$, and $\Gamma^\#$ is satisfiable. We have thus shown how we can associate to any set of sentences a set of $\forall$-sentences equivalent to it for satisfiability. This fact, however, does not exhaust the content of the Skolemization lemma. For it can also be used to give a proof of the Löwenheim–Skolem theorem, and in a stronger version than that stated in chapter 12 (and proved in chapter 13).

To state the strong Löwenheim–Skolem theorem we need the notion of what it is for one interpretation $\mathcal{B}$ to be a *subinterpretation* of another interpretation $\mathcal{A}$. Where function symbols are absent, the definition is simply that (1) the domain $|\mathcal{B}|$ should be a subset of the domain $|\mathcal{A}|$, (2) for any $b_1, \ldots, b_n$ in $|\mathcal{B}|$ and any predicate R one has

$$\text{(S1)} \qquad R^{\mathcal{B}}(b_1, \ldots, b_n) \quad \text{if and only if} \quad R^{\mathcal{A}}((b_1), \ldots, (b_n))$$

and (3) for every constant c one has

$$\text{(S2)} \qquad (c^{\mathcal{B}}) = c^{\mathcal{A}}.$$

Thus, $\mathcal{B}$ is just like $\mathcal{A}$, except that we 'throw away' the elements of $|\mathcal{A}|$ that are not in $|\mathcal{B}|$.

Where function symbols are present, we have also to require that for any $b_1, \ldots,$ b_n in $|\mathcal{B}|$ and any function symbol f the following should hold:

$$\text{(S3)} \qquad f^{\mathcal{B}}(b_1, \ldots, b_n) = f^{\mathcal{A}}(b_1, \ldots, b_n).$$

Note that this last implies that $f^{\mathcal{A}}(b_1, \ldots, b_n)$ must be in $|\mathcal{B}|$: Where function symbols are absent, *any* nonempty subset B of $\mathcal{A}$ can be the domain of a subinterpretation of $\mathcal{A}$, but where function symbols are present, only those nonempty subsets B

can be the domains of subinterpretations that are *closed* under the functions $f^{\mathcal{A}}$ that are the denotations of function symbols of the language, or in other words, that contain the value of any of these functions for given arguments if they contain the arguments themselves.

When $\mathcal{B}$ is a subinterpretation of $\mathcal{A}$, we say that $\mathcal{A}$ is an *extension* of $\mathcal{B}$. Note that the notions of extension and subinterpretation pertain to enlarging or contracting the domain, while keeping the language fixed.

Note that it follows from (S1) and (S3) by induction on complexity of terms that every term has the same denotation in $\mathcal{B}$ as in $\mathcal{A}$. It then follows by (S1) and (S2) that any atomic sentence has the same truth value in $\mathcal{B}$ as in $\mathcal{A}$. It then follows by induction on complexity that every quantifier-free sentence has the same truth value in $\mathcal{B}$ as in $\mathcal{A}$. Essentially the same argument shows that, more generally, any given elements of $\mathcal{B}$ satisfy the same quantifier-free formulas in $\mathcal{B}$ as in $\mathcal{A}$. If an $\exists$-sentence $\exists x_1 \ldots \exists x_n R(x_1, \ldots, x_n)$ is true in $\mathcal{B}$, then there are elements $b_1, \ldots, b_n$ of $|\mathcal{B}|$ that satisfy the quantifier-free formula $R(x_1, \ldots, x_n)$ in $\mathcal{B}$ and hence, by what has just been said, in $\mathcal{A}$ as well, so that the $\exists$-sentence $\exists x_1 \ldots \exists x_n R(x_1, \ldots, x_n)$ is also true in $\mathcal{A}$. Using the logical equivalence of the negation of an $\forall$-sentence to an $\exists$-sentence, we have the following result.

19.7 Proposition. Let $\mathcal{A}$ be any interpretation and $\mathcal{B}$ any subinterpretation thereof. Then any $\forall$-sentence true in $\mathcal{A}$ is true in $\mathcal{B}$.

19.8 Example (Subinterpretations). Proposition 19.7 is in general as far as one can go. For consider the language with just the two-place predicate $<$. Let $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$ have domains the integers, rational numbers, and real numbers, respectively, and let the denotation of $<$ in each case be the usual order relation $<$ on the numbers in question. Since the order of integers *qua* integers is the same as their order *qua* rational numbers, and the order of rational numbers *qua* rational numbers is the same as their order *qua* real numbers, $\mathcal{P}$ is a subinterpretation of $\mathcal{Q}$ and $\mathcal{R}$, and $\mathcal{Q}$ is a subinterpretation of $\mathcal{R}$. Consider, however, the sentence

$$\forall x \forall y (x < y \rightarrow \exists z (x < z \,\&\, z < y))$$

or its prenex equivalent

$$\forall x \forall y \exists z (x < y \rightarrow (x < z \,\&\, z < y)).$$

$\mathcal{R}$ and $\mathcal{Q}$ are models of this sentence, since between any two real numbers a and b with $a < b$ there is some other real number c with $a < c$ and $c < b$, such as $(b - a)/2$, and similarly for rational numbers. But $\mathcal{P}$ is not a model of the sentence, since between the integers 0 and 1 there is no other integer. (Of course, the sentence here is not an $\forall$-sentence, but it is, so to speak, just one step beyond, an $\forall\exists$-sentence.)

Thus a subinterpretation of a model of a sentence C (or set of sentences Γ) may, but in general need not, also be a model of C (or Γ): if it is, it is called a sub*model*. Without further ado, here is the strong version of the Löwenheim–Skolem theorem. (The phrase 'enumerable' is redundant, given that we are restricting our attention to enumerable languages, but we include it to emphasize that we are making this restriction.)

19.9 Theorem (Strong Löwenheim–Skolem theorem). Let $\mathcal{A}$ be a nonenumerable model of an enumerable set of sentences Γ. Then $\mathcal{A}$ has an enumerable subinterpretation that is also a model of Γ.

Proof: It will suffice to prove the theorem for the special case of sets of $\forall$-sentences. For suppose we have proved the theorem in this special case, and consider the general case where $\mathcal{A}$ is a model of some arbitrary set of sentences Γ. Then as in our earlier discussion, $\mathcal{A}$ has an expansion $\mathcal{A}^{\#}$ to a model of $\Gamma^{\#}$, the set of Skolem forms of the sentences in Γ. Since Skolem forms are $\forall$-sentences, by the special case of the theorem there will an enumerable subinterpretation $\mathcal{B}^{\#}$ that is also a model of $\Gamma^{\#}$. Then since the Skolem form of a sentence implies the original sentence, $\mathcal{B}^{\#}$ will also be a model of Γ, and so will be its reduct $\mathcal{B}$ to the original language. But this $\mathcal{B}$ will be an enumerable subinterpretation of $\mathcal{A}$.

To prove the theorem in the special case where all sentences in Γ are $\forall$-sentences, consider the set B of all denotations in $\mathcal{A}$ of closed terms of the language of Γ. (We may assume there are some closed terms, since if not, we may add a constant c to the language and the logically valid sentence $c = c$ to Γ.) Since that language is enumerable, so is the set of closed terms, and so is B. Since B is closed under the functions that are denotations of the function symbols of the language, it is the domain of an enumerable subinterpretation $\mathcal{B}$ of $\mathcal{A}$. And by Proposition 19.7, every $\forall$-sentence true in $\mathcal{A}$ is true in $\mathcal{B}$, so $\mathcal{B}$ is a model of Γ.

Two interpretations $\mathcal{A}$ and $\mathcal{B}$ for the same language are called *elementarily equivalent* if every sentence true in the one is true in the other. Taking for Γ in the above version of the Löwenheim–Skolem theorem the set of *all* sentences true in $\mathcal{A}$, the theorem tells us that any interpretation has an enumerable subinterpretation that is elementarily equivalent to it. $\mathcal{A}$ subinterpretation $\mathcal{B}$ of an interpretation $\mathcal{A}$ is called an *elementary subinterpretation* if for any formula $F(x_1, \ldots, x_n)$ and any elements $b_1, \ldots, b_n$ of $|\mathcal{B}|$, the elements satisfy the formula in $\mathcal{A}$ if and only if they satisfy it in $\mathcal{B}$. This implies elementary equivalence, but is in general a stronger condition. By extending the notion of Skolem normal form to formulas with free variables, the above strong version of the Löwenheim–Skolem theorem can be sharpened to a still stronger one telling us that any interpretation has an enumerable elementary subinterpretation.

Applications of Skolem normal form will be given in the next section. Since we are not going to be needing the sharper result stated in the preceding paragraph for these applications, we do not go into the (tedious but routine) details of its proof. Instead, before turning to applications we wish to discuss another, more philosophical issue. At one time the Löwenheim–Skolem theorem (especially in the strong form in which we have proved it in this section) was considered philosophically perplexing because some of its consequences were perceived as anomalous. The apparent anomaly, sometime called ‘Skolem’s paradox’, is that there exist certain interpretations in which a certain sentence, which seems to say that nonenumerably many sets of natural numbers exist, is true, even though the domains of these interpretations contain only enumerably many sets of natural numbers, and the predicate in the sentence that we would be inclined to translate as 'set (of natural numbers)' is true just of the sets (of natural numbers) in the domains.

19.10 Example (The 'Skolem paradox'). There is no denying that the state of affairs thought to be paradoxical does obtain. In order to see how it arises, we first need an alternative account of what it is for a set E of sets of natural numbers to be enumerable, and for this we need to use the coding of an ordered pair (m, n) of natural numbers by a single number $J(m, n)$, as described in section 1.2. We call a set w of natural numbers an *enumerator* of a set E of sets of natural numbers if

$$\begin{aligned}&\forall z(z \text{ is a set of natural numbers} \,\&\, z \text{ is in } E \rightarrow\\ &\quad \exists x(x \text{ is a natural number} \,\&\\ &\qquad \forall z(\forall y(y \text{ is a natural number} \rightarrow (y \text{ is in } z \leftrightarrow J(x, y) \text{ is in } w)))).\end{aligned}$$

The fact about enumerators and enumerability we need is that *a set E of sets of natural numbers is enumerable if and only if E has an enumerator.*

[The reason: suppose E is enumerable. Let $e_0, e_1, e_2, \ldots$ be an enumeration of sets of natural numbers that contains all the members of E, and perhaps other sets of natural numbers also. Then the set of numbers $J(x, y)$ such that y is in e_x is an enumerator of E. Conversely, if w is an enumerator of E, then letting e_x be the set of those numbers y such that $J(x, y)$ is in w, we get an enumeration $e_0, e_1, e_2, \ldots$ that contains all members of E, and E is enumerable.]

We want now to look at a language and some of its interpretations. The language contains just the following: constants **0**, **1**, **2**, . . . , two one-place predicates **N** and **S**, a two-place predicate $\in$, and a two-place function symbol **J**. An interpretation $\mathcal{I}$ of the kind we are interested in will have as the elements of its domain all the natural numbers, some or all of the sets of natural numbers, and nothing else. The denotations of **0**, **1**, **2**, and so on will be the numbers 0, 1, 2, and so on. The denotation of **N** will be the set of all natural numbers, and of **S** will be the set of all sets of natural numbers in the domain; while the denotation of $\in$ will be the relation of membership between numbers and sets of numbers. Finally, the denotation of **J** will be the function J, extended to give some arbitrary value—say 17—for arguments that are not both numbers (that is, one or both of which are sets). Among such interpretations, the *standard* interpretation $\mathcal{J}$ will be the one in which the domain contains *all* sets of natural numbers.

Consider the sentence $\sim\exists w\, F(w)$ where $F(w)$ is the formula

$$\mathbf{S}w \,\&\, \forall z(\mathbf{S}z \rightarrow \exists x(\mathbf{N}x \,\&\, \forall y(\mathbf{N}y \rightarrow (y \in z \leftrightarrow \mathrm{J}(x, y) \in w)))).$$

In each of the interpretations I that concern us, $\sim\exists w\, F(w)$ will have a truth value. It will be *true* in I if and only if there is set *in the domain of I* that is an enumerator of the set of all sets of numbers that are *in the domain of I*, as can be seen by compairing the formula $F(w)$ with the definition of enumerator above. We can*not* say, more simply, that the sentence is true in the interpretation if and only if there is no enumerator of the set of all sets of numbers in its domain, because the quantifier $\exists w$ only 'ranges over' or 'refers to' sets *in the domain.*

There is, as we know, *no* enumerator of the set of *all* sets of numbers, so the sentence $\sim\exists w\, F(w)$ is true in the standard interpetation $\mathcal{J}$, and can be said to mean 'nonenumerably many sets of numbers exist' when interpreted 'over' $\mathcal{J}$, since it then denies that there is an enumerator of the set of all sets of numbers. By the Löwenheim–Skolem theorem, there is an enumerable subinterpretation $\mathcal{K}$ of $\mathcal{J}$ in which the sentence is also true. (Note that all numbers will be in its domain, since each is the denotation of some constant.) Thus there is an interpretation $\mathcal{K}$ whose domain contains only enumerably many sets of numbers, and in which **S** is true of just the sets of numbers in its domain. This is the 'Skolem paradox'.

How is the paradox to be resolved? Well, though the set of all sets of numbers in the domain of $\mathcal{K}$ does indeed have an enumerator, since the domain is enumerable, none of its enumerators can be *in* the domain of $\mathcal{K}$. [Otherwise, it would satisfy $F(x)$, and $\exists w\, F(x)$ would be true in $\mathcal{K}$, as it is not.] So part of the explanation of how the sentence $\sim\exists w\, F(x)$ can be true in $\mathcal{K}$ is that those sets that 'witness' that the set of sets of numbers in the domain of $\mathcal{K}$ is enumerable are not themselves members of the domain of $\mathcal{K}$.

A further part of the explanation is that what a sentence should be understood as saying or meaning or denying is at least as much as function of the domain over which the sentence is interpreted (and even of the way in which that interpretation is described or referred to) as of the symbols that constitute the sentence. $\sim\exists w\, F(x)$ can be understood as saying 'nonenumerably many sets of numbers exist' when its quantifiers are understood as ranging over a collection containing all numbers and all sets of numbers, as with the domain of the standard interpretation $\mathcal{J}$; but it cannot be so understood when its quantifiers range over other domains, and in particular not when they range over the members of enumerable domains. The sentence $\sim\exists w\, F(x)$—that sequence of symbols—'says' something only when supplied with an interpretation. It may be surprising and even amusing that it is true in all sorts of interpretations, including perhaps some subinterpretations $\mathcal{K}$ of $\mathcal{J}$ that have enumerable domains, but it should not *a priori* seem impossible for it to be true in these. Interpreted over such a $\mathcal{K}$, it will only say 'the domain of $\mathcal{K}$ contains no enumerator of the set of sets of numbers in $\mathcal{K}'$. And this, of course, is true.

19.3 Herbrand's Theorem

The applications of Skolem normal form with which we are going to be concerned in this section require some preliminary machinery, with which we begin. We work throughout in logic *without identity*. (Extensions of the results of this section to logic *with identity* are possible using the machinery to be developed in the next section, but we do not go into the matter.)

Let $A_1, \ldots, A_n$ be atomic sentences. A (*truth-functional*) *valuation* of them is simply a function ω assigning each of them one of the truth values, true or false (represented by, say, 1 and 0). The valuation can be extended to truth-functional compounds of the A_i (that is, quantifier-free sentences built up from the A_i using $\sim$ and & and $\vee$) in the same way that the notion of truth in an interpretation is extended from atomic to quantifier-free sentences:

$$\begin{aligned} \omega(\sim B) = 1 &\quad \text{if and only if} \quad \omega(B) = 0 \\ \omega(B \,\&\, C) = 1 &\quad \text{if and only if} \quad \omega(B) = 1 \text{ and } \omega(C) = 1 \\ \omega(B \vee C) = 1 &\quad \text{if and only if} \quad \omega(B) = 1 \text{ or } \omega(C) = 1. \end{aligned}$$

A set Γ of quantifier-free sentences built up from the A_i is said to be *truth-functionally satisfiable* if there is some valuation ω giving every sentence S in Γ the value 1.

Now if Γ is satisfiable in the ordinary sense, that is, if there is an interpretation $\mathcal{A}$ in which every sentence in Γ comes out true, then certainly Γ is truth-functionally satisfiable. Simply take for ω the function that gives a sentence the value 1 if and only if it is true in $\mathcal{A}$.

The converse is also true. In other words, if there is a valuation ω that gives every sentence in Γ the value 1, then there is an interpretation $\mathcal{A}$ in which every sentence in Γ comes out true. To show this, it is enough to show that for any valuation ω, there is an interpretation $\mathcal{A}$ such that each A_i comes out true in $\mathcal{A}$ just in case ω assigns it the value 1. This is in fact the case even if we start with an *infinite* set of atomic formulas A_i. To specify $\mathcal{A}$, we must specify a domain, and assign a denotation to each constant, function symbol, and predicate occurring in the A_i. Well, simply take for each closed term t in the language some object t^*, with distinct terms corresponding to distinct objects. We take the domain of our interpretation to consist of these objects t^*. We take the denotation of a constant c to be c^*, and we take the denotation of a function symbol f to be the function that given the objects $t_1{}^*, \ldots, t_n{}^*$ associated with terms $t_1, \ldots, t_n$ as arguments, yields as value the object $f(t_1, \ldots, t_n)^*$ associated with the term $f(t_1, \ldots, t_n)$. It follows by induction on complexity that the denotation of an arbitrary term t is the object t^* associated with it. Finally, we take as the denotation of a predicate P the relation that holds of objects the objects $t_1{}^*, \ldots, t_n{}^*$ associated with terms $t_1, \ldots, t_n$ if and only if the sentence $P(t_1, \ldots, t_n)$ is one of the A_i and ω assigns it the value 1. Thus truth-functional satisfiability and satisfiability in the ordinary sense come to the same thing for quantifier-free sentences.

Let now Γ be a set of $\forall$-formulas of some language L, and consider the set Δ of all instances $P(t_1, \ldots, t_n)$ obtained by substituting in sentences $\forall x_1 \ldots \forall x_n P(x_1, \ldots, x_n)$ of Γ terms $t_1, \ldots, t_n$ of L for the variables. If every finite subset of Δ is truth-functionally satisfiable, then every finite subset of Δ is satisfiable, and hence so is Δ, by the compactness theorem.

Moreover, by Proposition 12.7, if $\mathcal{A}$ is an interpretation in which every sentence in Δ comes out true, and $\mathcal{B}$ is the subinterpretation of $\mathcal{A}$ whose domain is the set of all denotations of closed terms, then every sentence in Δ also comes out true in $\mathcal{B}$. Since in $\mathcal{B}$ every element of the domain is the denotation of some term, from the fact that every instance $P(t_1, \ldots, t_n)$ comes out true it follows that the $\forall$-formula $\forall x_1 \ldots \forall x_n$ $P(x_1, \ldots, x_n)$ comes out true, and thus $\mathcal{B}$ is a model of Γ. Hence Γ is satisfiable.

Conversely, if Γ is satisfiable, then since a sentence implies all its substitution instances, every finite or infinite set of substitution instances of sentences in Γ will be satisfiable and hence truth-functionally satisfiable. Thus we have proved the following.

19.11 Theorem (Herbrand's theorem). Let Γ be a set of $\forall$-sentences. Then Γ is satisfiable if and only if every finite set of substitution instances of sentences in Γ is truth-functionally satisfiable.

It is possible to avoid dependence on the compactness theorem in the foregoing proof, by proving a kind of compactness theorem for truth-functional valuations, which is considerably easier than proving the ordinary compactness theorem. (Then, starting with the assumption that every finite subset of Δ is truth-functionally satisfiable, instead of arguing that each finite subset is therefore satisfiable, and hence that Δ is satisfiable by compactness, instead one would apply compactness for truth-functional satisfiability to conclude that Δ is truth-functionally satisfiable, from which it follows that Δ is satisfiable.) Herbrand's theorem actually then *implies* the compactness theorem: Given a set Γ of sentences, let $\Gamma^{\#}$ be the set of Skolem forms of sentences in Γ. We know from the preceding section that if every finite subset of Γ is satisfiable, then every finite subset of $\Gamma^{\#}$ is satisfiable and hence truth-functionally satisfiable, and so by Herbrand's theorem $\Gamma^{\#}$ is satisfiable, whence the original Γ is satisfiable.

Herbrand's theorem also implies the soundness and Gödel completeness theorems for an appropriate kind of proof procedure (different from that used earlier in this book and from those used in introductory textbooks), which we next describe. Suppose we are given a finite set Γ of sentences and wish to know if Γ is unsatisfiable. We first replace the sentences in Γ by Skolem forms: the proofs of the normal form theorems given in the preceding two sections implicitly provide an effective method of doing so. Now having the finite set $S_1, \ldots, S_n$, of $\forall$-sentences that are the Skolem forms of our original sentences, and any effective enumeration $t_1, t_2, t_3, \ldots$ of terms of the language, we set about effectively generating all possible substitution instances. (We could do this by first substituting in each formula for each of its variables the term t_1, then substituting for each variable in each formula either t_1 or t_2, then substituting for each variable in each formula either t_1 or t_2 or t_3, and so on. At each stage we get only finitely many substitution instances, namely, at stage m just k^m, where k is the total number of variables; but in the end we get them all.)

Each time we generate a new substitution instance, we check whether the finitely many instances we have generated so far are truth-functionally satisfiable. This can be done effectively, since on the one hand at any given stage we will have generated so far only finitely many substitution instances, so that there are only finitely many valuations to be considered (if the substitution instances generated so far involve m distinct atomic sentences, the number of possible valuations will be 2^m); while on the other hand, given a valuation ω and a truth-functional compound B of given atomic sentences A_i, we can effectively work out the value $\omega(B)$ required (the method of *truth tables* expounded in introductory textbooks being a way of setting out the work).

If any finite set of Skolem instances (that is, of substitution instances of Skolem forms) turns out to be truth-functionally unsatisfiable, then the original set Γ is unsatisfiable: producing such a set of Skolem instances is a kind of refutation of Γ. Conversely, if Γ is unsatisfiable, the above-described procedure will eventually produce such a refutation. This is because we know from the preceding section that Γ is unsatisfiable if and only if $\Gamma^{\#}$ is, and so, by Herbrand's theorem, Γ is unsatisfiable if and only if some finite set of substitution instances of Skolem forms is truth-functionally unsatisfiable. The refutation procedure just described is thus sound and complete (hence so would be the proof procedure that proves Γ implies D by refuting $\Gamma \cup \{\sim D\}$).

19.4 Eliminating Function Symbols and Identity

While the presence of identity and function symbols is often a convenience, their absence can often be a convenience, too, and in this section we show how they can, in a sense to be made precise, be ‘eliminated’. (Constants will be treated as a special sort of function symbol, namely 0-place function symbols. Whatever we say about function symbols in this section goes for constants, too, and they will not be given separate consideration.)

Let us take up the elimination of function symbols first. The first fact we need is that any sentence is logically *equivalent* to one in which all function symbols occur immediately to the right of the identity symbol. This means that no function symbols occur in the blanks to the right of predicates other than the identity predicate, or in the blanks to the right of a function symbol, or in the blank to the left of the identity symbol, so the only occurrences of an n-place function symbol f are in atomic subformulas of the type $v = f(u_1, \ldots, u_n)$, where v and the u_i are variables (not necessarily all distinct).

The proof is quite simple: Suppose that S is a sentence with at least one occurrence of a function symbol f in a position other than immediately to the right of the identity symbol. In any such occurrence, f occurs as the first symbol in some term t that occurs (possibly as a subterm of a more complex term) in some atomic subformula $A(t)$. Let v be any variable not occurring in S, and let S^- be the result of replacing $A(t)$ by the logically equivalent $\exists v(v = t \,\&\, A(v))$. Then S is logically equivalent to S^-, and S^-contains one fewer occurrence of function symbols in positions other than immediately to the right of the identity symbol. Reducing the number of such occurrences one at a time in this way, S is ultimately equivalent to a sentence with no such occurrences. So for the remainder of this chapter, we consider only sentences without such occurrences.

We show how to eliminate function symbols one at a time from such sentences. (The process may be repeated until all function symbols, including constants, have been eliminated.) If S is such a sentence and f an n-place function symbol occurring in it, let R be a new $(n+1)$-place predicate. Replace each subformula of the type $v = f(u_1, \ldots, u_n)$ in which f occurs—and remember, these are the *only* kind of occurrences of f in S—by $R(u_1, \ldots, u_n, v)$, and call the result $S^{\pm}$. Let C be the following sentence, which we call the *functionality axiom*:

$$\forall x_1 \ldots \forall x_n \exists y \forall z (R(x_1, \ldots, x_n, z) \leftrightarrow z = y).$$

Let D be the following sentence, which we call the *auxiliary axiom*:

$$\forall x_1 \ldots \forall x_n \forall z (R(x_1, \ldots, x_n, z) \leftrightarrow z = f(x_1, \ldots, x_n)).$$

The precise sense in which the symbol f is ‘dispensable’ is indicated by the following proposition (and its proof).

19.12 Proposition. S is satisfiable if and only if $S^{\pm}$ & C is satisfiable.

Proof: Let us begin by sorting out the relationships among the various sentences we have introduced. If we call the language to which the original sentence S belonged L, the language obtained by adding the new predicate R to this language L^+, and the language obtained by removing the old function symbol f from the latter language $L^{\pm}$, then S belongs to L, D to L^+, and $S^{\pm}$ and C to $L^{\pm}$. Note that D implies C, and D implies $S \leftrightarrow S^{\pm}$.

Now note that every interpretation $\mathcal{M}$ of L has a unique expansion to an interpretation $\mathcal{N}$ of L^+ that is a model of D. The one and only way to obtain such an $\mathcal{N}$ is to take as the denotation $R^{\mathcal{N}}$ of the new predicate the relation that holds of $a_1, \ldots, a_n$, b if and only if $b = f^{\mathcal{M}}(a_1, \ldots, a_n)$. Also, every interpretation $\mathcal{P}$ of $L^{\pm}$ that is a model of C has a unique expansion to an interpretation $\mathcal{N}$ of L^+ that is a model of D. The one and only way to obtain such an $\mathcal{N}$ is to take as the denotation $f^{\mathcal{N}}$ of the new function symbol the function that given $a_1, \ldots, a_n$ as arguments yields as value the unique b such that $R^{\mathcal{P}}(a_1, \ldots, a_n, b)$ holds. (The truth of C in P is need to guarantee that there will exist such a b and that it will be unique.)

If S has a model $\mathcal{M}$, by our observations in the preceding paragraph it has an expansion to a model $\mathcal{N}$ of S & D. Then since D implies $S \leftrightarrow S^{\pm}$ and C, $\mathcal{N}$ is a model of $S^{\pm}$ & C. Conversely, if $S^{\pm}$ & C has a model $\mathcal{P}$, then by our observations in the preceding paragraph, it has an expansion to a model $\mathcal{N}$ of $S^{\pm}$ & D. Then since D implies $S \leftrightarrow S^{\pm}$, $\mathcal{N}$ is a model of S.

We now turn to the matter of eliminating the identity symbol, supposing that function symbols have already been eliminated. Thus we begin with a language L whose only nonlogical symbols are predicates. We add a further two-place relation-symbol $\equiv$ and consider the following sentence E, which we have already encountered in chapter 12, and will call the *equivalence axiom*:

$$\begin{aligned}&\forall x\, x \equiv x\ \&\\ &\forall x \forall y (x \equiv y \to y \equiv x)\ \&\\ &\forall x \forall y \forall z ((x \equiv y\ \&\ y \equiv z) \to x \equiv z).\end{aligned}$$

In addition, for each predicate P of L we consider the following sentence C_P, which we will call the *congruence* axiom for P:

$$\forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n ((x_1 \equiv y_1\ \&\ \ldots\ \&\ x_n \equiv y_n) \to \\ (P(x_1,\ \ldots, x_n) \leftrightarrow P(y_1,\ \ldots, y_n))).$$

Note that the result of replacing the new sign $\equiv$ by the identity sign $=$ in E or any C_P is a logically valid sentence. For any sentence S, we let S^* be the result of replacing the identity sign $=$ throughout by this new sign $\equiv$, and C_S the conjunction of the C_P for all predicates P occurring in S. The precise sense in which the symbol $=$ is 'dispensable' is indicated by the following proposition (and its proof).

19.13 Proposition. S is satisfiable if and only if $S^*\ \&\ E\ \&\ C_S$ is satisfiable.

Proof: One direction is easy. Given a model of S, we get a model of S^* & E & C_S by taking the identity relation as the denotation of the new sign.

For the other direction, suppose we have a model $\mathcal{A}$ of S^* & E & C_S. We want to show there is a model $\mathcal{B}$ of S. Since E is true in $\mathcal{A}$, the denotation $\equiv^{\mathcal{A}}$ of the new sign in $\mathcal{A}$ is an equivalence relation on the domain $|\mathcal{A}|$. We now specify an interpretation $\mathcal{B}$ whose domain $|\mathcal{B}|$ will be the set of all equivalence classes of elements of $|\mathcal{A}|$. We need to specify what the denotation $P^{\mathcal{B}}$ of each predicate P of the original language is to be. For any equivalence classes $b_1, \ldots, b_n$ in $|\mathcal{B}|$, let $P^{\mathcal{B}}$ hold of them if and only if $P^{\mathcal{A}}$ holds of $a_1, \ldots, a_n$ for some a_1 in $b_1, \ldots,$ and a_n in b_n. We also need to specify what the denotation $\equiv^{\mathcal{B}}$ of the new sign is to be. We take it to be the genuine identity relation.

Let now j be the function from $|\mathcal{A}|$ to $|\mathcal{B}|$ whose value for argument a is the equivalence class of a. If $P^{\mathcal{A}}(a_1, \ldots, a_n)$ holds, then by definition of $P^{\mathcal{B}}$, $P^{\mathcal{B}}(j(a_1), \ldots, j(a_n))$ holds; while if $P^{\mathcal{B}}(j(a_1), \ldots, j(a_n))$ holds, then again by definition of $P^{\mathcal{B}}$, $P^{\mathcal{A}}(a'_1, \ldots, a'_n)$ holds for some a'_i, where each a'_i belongs to the same equivalence class $j(a'_i) = j(a_i)$ as a_i. The truth of C_P in A guarantees that in that case $P^{\mathcal{A}}(a_1, \ldots, a_n)$ holds. Trivially, $a_1 \equiv^{\mathcal{A}} a_2$ holds if and only if $j(a_1) = j(a_2)$, which is to say, if and only if $j(a_1) \equiv^{\mathcal{B}} j(a_2)$ holds. Thus the function j has all the properties of an isomorphism except for not being one-to-one. If we look at the proof of the isomorphism lemma, according to which exactly the same sentences are true in isomorphic interpretations, we see that the property of being one-to-one was used *only* in connection with identity. Hence, so far as sentences *not* involving identity are concerned, by the same proof as that of the isomorphism lemma, the same ones are true in $\mathcal{B}$ as in $\mathcal{A}$. (See Proposition 12.5 and its proof.) In particular S^* is true in $\mathcal{B}$. But since $\equiv^{\mathcal{B}}$ is the genuine identity relation, it follows that the result of replacing $\equiv$ by $=$ in S^* will also be true in $\mathcal{B}$—and the result of this substitution is precisely the original S. So we have a model $\mathcal{B}$ of S as required.

Propositions 19.12 and 19.13 can both be stated more generally. If Γ is any set of sentences and $\Gamma^{\pm}$ the set of all $S^{\pm}$ for S in Γ, together with all functionality axioms, then Γ is satisfiable if and only if $\Gamma^{\pm}$ is. If Γ is any set of sentences not involving function symbols, and Γ^* is the set of all S^* for S in Γ together with the equivalence axiom and all congruence axioms, then Γ is satisfiable if and only if Γ^* is satisfiable. Applications of the function-free and identity-free normal forms of the present section will be indicated in the next two chapters.

Problems

19.1 Find equivalents

(a) in negation-normal form

(b) in disjunctive normal form

(c) in full disjunctive normal form

for $\sim((\sim A \,\&\, B) \vee (\sim B \,\&\, C)) \vee \sim(\sim A \vee C)$.

19.2 Find equivalents in prenex form for

(a) $\exists x(P(x) \rightarrow \forall x P(x))$

(b) $\exists x(\exists x\, P(x) \rightarrow P(x))$.

19.3 Find an equivalent in prenex form for the following, and write out its Skolem form:

$$\forall x(Qx \rightarrow \exists y(Py \,\&\, Ryx)) \leftrightarrow \exists x(Px \,\&\, \forall y(Qy \rightarrow Rxy).$$

19.4 Let T be a set of finite sequences of 0s and 1s such that any initial segment $(e_0, \ldots, e_{m-1})$, $m < n$, of any element $(e_0, \ldots, e_{n-1})$ in T is in T. Let T^* be the subset of T consisting of all finite sequences s such that there are infinitely many finite sequences t in T with s is an initial segment of t. Show that if T is infinite, then there is an infinite seqeunce $e_1, e_2, \ldots$ of 0s and 1s such that every initial segment $(e_0, \ldots, e_{m-1})$ is in $T*$.

19.5 State and prove a compactness theorem for truth-functional valuations.

20

The Craig Interpolation Theorem

Suppose that a sentence A implies a sentence C. The Craig interpolation theorem *tells us that in that case there is a sentence B such that A implies B, B implies C, and B involves no nonlogical symbols but such as occur both in A and in B. This is one of the basic results of the theory of models, almost on a par with, say, the compactness theorem. The proof is presented in section 20.1. The proof for the special case where identity and function symbols are absent is an easy further application of the same lemmas that we have applied to prove the compactness theorem in Chapter 13, and could have been presented there. But the easiest proof for the general case is by reduction to this special case, using the machinery for the elimination of function symbols and identity developed in section 19.4. Sections 20.2 and 20.3, which are independent of each other, take up two significant corollaries of the interpolation theorem,* Robinson's joint consistency theorem *and* Beth's definability theorem.

20.1 Craig's Theorem and Its Proof

We begin with a simple observation.

20.1 Proposition. If a sentence A implies a sentence C, then there is a sentence B that A implies, that implies C, and that contains only such constants as are contained in both of A and C.

Proof: The reason is clear: If there are no constants in A not in C, we may take A for our B; otherwise, let $a_1, \ldots, a_n$ be all the constants in A and not in C, and let A^* be the result of replacing each a_i by some new variable v_i. Then, since $A \rightarrow C$ is valid, so is $\forall v_1 \cdots \forall v_n(A^* \rightarrow C)$, and hence so is $\exists v_1 \cdots \exists v_n A^* \rightarrow C$. Then $\exists v_1 \cdots \exists v_n A^*$ is a suitable B, for A implies it, it implies C, and all constants in it are in both A and C.

It might occur to one to ask whether the fact just proved about constants can be subsumed under one about constants, function symbols, and predicates; that is, to ask whether if A implies C, there is always a sentence B that A implies, that implies C, and that contains only constants, function symbols, and predicates that are in both A and C. The answer to the question, as stated, is *no.*

20.2 Example (A failure of interpolation). Let A be $\exists x Fx \,\&\, \exists x \sim Fx$, and let C be $\exists x \exists y\, x \neq y$. Then A implies C, but there is no sentence at all that contains only constants, function symbols, and predicates that are in both A and C, and therefore there is no such sentence that A implies and that implies C.

Suppose we do not count the *logical* predicate of identity, and ask whether, if A implies C, there is always a sentence B that A implies, that implies C, and that contains no nonlogical symbols (that is, no constants, no function symbols, and no *nonlogical* predicates) except such as are both in A and in C. The Craig interpolation theorem is the assertion that the answer to our question, thus restated, is *yes*.

20.3 Theorem (Craig interpolation theorem). If A implies C, then there is a sentence B that A implies, that implies C, and that contains no nonlogical symbols except such as are both in A and in C.

Such a B is called an *interpolant* between A and C. Before launching into the proof, let us make one clarificatory observation.

20.4 Example (Degenerate cases). It may happen that we need to allow the identity symbol to appear in the interpolant even though it appears in neither A nor C. Such a situation can arise if A is unsatisfiable. For instance, if A is $\exists x(Fx \,\&\, {\sim}Fx)$ and C is $\exists x Gx$, then $\exists x\, x \neq x$ will do for B, but there are no sentences at all containing only predicates that occur in both A and C, since there are no such predicates. A similar situation can arise if C is valid. For instance, if A is $\exists x Fx$ and C is $\exists x(Gx \vee {\sim}Gx)$, then $\exists x\, x = x$ will do for B, but again there are no predicates that occur in both A and C. Note that $\exists x\, x \neq x$ will do for an interpolant in *any* case where A is unsatisfiable, and $\exists x\, x = x$ in *any* case where C is valid. (We could avoid the need for identity if we admitted the logical constants $\top$ and $\bot$ of section 19.1.)

Proof, Part I: This noted, we may restrict our attention to cases where A is satisfiable and C is not valid. The proof that, under this assumption, an interpolant B exists will be, like so many other proofs, divided into two parts. First we consider the case where identity and function symbols are absent, then reduce the general case where they are present to this special case. (The proof, unlike that of Proposition 20.1, will be nonconstructive. It will prove the *existence* of an interpolant, without showing how to *find* one. More constructive proofs are known, but are substantially longer and more difficult.)

Let us begin immediately on the proof of the special case. Considering only sentences and formulas without identity or function symbols, let A be a sentence that is satisfiable and C a sentence that is not valid (which is equivalent to saying that ${\sim}C$ is satisfiable), such that A implies C (which is equivalent to saying $\{A, {\sim}C\}$ is unsatisfiable). We want to show there is a sentence B containing only predicates common to A and C, such that A implies B and B implies C (which is equivalent to saying ${\sim}C$ implies ${\sim}B$).

What we are going do is to show that if there is no such interpolant B, then $\{A, {\sim}C\}$ is after all satisfiable. To show this we apply the model existence theorem

(Lemma 13.3). This tells us that if L is a language containing all the nonlogical symbols of A and of C, and if L^* is a language obtained by adding infinitely many new constants to L, then $\{A, \sim C\}$ will be satisfiable provided it belongs to some set S of sets of sentences of L^* having certain properties. For the present situation, where identity and function symbols are absent, these properties are as follows:

(S0) If Γ is in S and Γ_0 is a subset of Γ, then Γ_0 is in S.

(S1) If Γ is in S, then for no sentence D are both D and $\sim D$ in Γ.

(S2) If Γ is in S and $\sim\sim D$ is in Γ, then $\Gamma \cup \{D\}$ is in S.

(S3) If Γ is in S and $(D_1 \vee D_2)$ is in Γ, then $\Gamma \cup \{D_i\}$ is in S for either $i = 1$ or $i = 2$.

(S4) If Γ is in S and $\sim(D_1 \vee D_2)$ is in Γ, then $\Gamma \cup \{\sim D_i\}$ is in S for both $i = 1$ and $i = 2$.

(S5) If Γ is in S and $\{\exists x F(x)\}$ is in Γ, and then $\Gamma \cup \{F(b)\}$ is in S for any constant b not in Γ or $\exists x F(x)$.

(S6) If Γ is in S and $\sim\exists x F(x)$ is in Γ, then $\Gamma \cup \{\sim F(b)\}$ is in S for any constant b at all.

What we need to do is to define a set S, use the hypothesis that there is no interpolant B to show $\{A, \sim C\}$ is in S, and establish properties (S0)–(S1) for S. Towards defining S, call a sentence D of L^* a *left* formula (respectively, a *right* formula) if every predicate in D is in A (respectively, is in C). If Γ_L is a satisfiable set of left sentences and Γ_R a satisfiable set of right sentences, let us say that B *bars* the pair Γ_L, Γ_R if B is both a left and a right sentence and Γ_L implies B while Γ_R implies $\sim B$. Our assumption that there is no interpolant, restated in this terminology, is the assumption that no sentence B of L bars $\{A\}, \{\sim C\}$. It follows—by a proof quite like that of Proposition 20.1—that no sentence B of L^* bars $\{A\}, \{\sim C\}$. Let S be the set of all Γ that admit and *unbarred division* in the sense that we can write Γ as a union $\Gamma_L \cup \Gamma_R$ of two sets of sentences where Γ_L consists of left and Γ_R of right sentences, each of Γ_L and Γ_R is satisfiable, and no sentence bars the pair Γ_L, Γ_R. Then what we have said so far is that $\{A, \sim C\}$ is in S. What remains to be done is to establish properties (S0)–(S6) for S.

(S0) is easy and left to the reader. For (S1), if $\Gamma = \Gamma_L \cup \Gamma_R$ is an unbarred division, then the assumptions that Γ_L is satisfiable implies that there is no sentence D with both D and $\sim D$ in Γ_L. Similarly for Γ_R. Nor can there be a D with D in Γ_L and $\sim D$ in Γ_R, for in that case D would be both a left sentence (since it belongs to Γ_L) and a right sentence (since it belongs to Γ_R) and therefore would be a sentence that is implied by Γ_L and whose negation is implied by Γ_R, and so would bar Γ_L, Γ_R. Similarly, the reverse case with $\sim D$ in Γ_L and D in Γ_R is impossible, since $\sim D$ would bar the pair Γ_L, Γ_R.

For (S2), suppose $\Gamma = \Gamma_L \cup \Gamma_R$ is an unbarred division and $\sim\sim D$ is in Γ. There are two cases according as $\sim\sim D$ is in Γ_L or in Γ_R, but the two are just alike, and we consider only the former. Then since $\sim\sim D$ is a left formula, so is D, and since D is implied by Γ_L, adding D to Γ_L cannot make it unsatisfiable if it was satisfiable before, nor can it make any sentence B a consequence that was not a consequence before. So $\Gamma \cup \{D\} = (\Gamma_L \cup \{D\}) \cup \Gamma_R$ is an unbarred division, and $\Gamma \cup \{D\}$ is in S. (S4)–(S6) are very similar.

(S3) is just slightly different. Suppose $\Gamma = \Gamma_L \cup \Gamma_R$ is an unbarred division and $(D_1 \vee D_2)$ is in Γ. Again there are two cases, and we consider only the one where $(D_1 \vee D_2)$ is in Γ_L. Each of the D_i is of course a left sentence, since their disjunction is. We claim that $\Gamma \cup \{D_i\} = (\Gamma_L \cup \{D_i\}) \cup \Gamma_R$ is an unbarred division for at least one i. Towards showing this, note that if $\Gamma_L \cup \{D_1\}$ is unsatisfiable, then Γ_L implies both $(D_1 \vee D_2)$ and $\sim D_1$, and hence implies D_2. In this case, the proof that $(\Gamma_L \cup \{D_2\}) \cup \Gamma_R$ is an unbarred division is just like the proof of the preceding paragraph. Similarly, if $\Gamma_L \cup \{D_2\}$ is unsatisfiable, then $(\Gamma_L \cup \{D_2\}) \cup \Gamma_R$ gives an unbarred division. So we are left to treat the case where $\Gamma_L \cup \{D_i\}$ is satisfiable for both i. In this case, $(\Gamma_L \cup \{D_i\}) \cup \Gamma_R$ can fail to give an unbarred division only because there is a sentence B_i that bars the pair $\Gamma_L \cup \{D_i\}$, Γ_R. What we claim is that there cannot exist such B_i both both $i = 1$ and $i = 2$. For suppose there did. Then since B_i is implied by $\Gamma_L \cup \{D_i\}$ for each i, and Γ_L contains $(D_1 \vee D_2)$, it follows that $B = (B_1 \vee B_2)$ is implied by Γ_L. Moreover, since each $\sim B_i$ is implied by Γ_R, so is $\sim B$. Finally, since each B_i is both a left and a right sentence, the same is true of B. Thus there is a sentence B that bars Γ_L, Γ_R, contrary to hypothesis. This contradiction completes the proof for the case where identity and function symbols are absent.

Proof, Part II: We next consider the case where identity is present but function symbols are still absent. Suppose A implies C. As in section 18.4, we introduce the new two-place predicate $\equiv$. We write E_L for the conjunction of the equality axioms and the congruence axioms for predicates in A, and E_R for the corresponding sentence for C. We write $*$ to indicate replacing $=$ by $\equiv$. Since A implies C, $A \mathbin{\&} \sim C$ is unsatisfiable. What the proof of Proposition 19.13 tells us is that therefore $E_L \mathbin{\&} E_R \mathbin{\&} A^* \mathbin{\&} \sim C^*$ is unsatisfiable. It follows that $E_L \mathbin{\&} A^*$ implies $E_R \rightarrow C^*$. By the interpolation theorem for sentences without identity, there is a sentence B^* involving only $\equiv$ and nonlogical predicates common to A and C, such that $E_L \mathbin{\&} A^*$ implies B^* and B^* implies $E_R \rightarrow C^*$. It follows that $E_L \mathbin{\&} A^* \mathbin{\&} \sim B^*$ and $E_R \mathbin{\&} B^* \mathbin{\&} C^*$ are unsatisfiable. Then we claim B, the result of replacing $\equiv$ by $=$ in B^*, is the required interpolant between A and C. Certainly its nonlogical predicates are common to A and C. Further, what the proof of Proposition 18.13 tell us is is that $A \mathbin{\&} \sim B$ and $B \mathbin{\&} \sim C$ are unsatisfiable, and therefore A implies B and B implies C. The treatment of function symbols is much the same, but using the machinery of Proposition 19.12 rather than of Proposition 19.13. Details are left to the reader.

In the remaining sections of this chapter we apply the interpolation theorem to prove two results about theories: one about the conditions under which the union of two theories is satisfiable, the other about the conditions under which definitions are consequences of theories. (Throughout 'theory' is being used, as elsewhere in this book, in a very broad way: A *theory* in a language is just a set of sentences of the language that contains every sentence of that language that is a logical consequence of the set. A *theorem* of a theory is just a sentence in that theory.)

20.2 Robinson's Joint Consistency Theorem

We begin with a preliminary result.

20.5 Lemma. The union $T_1 \cup T_2$ of two theories T_1 and T_2 is satisfiable if and only if there is no sentence in T_1 whose negation is in T_2.

Proof: The 'only if' part is obvious: if there were a sentence in T_1 whose negation was T_2, the union could not possibly be satisfiable; for there could be no interpretation in which both the sentence and its negation were true.

The 'if' part follows quickly from the compactness theorem and Craig's theorem: Suppose the union of T_1 and T_2 is unsatisfiable. Then by the compactness theorem, there is a finite subset S_0 of the union which is unsatisfiable. If there are no members of S_0 that belong to T_1, then T_2 is unsatisfiable, and so $\forall x\, x = x$ is a sentence in T_1 whose negation is in T_2; if no members of S_0 belong to T_2, then T_1 is unsatisfiable, and so $\sim\forall x\, x = x$ is a sentence in T_1 whose negation is in T_2. So we may suppose that S_0 contains some members both of T_1 and of T_2. Let $F_1, \ldots, F_m$ be the members of S_0 that are in T_1; let $G_1, \ldots, G_n$ be the members of S_0 that are in T_2.

Let A be $(F_1 \,\&\, \ldots \,\&\, F_m)$, and let C be $\sim(G_1 \,\&\, \ldots \,\&\, G_n)$. A implies C. By Craig's theorem, there is a sentence B implied by A, implying C, and containing only nonlogical symbols contained in both A and C. B is therefore a sentence in the languages of both T_1 and T_2. Since A is in T_1 and implies B, B is in T_1. Since $(G_1 \,\&\, \ldots \,\&\, G_n)$ is in T_2, so is $\sim B$, as $(G_1 \,\&\, \ldots \,\&\, G_n)$ implies $\sim B$. So B is a sentence in T_1 whose negation is in T_2.

An *extension* T' of a theory T is just another theory containing it. The extension is called *conservative* if every sentence of the language of T that is a theorem of T' is a theorem of T. We next prove a theorem about conservative extensions.

20.6 Theorem. Let L_0, L_1, L_2 be languages, with $L_0 = L_1 \cap L_2$. Let T_i be a theory in L_i for $i = 0, 1, 2$. Let T_3 be the set of sentences of $L_1 \cup L_2$ that are consequences of $T_1 \cup T_2$. Then if T_1 and T_2 are both conservative extensions of T_0, then T_3 is also a conservative extension of T_0.

Proof: Suppose B is a sentence of L_0 that is a theorem of T_3. We must show that B is a theorem of T_0. Let U_2 be the set of sentences of L_2 that are consequences of $T_2 \cup \{\sim B\}$. Since B is a theorem of T_3, $T_1 \cup T_2 \cup \{\sim B\}$ is unsatisfiable, and therefore $T_1 \cup U_2$ is unsatisfiable. Therefore there is a sentence D in T_1 whose negation $\sim D$ is in U_2. D is a sentence of L_1, and $\sim D$ of L_2. Thus D and $\sim D$ are both in L_0, and hence so is $(\sim B \rightarrow \sim D)$. Since D is in T_1, which is a conservative extension of T_0, D is in T_0. And since $\sim D$ is in U_2, $(\sim B \rightarrow \sim D)$ is in T_2, which is a conservative extension of T_0. Thus $(\sim B \rightarrow \sim D)$ is also in T_0, and therefore so is B, which follows from D and $(\sim B \rightarrow \sim D)$.

An immediate consequence is

20.7 Corollary (Robinson's joint consistency theorem). Let L_0, L_1, L_2 be languages, with $L_0 = L_1 \cap L_2$. Let T_i be a theory in L_i for $i = 0, 1, 2$. If T_0 is complete, and T_1 and T_2 are satisfiable extensions of T_0, then $T_1 \cup T_2$ is satisfiable.

Proof: A satisfiable extension of a complete theory is conservative, and a conservative extension of a satisfiable theory is satisfiable. Thus if the T_i satisfy the hypotheses of Corollary 20.7, then T_3 as defined in Theorem 20.6 is a satisfiable extension of T_0, and therefore $T_1 \cup T_2$ is satisfiable.

20.8 Example (Failures of joint consistency). Let $L_0 = L_1 = L_2 = \{P, Q\}$, where P and Q are one-place predicates. Let T_1 (respectively, T_2) be the set of consequences in L_1 (respectively, L_2) of $\{\forall x Px, \forall x Qx\}$ (respectively, $\{\forall x Px, \forall x \sim Qx\}$. Let T_0 be the set of consequences in L_0 of $\forall x Px$. Then $T_1 \cup T_2$ is not satisfiable, though each of T_1 and T_2 is a satisfiable extension of T_0. This is not a counterexample to Robinson's theorem, because T_0 is not complete. If instead we let $L_0 = \{P\}$, then again we do not get a counterexample, because then L_0 is not the intersection of L_1 and L_2, while L is. This shows the hypotheses in Corollary 20.7 are needed.

We have proved Robinson's theorem using Craig's theorem. Robinson's theorem can also be proved a different way, not using Craig's theorem, and then used to prove Craig's theorem. Let us indicate how a 'double compactness' argument yields Craig's theorem from Robinson's.

Suppose A implies C. Let L_1 (L_2) be the language consisting of the nonlogical symbols occurring in A (C). Let $L_0 = L_1 \cap L_2$. We want to show there is a sentence B of L_0 implied by A and implying C. Let Δ be the set of sentences of L_0 that are implied by A. We first show that $\Delta \cup \{\sim C\}$ is unsatisfiable. Suppose that it is not and that $\mathcal{M}$ is a model of $\Delta \cup \{\sim C\}$. Let T_0 be the set of sentences of L_0 that are true in $\mathcal{M}$. T_0 is a complete theory whose langauge is L_0. Let T_1 (T_2) be the set of sentences of L_1 (L_2) that are consequences of $T_0 \cup \{A\}$ ($T_0 \cup \{\sim C\}$). T_2 is a satisfiable extension of T_0: $\mathcal{M}$ is a model of $T_0 \cup \{\sim C\}$, and hence of T_2. But $T_1 \cup T_2$ is not satisfiable: any model of $T_1 \cup T_2$ would be a model of $\{A, \sim C\}$, and since A implies C, there is no such model. Thus by the joint consistency theorem, T_1 is not a satisfiable extension of T_0, and therefore $T_0 \cup \{A\}$ is unsatisfiable. By the compactness theorem, there is a finite set of sentences in T_0 whose conjunction D, which is in L_0, implies $\sim A$. Thus A implies $\sim D$, $\sim D$ is in L_0, $\sim D$ is in Δ, and $\sim D$ is therefore true in $\mathcal{M}$. But this is a contradiction, as all of the conjuncts of D are in T_0 and are therefore true in $\mathcal{M}$. So $\Delta \cup \{\sim C\}$ is unsatisfiable, and by the compactness theorem again, there is a finite set of members of Δ whose conjunction B implies C. B is in L_0, since its conjuncts are, and as A implies each of these, A implies B.

20.3 Beth's Definability Theorem

Beth's definability theorem is a result about the relation between two different explications, or ways of making precise, the notion of a *theory's giving a definition of one concept in terms of other concepts*. As one might expect, each of the explications discusses a relation that may or may not hold between a theory, a symbol in the language of that theory (which is supposed to 'represent' a certain concept), and other symbols in the language of the theory (which 'represent' other concepts), rather than directly discussing a relation that may or may not hold between a theory, a concept, and other concepts. The supposition of Beth's theorem, then, is that α and $\beta_1, \ldots, \beta_n$ are nonlogical symbols of the language L of some theory T and that α is not among the β_i.

The first explication is straightforward and embodies the idea that a theory defines a concept in terms of others when 'a definition of that concept in terms of the others is a consequence of the theory'. This sort of definition is called an explicit definition: we say that α is *explicitly definable* in terms of the β_i in T if a definition of α from the β_i is one of the sentences in T. What precisely is meant by a definition of α in terms of the β_i depends on whether α is a predicate or a function symbol. In the case of a $(k + 1)$-place predicate, such a definition is a sentence of the form

$$\forall x_0 \forall x_1 \cdots \forall x_k(\alpha(x_0, x_1, \ldots, x_k) \leftrightarrow B(x_0, \ldots, x_k))$$

and in case of a k-place function symbol, such a definition is a sentence of the form

$$\forall x_0 \forall x_1 \cdots \forall x_k(x_0 = \alpha(x_1, \ldots, x_k) \leftrightarrow B(x_0, \ldots, x_k))$$

where in either case B is a formula whose only nonlogical symbols are among the β_i. (Constants may be regarded as 0-place function symbols, and do not require separate discussion. In this case the right side of the biconditional would simply be $x_0 = \alpha$.) The general form of a definition may be represented as

$$\forall x_0 \cdots \forall x_k(\text{—}\alpha, x_0, \ldots, x_k\text{—} \leftrightarrow B(x_0, \ldots, x_k)).$$

The second explication is rather more subtle, and incorporates the idea that a theory defines a concept in terms of others if 'any specification of the universe of discourse of the theory and the meanings of the symbols representing the other concepts (that is compatible with the truth of all the sentences in the theory) uniquely determines the meaning of the symbol representing that concept'. This sort of definition is called implicit definition: we say that α is *implicitly definable* from the β_i in T if any two models of T that have the same domain and agree in what they assign to the β_i also agree in what they assign to α.

It will be useful to develop a more 'syntactic' reformulation of this 'semantic' definition of implicit definability. To this end, we introduce a new language L' obtained from L by replacing every nonlogical symbol γ of L, other than the β_i, by a new symbol γ' of the same kind: 17-place function symbols are replaced by 17-place function symbols, 59-place predicates by 59-place predicates, and so on.

Given two models $\mathcal{M}$ and $\mathcal{N}$ of T that have the same domain and agree on what they assign to the β_i, we let $\mathcal{M} + \mathcal{N}$ be the interpretation of $L \cup L'$ that has the same domain, and assigns the same denotations to the β_i, and, for any other nonlogical symbol γ of L, assigns to γ what $\mathcal{M}$ assigns to γ, and assigns to γ' what $\mathcal{N}$ assigns to γ. Then $\mathcal{M} + \mathcal{N}$ is a model of $T \cup T'$.

Conversely, if $\mathcal{K}$ is a model of $T \cup T'$, then $\mathcal{K}$ can clearly be 'decomposed' into two models $\mathcal{M}$ and $\mathcal{N}$ of T, which have the same domain (as each other, and as $\mathcal{K}$) and agree (with each other and with $\mathcal{K}$) on what they assign to the β_i, where for any other nonlogical symbol γ of L, what $\mathcal{M}$ assigns to γ is what $\mathcal{K}$ assigns to γ, and what $\mathcal{N}$ assigns to γ is what $\mathcal{K}$ assigns to γ'.

20.9 Lemma. α is implicitly definable from $\beta_1, \ldots, \beta_n$ in T if and only if

$$\forall x_0 \cdots \forall x_k(\text{—}\alpha, x_0, \ldots, x_k\text{—} \leftrightarrow \text{—}\alpha', x_0, \ldots, x_k\text{—}) \tag{1}$$

is a consequence of $T \cup T'$.

Proof: Here, of course, by —$\alpha', x_0, \ldots, x_k$— is meant the result of substituting α' for α in —$\alpha, x_0, \ldots, x_k$—. Note that (1) will be true in a given interpretation if and only if that interpretation assigns the same denotation to α and to α'.

For the left-to-right direction, suppose α is implicitly definable from the β_i in T. Suppose $\mathcal{K}$ is a model of $T \cup T'$. Let $\mathcal{M}$ and $\mathcal{N}$ be the models into which $\mathcal{K}$ can be decomposed as above, so that $\mathcal{K} = M + N$. Then $\mathcal{M}$ and $\mathcal{N}$ have the same domain and agree on what they assign to the β_i. By the supposition of implicit definability, they must therefore agree on what they assign to α. Therefore the biconditional (1) is true in $\mathcal{K}$. In other words, any model of $T \cup T'$ is a model of (1), which therefore is a consequence of $T \cup T'$.

For the right-to-left direction, suppose that (1) follows from $T \cup T'$. Suppose $\mathcal{M}$ and $\mathcal{N}$ are models of T that have the same domain and agree on what they assign to the β_i. Then $\mathcal{M} + \mathcal{N}$ is a model of $T \cup T'$ and therefore of (1), by the supposition that (1) is a consequence of $T \cup T'$. It follows that $\mathcal{M} + \mathcal{N}$ assigns the same denotation to α and α', and therefore that $\mathcal{M}$ and $\mathcal{N}$ assign the same denotation to α. Thus α is implicitly definable from the β_i in T.

One direction of the connection between implicit and explicit definability is now easy.

20.10 Proposition (Padoa's method). If α is not implicitly definable from the β_i in T, then α is not explicitly definable from the β_i in T.

Proof: Suppose α *is* explicitly definable from the β_i in T. Then some definition

$$\forall x_0 \cdots \forall x_k(\text{—}\alpha, x_0, \ldots, x_k\text{—} \leftrightarrow B(x_0, \ldots, x_k)) \tag{2}$$

of α from the β_i is in T. Therefore

$$\forall x_0 \cdots \forall x_k(\text{—}\alpha', x_0, \ldots, x_k\text{—} \leftrightarrow B(x_0, \ldots, x_k)) \tag{3}$$

is in T'. (Recall that B involves only the β_i, which are *not* replaced by new nonlogical symbols.) Since (1) of Lemma 20.9 is a logical consequence of (2) and (3), it is a consequences of $T \cup T'$, and by that lemma, α is implicitly definable from the β_i in T.

20.11 Theorem (Beth's definability theorem). α is implicitly definable from the β_i in T if and only if α is explicitly definable from the β_i in T.

Proof: The 'if' direction is the preceding proposition, so it only remains to prove the 'only if' direction. So suppose α is implicitly definable from the β_i in T. Then (1) of Lemma 20.9 is a consequence of $T \cup T'$. By the compactness theorem, it is a consequence of some finite subset of $T \cup T'$. By adding finitely many extra sentences to it, if necessary, we can regard this finite subset as $T_0 \cup T_0'$, where T_0 is a finite subset of T, and T_0' comes from T_0 on replacing each nonlogical symbol γ other than the β_i

by γ'. Let $A(A')$ be the conjunction of the members of T_0 (T_0'). Then (1) is implied by A & A' Let $c_0, \ldots, c_k$ be constants not occurring in $T \cup T'$, and hence not in A, A', —$\alpha, x_0, \ldots, x_k$—, or —$\alpha', x_0, \ldots, x_k$—. Then

$$—\alpha, c_0, \ldots, c_k— \leftrightarrow —\alpha', c_0, \ldots, c_k—$$

is a consequence of (1) and therefore of A & A'. Here of course by —$\alpha, c_0, \ldots, c_k$— is meant the result of substituting c_i for x_i in —$\alpha, x_0, \ldots, x_k$—for all i, and similarly for —$\alpha', c_0, \ldots, c_k$—. It follows that

$$(A \,\&\, A') \rightarrow (—\alpha, c_0, \ldots, c_k— \leftrightarrow —\alpha', c_0, \ldots, c_k—)$$

is valid, and hence that

$$(4) \qquad A \,\&\, —\alpha, c_0, \ldots, c_k—$$

implies

$$(5) \qquad A' \rightarrow —\alpha', c_0, \ldots, c_k—.$$

We now apply the Craig interpolation lemma. It tells us that there is a sentence $B(c_0, \ldots, c_k)$ implied by (4) and implying (5), such that the nonlogical symbols of B are common to (4) and (5). This means that they can include only the c_i, which we have displayed, and the β_i. Since (4) implies $B(c_0, \ldots, c_k)$, A and therefore T implies

$$—\alpha, c_0, \ldots, c_k— \rightarrow B(c_0, \ldots, c_k)$$

and since the c_i do not occur in T, this means T implies

$$(6) \qquad \forall x_0 \cdots \forall x_k(—\alpha, x_0, \ldots, x_k— \rightarrow B(x_0, \ldots, x_k)).$$

Since $B(c_0, \ldots, c_k)$ implies (5), A' and therefore T' implies

$$B(c_0, \ldots, c_k) \rightarrow —\alpha', c_0, \ldots, c_k—$$

and since the c_i do not occur in T', this means T' implies

$$\forall x_0 \cdots \forall x_k(B(x_0, \ldots, x_k) \rightarrow —\alpha', x_0, \ldots, x_k—).$$

Replacing each symbol γ' by γ', it follows that T implies

$$(7) \qquad \forall x_0 \cdots \forall x_k(B(x_0, \ldots, x_k) \rightarrow —\alpha, x_0, \ldots, x_k—).$$

But (6) and (7) together imply, and therefore T implies, the explicit definition

$$\forall x_0 \cdots \forall x_k(—\alpha, x_0, \ldots, x_k— \leftrightarrow B(x_0, \ldots, x_k)).$$

Thus, α is explicitly definable from the β_i in T, and Beth's theorem is proved.

Problems

20.1 (*Lyndon's interpolation theorem*) Let A and C be sentences without constants or function symbols and in negation-normal form. We say that an occurrence of a predicate in such a sentence is *positive* if it is not preceded by ∼, and *negative* if it is preceded by $\sim$. Show that if A implies C, and neither $\sim A$ nor C is valid, then there is another such sentence B such that: (i) A implies B; (ii) B implies C; (iii) any predicate occurs positively in B only if it occurs positively in both A and C, and occurs negatively in B if and only if it occurs negatively in both A and C.

20.2 Give an example to show that Lyndon's theorem does not hold if constants are present.

20.3 (*Kant's theorem on the indefinability of chirality*). For points in the plane, we say y is *between* x and z if the three points lie on a straight line and y is between x and z on that line. We say w and x and y and z are *equidistant* if the distance from w to x and the distance from y to z are the same. We say x and y and z form a *right-handed* triple if no two distances between different pairs of them are the same, and traversing the shortest side, then the middle side, then the longest side of the triangle having them as vertices takes one around the triangle *clockwise*, as on the right in Figure 20-1.

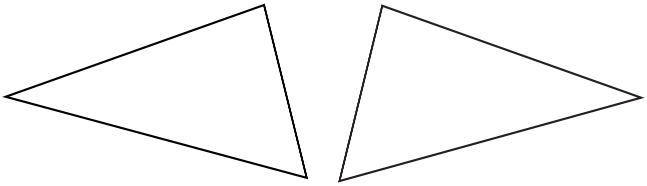


Figure 20-1. Right and left handed triangles.

Show that right-handedness cannot be defined in terms of betweenness and equidistance. (More formally, consider the language with a three-place predicate P, a four-place predicate Q, and a three-place predicate R; consider the interpretation whose domain is the set of the points in the plane and that assigns betweenness and equidistance and right-handedness as the denotations of P and Q and R; and finally consider the theory T whose theorems are all the sentences of the language that come out true under this interpretation. Show that R is not definable in terms of P and Q in this theory.)

21

Monadic and Dyadic Logic

We have given in earlier chapters several different proofs of Church's theorem to the effect that first-order logic is undecidable: there is no effective procedure that applied to any first-order sentence will in a finite amount of time tell us whether or not it is valid. This negative result leaves room on the one hand for contrasting positive results, and on the other hand for sharper negative results. The most striking of the former is the Löwenheim–Behmann theorem*, to the effect that the logic of* monadic *(one-place) predicates is decidable, even when the two-place logical predicate of identity is admitted. The most striking of the latter is the* Church–Herbrand theorem *that the logic of a single dyadic (two-place) predicate is undecidable. These theorems are presented in sections 21.2 and 21.3 after some general discussion of solvable and unsolvable cases of the decision problem for logic in section 21.1. While the proof of Church's theorem requires the use of considerable computability theory (the theory of recursive functions, or of Turing machines), that is not so for the proof of the Löwenheim–Behmann theorem or for the proof that Church's theorem* implies *the Church–Herbrand theorem. The former uses only material developed by Chapter 11. The latter uses also the elimination of function symbols and identity from section 19.4, but nothing more than this. The proofs of these two results, positive and negative, are independent of each other.*

21.1 Solvable and Unsolvable Decision Problems

Let K be some syntactically defined class of first-order sentences. By the *decision problem* for K is meant the problem of devising an effective procedure that, applied to any sentence S in K, will in a finite amount of time tell us whether or not S is valid. Since S is valid if and only if $\sim S$ is not satisfiable, and S is satisfiable if and only if $\sim S$ is not valid, for any class K that contains the negation of any sentence it contains, the decision problem for K is equivalent to the *satisfiability problem* for K, the problem of devising an effective procedure that, applied to any sentence S in K, will in a finite amount of time tell us whether or not S is satisfiable, or has a model. The formulation in terms of satisfiability turns out to be the more convenient for our purposes in this chapter.

The most basic result in this area is a negative one, Church's theorem, which asserts the unsolvability of the satisfiability problem full first-order logic, where K is the class of *all* sentences. We have given three different proofs of this result, two in Chapter 11 and another in section 17.1; but none of the machinery from any of these proofs of

Church's theorem need be recalled for purposes of this chapter. We are going to prove sharper results than Church's theorem, to the effect that the satisfiability problem is unsolvable for narrower classes K than the class of all first-order sentences; but in no case will we prove these sharper results by going back to the proof of Church's theorem and sharpening the proof. Instead, we are simply going to prove that *if* the satisfiability problem for K were solvable, *then* the satisfiability problem for full first-order logic would be solvable, as Church's theorem tells us it is not. And we are going to prove this simply by showing how one can effectively associate to any arbitrary sentence a sentence in K that is equivalent to it for satisfiability.

We have in fact already done this in one case in section 19.4, where we showed how one can effectively associate to any arbitrary sentence a sentence of *predicate logic* (that is, one not involving constants or function symbols), and indeed one of predicate logic *without identity*, that is equivalent to it for satisfiability. Thus we have already proved the following slight sharpening of Church's theorem.

21.1 Lemma. The satisfiability problem for predicate logic without identity is unsolvable.

Sharper results will be obtained by considering narrower classes of sentences: *dyadic logic*, the part of predicate logic without identity where only two-place predicates are allowed; *the logic of a triadic predicate*, where only a single three-place predicate is allowed; and finally *the logic of a dyadic predicate*, where only a single two-place predicate is allowed. Section 21.3 will be devoted to proving the following three results.

21.2 Lemma. The satisfiability problem for dyadic logic is unsolvable.

21.3 Lemma. The satisfiability problem for the logic of a triadic predicate is unsolvable.

21.4 Theorem (The Church–Herbrand theorem). The satisfiability problem for the logic of a dyadic predicate is unsolvable.

Let us now turn to positive results. Call a sentence n-satisfiable if has a model of some size $m \leq n$. Now note three things. First, we know from section 12.2 that if a sentence comes out true in some interpretation of size m, then it comes out true in some interpretation whose domain is the set of natural numbers from 1 to m. Second, for a given finite language, there are only finitely many interpretations whose domain is the set of natural numbers from 1 to m. Third, for any given one of them we can effectively determine for any sentence whether or not it comes out true in that interpretation.

[It is easy to see this last claim holds for quantifier-free sentences: the specification of the model tells us which atomic sentences are true, and then we can easily work out whether a given truth-functional compound of them is true. Perhaps the easiest way to see the claim holds for all sentences is to reduce the general case to the special case of quantifier-free sentences. To do so, for each $1 \leq k \leq m$ add to the language a constant $\mathbf{k}$ denoting k. To any sentence A of the expanded language we can effectively associate a quantifier-free sentence A^* as follows. If A is atomic, A^* is A. If A is a truth-functional compound, then A^* is the same compound of the quantifier-free sentences associated with the sentences out of which it is compounded. For instance, $(B \,\&\, C)^*$ is $B^* \,\&\, C^*$, and analogously for $\vee$. If A is $\forall x F(x)$, then A^* is $(F(\mathbf{1})\& \ldots \&F(\mathbf{m}))^*$, and analogously for $\exists$. Then A comes out true in the interpretation if and only if A^* does.]

Putting our three observations together, we have proved the following.

21.5 Lemma. For each n, the n-satisfiability problem for first-order logic is solvable.

To show that the decision problem for a class K *is* solvable, it is sufficent to show how one can effectively calculate for any sentence S in K a number n such that if S has a model at all, then it has a model of size $\leq n$. For if this can be shown, then for K the satisfiability problem is reduced to the n-satisfiability problem. The most basic positive result that can be proved in this way concerns *monadic logic*, where only one-place predicates are allowed.

21.6 Theorem. The decision problem for monadic logic is solvable.

A stronger result concerns *monadic logic with identity*, where in addition to one-place predicates, the two-place logical predicate of identity is allowed.

21.7 Theorem. The decision problem for monadic logic with identity is solvable.

These results are immediate from the following lemmas, whose proofs will occupy section 21.2.

21.8 Lemma. If a sentence involving only monadic predicates is satisfiable, then it has a model of size no greater than 2^k, where k is the number of predicates in the sentence.

21.9 Lemma. If a sentence involving only n monadic predicates and identity is satisfiable, then it has a model of size no greater than $2^k \cdot r$, where k is the number of monadic predicates and r the number of variables in the sentence.

Before launching into the proofs, some brief historical remarks may be in order. The first logician, Aristotle, was concerned with arguments such as

All horses are mammals.
All mammals are animals.
Therefore, all horses are animals.

The form of such an argument would in modern notation be represented using one-place predicates. Later logicians down through George Boole in the middle 19th century considered more complicated arguments, but still ones involving only one-place predicates. The existence had been noticed of intuitively valid arguments involving many-place predicates, such as

All horses are animals.
Therefore, all who ride horses ride animals.

But until the later 19th century, and especially the work of Gottlob Frege, logicians did not treat such arguments systematically. The extension of logic beyond the monadic to the polyadic is indispensable if the forms of arguments used in mathematical proofs are to be represented, but the ability of contemporary logic to represent the forms of such arguments comes at a price, namely, that of the undecidability of the contemporary notions of validity and satisfiability. For, as the results listed above make plain, undecidability sets in precisely when two-place predicates are allowed.

21.2 Monadic Logic

Let us get straight to work.

Proof of Lemma 21.9: Let S be a sentence of monadic logic with identity involving k one-place predicates (possibly $k = 0$) and r variables. Let $P_1, \ldots, P_k$ be predicates and $v_1, \ldots, v_r$ the variables. Suppose $\mathcal{M}$ is a model of S.

For each d in the domain $M = |\mathcal{M}|$ let the *signature* $\sigma(d)$ of d be the sequence $(j_1, \ldots, j_k)$ whose ith entry j_i is 1 or 0 according as $P_i^{\mathcal{M}}$ does or does not hold of d [if $k = 0$, then $\sigma(d)$ is the empty sequence ( )]. There are at most 2^k possible signatures. Call e and d *similar* if they have the same signature. Clearly similarity is an equivalence relation. There are at most 2^k equivalence classes.

Now let N be a subset of M containing *all* the elements of any equivalence class that has $\leq r$ elements, and exactly r elements of any equivalence class that has $\geq r$ elements. Let $\mathcal{N}$ be the subinterpretation of $\mathcal{M}$ with domain $|\mathcal{N}| = N$. Then $\mathcal{N}$ has size $\leq 2^k \cdot r$. To complete the proof, it will suffice to prove that $\mathcal{N}$ is a model of S.

Towards this end we introduce an auxiliary notion. Let $a_1, \ldots, a_s$ and $b_1, \ldots, b_s$ be sequences of elements of M. We say they *match* if for each i and j between 1 and n, a_i and b_i are similar, and $a_i = a_j$ if and only if $b_i = b_j$. We claim that if $R(u_1, \ldots, u_s)$ is a subformula of S (which implies that $s \leq r$ and that each of the us is one of the vs) and $a_1, \ldots, a_s$ and $b_1, \ldots, b_s$ are matching sequences of elements of M, with the b_i all belonging to N, then the a_i satisfy R in $\mathcal{M}$ if and only if the b_i satisfy R in $\mathcal{N}$. To complete the proof it will suffice to prove this claim, since, applied with $s = 0$, it tells us that since S is true in $\mathcal{M}$, S is true in $\mathcal{N}$, as desired.

The proof of the claim is by induction on complexity. If R is atomic, it is either of the form $P_j(u_i)$ or of the form $u_i = u_j$. In the former case, the claim is true because matching requires that a_i and b_i have the same signature, so that $P_j^{\mathcal{M}}$ holds of the one if and only if it holds of the other. In the latter case, the claim is true because matching requires that $a_i = a_j$ if and only if $b_i = b_j$.

If R is of form $\sim Q$, then the as satisfy R in $\mathcal{M}$ if and only if they do not satisfy Q, and by the induction hypothesis the as fail to satisfy Q in $\mathcal{M}$ if and only if the bs fail to satisfy Q in $\mathcal{N}$, which is the case if and only if the bs satisfy R in $\mathcal{N}$, and we are done. Similarly for other truth-functional compounds.

It remains to treat the case of universal quantification (and of existential quantification, but that is similar and is left to the reader). So let R $(u_1, \ldots, u_s)$ be of form $\forall u_{s+1} Q(u_1, \ldots, u_s, u_{s+1})$, where $s + 1 \leq r$ and each of the us is one of the vs. We need to show that $a_1, \ldots, a_s$ satisfy R in $\mathcal{M}$ (which is to say that for any a_{s+1} in M, the longer sequence of elements $a_1, \ldots, a_s, a_{s+1}$ satisfies Q in $\mathcal{M}$) if and only if $b_1, \ldots, b_s$ satisfy R in $\mathcal{N}$ (which is to say that for all b_{s+1} in N, the longer sequence of elements $b_1, \ldots, b_s, b_{s+1}$ satisfies Q in $\mathcal{N}$). We treat the 'if' direction and leave the 'only if' direction to the reader.

Our induction hypothesis is that if $a_1, \ldots, a_s, a_{s+1}$ and $b_1, \ldots, b_s, b_{s+1}$ match, then $a_1, \ldots, a_s, a_{s+1}$ satisfy Q in $\mathcal{M}$ if and only if $b_1, \ldots, b_s, b_{s+1}$ satisfy Q in $\mathcal{N}$. What we want to show is that if $b_1, \ldots, b_s, b_{s+1}$ satisfy Q in $\mathcal{N}$ for all b_{s+1} in N, then $a_1, \ldots, a_s, a_{s+1}$ satisfy Q in $\mathcal{M}$ for all a_{s+1} in M. Therefore it will be enough to show that if $a_1, \ldots, a_s$ and $b_1, \ldots, b_s$ match, where $s < r$, then for any a_{s+1} in M there is a b_{s+1} in N such that $a_1, \ldots, a_s, a_{s+1}$ and $b_1, \ldots, b_s, b_{s+1}$ match.

In the degenerate case where a_{s+1} is identical with one of the previous a_i, we may simply take b_{s+1} to be identical with the corresponding b_i. In the non-degenerate case, a_{s+1} belongs to some equivalence class C and is distinct from any and all previous a_i that belong to C. Let the number of such a_i be t (where possibly $t = 0$), so that there are at least $t + 1$ elements in C, counting a_{s+1}. To ensure matching, it will suffice to choose b_{s+1} to be some element of C that is distinct from any and all previous b_i that belong to C. Since $a_1, \ldots, a_s$ and $b_1, \ldots, b_s$ match, the number of such b_i will also be t. Since $t \leq s < r$, and there are at least $t + 1 \leq r$ elements in C, there will be at least that many elements of C in N, and so we can find an appropriate b_{s+1}, to complete the proof.

21.10 Corollary. If a sentence involving no nonlogical symbols (but only identity) is satisfiable, then it has a model of size no greater than r, where r is the number of variables in the sentence.

21.11 Corollary. If a sentence of monadic logic involving only one variable is satisfiable, then it has a model of size no greater than 2^k, where k is the number of monadic predicates in the sentence.

Proofs: These are simply the cases $k = 0$ and $r = 1$ of Lemma 21.9.

Proof of Lemma 21.7: This is immediate from Corollary 21.11 and the following, which is a kind of normal form theorem.

21.12 Lemma. Any sentence of monadic logic without identity is logically equivalent to one with the same predicates and only one variable.

Proof: Call a formula *clear* if in any subformula $\forall x B(x)$ or $\exists x B(x)$ that begins with a quantifier, no variable other than the variable x attached to the quantifier appears in F. Thus $\forall x \exists y(Fx \mathbin{\&} Gy)$ is not clear, but $\forall x Fx \mathbin{\&} \exists\, yGy$ is clear. To prove the lemma, we show how one can inductively associate to any formula A of monadic logic without identity an equivalent formula $A^{©}$ with the same predicates that is clear (as in our example the first formula is equivalent to the second). We then note that any clear sentence is equivalent to the result of rewriting all its variables to be the same (as in our example the second sentence is equivalent to $\forall z Fz \mathbin{\&} \exists z Gz$). The presence of identity would make such clearing impossible. (There is no clear sentence equivalent to $\forall x \exists y\, x \neq y$, for instance.)

To an atomic formula we associate itself. To a truth-functional compound of formulas to which clear equivalents have been associated, we associate the same truth-functional compound of those equivalents. Thus $(B \vee C)^{©}$ is $B^{©} \vee C^{©}$, for instance, and analogously for &. The only problem is how to define the associate $(\exists x B(x))^{©}$

to a quantified formula $\exists x B(x)$ in terms of the associate $(B(x))^{©}$ of the subformula $B(x)$, and analogously for $\forall$.

$\exists x B(x)$ will of course be equivalent to $\exists x(B(x))^{©}$. And $(B(x))^{©}$ will be a truth-functional compound of clear formulas $A_1, \ldots, A_n$, each of which either is atomic or begins with a quantifier. Consider a formula equivalent to $(B(x))^{©}$ that is in disjunctive normal form in the A_i. It will be a disjunction $B_1 \vee \cdots \vee B_r$ of formulas B_j, each of which is a conjunction of some of the A_i and their negations. We may assume each B_j has the form

$$C_{j,1} \,\&\, \ldots \,\&\, C_{j,r} \,\&\, D_{j,1} \,\&\, \ldots \,\&\, D_{j,s}$$

where the Cs are the conjuncts in which the variable x does occur and the Ds those in which it does not; by clarity, the Ds will include all conjuncts that begin with or are the negations of formulas beginning with quantifiers, and the Cs will all be atomic. Then as $\exists x(B(x))^{©}$ we may take the disjunction $B'_1 \vee \cdots \vee B'_r$, where B'_j is

$$\exists x(C_{j,1} \,\&\, \cdots \,\&\, C_{j,r}) \,\&\, D_{j,1} \,\&\, \ldots \,\&\, D_{j,s}.$$

(In the degenerate case where $r = 0$, B'_j is thus the same as B_j.)

21.3 Dyadic Logic

Again we go straight to work.

Proof of Lemma 21.2: Lemma 21.1 tells us the satisfiability problem is unsolvable for predicate logic, and we want to show it is unsolvable for dyadic logic. It will be enough to show how one can effectively associate to any sentence of predicate logic a sentence of dyadic logic such that the former will be satisfiable if and only if the latter is. What we are going to do is to show how to eliminate one three-place predicate (at the cost of introducing new two- and one-place predicates). The same method will work for k-place predicates for any $k \geq 3$, and applying it over and over we can eliminate all but two- and one-place predicates. The one-place ones can also be eliminated one at a time, since given a sentence S containing a one-place predicate P, introducing a new two-place predicate P^* and replacing each atomic subformula Px by P^*xx clearly produces a sentence S^* that is satisfiable if and only if S is. Thus we can eliminate all but two-place predicates.

To indicate the method for eliminating a three-place predicate, let S be a sentence containing such a predicate P. Let P^* be a new one-place predicate, and Q_i for $i = 1, 2, 3$ a trio of new two-place predicates. Let w be a variable not appearing in S, and let S^* be the result of replacing each atomic subformula of form $Px_1x_2x_3$ in S by

$$\exists w(Q_1wx_1 \,\&\, Q_2wx_2 \,\&\, Q_3wx_3 \,\&\, P^*w).$$

We claim S is satisfiable if and only if S^* is satisfiable. The 'if' direction is easy. For if S is unsatisfiable, then $\sim S$ is valid, and substitution (of a formula with the appropriate free variables for a predicate) preserves validity, so $\sim S^*$ is valid, and S^* is unsatisfiable.

For the 'only if' direction, suppose S has a model $\mathcal{M}$. By the canonical domains theorem (Corollary 12.18) we may take the domain of $\mathcal{M}$ be the set of natural numbers. We want to show that S^* has a model $\mathcal{M}^*$. We will take $\mathcal{M}^*$ to have domain the set of natural numbers, and to assign to every predicate in S other than P the same denotation that $\mathcal{M}$ assigns. It will suffice to show that we can assign denotations to P^* and the Q_i in such a way that natural numbers a_1, a_2, a_3 will satisfy $\exists w(Q_1wx_1$ & Q_2wx_2 & Q_3wx_3 & $P^*w)$ in $\mathcal{M}^*$ if and only if they satisfy $Px_1x_2x_3$ in $\mathcal{M}$. To achieve this, fix a function f from the natural numbers onto the set of all triples of natural numbers. It then suffices to take as the denotation of P^* in $\mathcal{M}^*$ the relation that holds of a number b if and only if $f(b)$ is a triple a_1, a_2, a_3 for which the relation that is the denotation of P in $\mathcal{M}$ holds, and to take as the denotation of Q_i in $\mathcal{M}^*$ the relation that holds of b and a if and only if a is the ith component of the triple $f(b)$.

Proof of Lemma 21.3: We want next to show that we can eliminate any number of two-place predicates $P_1, \ldots, P_k$ in favour of a single three-place predicate Q. So given a sentence S containing the P_i, let $u_1, \ldots, u_k$ be variables not occurring in S, and let S^* be the result of replacing each atomic subformula of form $P_ix_1x_2$ in S by $Qv_ix_1x_2$, and let $S^\dagger$ be the result of prefixing S^* by $\exists v_1 \cdots \exists v_k$. For instance, if S is

$$\forall x \exists y(P_2yx \mathbin{\&} \forall z(P_1xz \mathbin{\&} P_3zy))$$

then $S^\dagger$ will be

$$\exists v_1 \exists v_2 \exists v_3 \forall x \exists y(Qv_2yx \mathbin{\&} \forall z(Qv_1xz \mathbin{\&} Qv_3zy)).$$

We claim S is satisfiable if and only if $S^\dagger$ is satisfiable. As in the preceding proof, the 'if' direction is easy, using the fact that substitution preserves validity. (More explicitly, if there is a model $\mathcal{M}^\dagger$ of $S^\dagger$, some elements $a_1, \ldots, a_k$ of its domain satisfy the formula S^*. We can now get a model $\mathcal{M}$ of S by taking the same domain, and assigning as denotation to P_i in $\mathcal{M}$ the relation that holds between b_1 and b_2 if and only if the relation that is the denotation of Q in $\mathcal{M}^\dagger$ holds among a_i and b_1 and b_2.)

For the 'only if' direction, suppose $\mathcal{M}$ is a model of S. As in the preceding proof, we may take the domain of $\mathcal{M}$ to be the set of natural numbers, using the canonical domains theorem. We can now get a model $\mathcal{M}^\dagger$ of $S^\dagger$, also with domain the natural numbers, by taking as the denotation of Q in $\mathcal{M}^\dagger$ the relation that holds among natural numbers a and b_1 and b_2 if and only if $1 \leq a \leq k$, and as the denotation of P_a in $\mathcal{M}$ the relation that holds between b_1 and b_2. From the fact that $\mathcal{M}$ is a model of S, it follows that $1, \ldots, k$ satisfy S^* in $\mathcal{M}^\dagger$, and hence $S^\dagger$ is true in $\mathcal{M}^\dagger$.

Proof of Theorem 21.4: We want next to show that we can eliminate a single three-place predicate P in favour of a single two-place predicate Q. So given a sentence S containing the P, let u_1, u_2, u_3, u_4 be variables not occurring in S, and let S^* be the result of replacing each atomic subformula of form $P_ix_1x_2x_3$ in S by a certain formula $P^*(x_1, x_2, x_3)$, namely

$$\begin{aligned}&\exists u_1 \exists u_2 \exists u_3 \exists u_4 (\sim Qu_1u_1 \mathbin{\&} Qu_1u_2 \mathbin{\&} Qu_2u_3 \mathbin{\&} Qu_3u_4\\ &\quad \mathbin{\&} Qu_4u_1 \mathbin{\&} Qu_1x_1 \mathbin{\&} Qu_2x_2 \mathbin{\&} Qu_3x_3 \mathbin{\&} \sim Qx_1u_2 \mathbin{\&} \sim Qx_2u_3 \mathbin{\&} \sim Qx_3u_4 \mathbin{\&} Qu_4x_1).\end{aligned}$$

We then claim S is satisfiable if and only if S^* is satisfiable. As in the preceding proofs, the 'if' direction is easy, and for the 'only if' direction what we need to do is to show, given a model $\mathcal{M}$ of S, which may be taken to have domain the natural numbers, that we can define an interpretation $\mathcal{M}^*$, also with domain the natural numbers, which will assign as denotation to Q in $\mathcal{M}^*$ a relation such that for any natural numbers b_1, b_2, b_3, those numbers will satisfy $P^*(x_1, x_2, x_3)$ in $\mathcal{M}^*$ if and only if those numbers satisfy $P(x_1, x_2, x_3)$ in $\mathcal{M}$. To accomplish this last and so complete the proof, it will be enough to establish the following lemma.

21.13 Lemma. Let R be a three-place relation on the natural numbers. Then there is a two-place relation S on the natural numbers such that if a, b, c are any natural numbers, then we have $Rabc$ if and only if for some natural numbers w, x, y, z we have

$$\begin{gathered}\sim Sww \,\&\, Swx \,\&\, Sxy \,\&\, Syz \,\&\, Szw \,\&\\ Swa \,\&\, Sxb \,\&\, Syc \,\&\, {\sim} Sax \,\&\, {\sim} Sby \,\&\, {\sim} Scz \,\&\, Sza.\end{gathered} \tag{1}$$

Proof: One of the several ways of enumerating all triples of natural numbers is to order them by their sums, and where these are the same by their first components, and where these also are the same by their second components, and where these also are the same by their third components. Thus the first few triples are

$$\begin{gathered}(0, 0, 0)\\(0, 0, 1)\\(0, 1, 0)\\(1, 0, 0)\\(0, 0, 2)\\(0, 1, 1)\\(0, 2, 0)\\(1, 0, 1)\\(1, 1, 0)\\(2, 0, 0)\\(0, 0, 3)\\\vdots\end{gathered}$$

Counting the initial triple as the first rather than the zeroth, it is clear that if the nth triple is (a, b, c), then a, b, c are all $<n$. It follows that if w, x, y, and z are respectively $4n + 1, 4n + 2, 4n + 3$, and $4n + 4$, then a, b, c are all less than $w - 4, x - 4, y - 4$, and $z - 4$. (For instance, $a < n$ implies $a + 1 \leq n$, which implies $4a + 4 \leq 4n < 4n + 1$.)

Now to define S. If the nth triple is (a, b, c), we let Svu hold in each of the following four cases:

$$\begin{array}{lll} v = 4n+1 & \text{and} & (u = 4n+2 \text{ or } u = a)\\ v = 4n+2 & \text{and} & (u = 4n+2 \text{ or } u = 4n+3 \text{ or } u = b)\\ v = 4n+3 & \text{and} & (u = 4n+3 \text{ or } u = 4n+4 \text{ or } u = c)\\ v = 4n+4 & \text{and} & (u = 4n+4 \text{ or } u = 4n+1 \text{ or } (u = a \text{ and } Rabc)).\end{array}$$

Svu is not to hold in any other cases. Note that

(2) if Svu, then $v + 1 \geq u$

(3) there is at most one $u < v - 4$ such that Svu.

We must now show that *Rabc* holds if and only if there are w, x, y, z such that (1) above holds. The 'only if' direction is immediate: if *Rabc*, take w, x, y, z to be $4n + 1, 4n + 2, 4n + 3, 4n + 4$, where (a, b, c) is the nth triple, and (1) will hold. [$\sim$*Sax* holds because $a < x - 4$ holds, so $a + 1 \geq x$ fails, so *Sax* fails by (2); similarly for the other negations in (1).]

Now suppose (1) holds for some w, x, y, z. We must show that *Rabc*. To begin with, (1) gives us $\sim Sww$, so w must be of the form $4n + 1$ for some $n \geq 1$.

Also, (1) gives us Swx, Sxy, Syz, Szw. Therefore, by (2), $x + 3 \geq y + 2 \geq z + 1 \geq w$, whence $x \geq w - 3$. Similarly, $y \geq x - 3$ and $z \geq y - 3$. So neither $x < w - 4$ nor $y < x - 4$ nor $z < y - 4$. Since Swx holds while $x < w - 4$ fails, we must have $x = w + 1 = 4n + 2$.

Also, since *Sxy* holds while $y < x - 4$ fails, either $y = x$ or $y = x + 1$. Similarly, either $z = y$ or $z = y + 1$. But if either $y = x$ or $z = y$, then $z = w + 1 = 4n + 2$ or $z = w + 2 = 4n + 3$. But this is impossible, since Svu never holds for $u = 4m + 1$ and $v = 4n + 2$ or $4n + 3$, whereas we have Szw. It follows that $y = x + 1 = 4n + 3$ and $z = y + 1 = 4n + 4$.

If we can show that the nth triple is (a, b, c), then we can conclude that *Rabc*: for if (a, b, c) is the nth triple, then *Sza* if and only if *Rabc*, and (1) gives *Sza*.

We have Swa, Swb, Syc and $\sim$*Sax*, $\sim$*Sby*, $\sim$*Scz* from (1). And since we know $w = 4n + 1, x = 4n + 2, y = 4n + 3, z = 4n + 4$, we also have *Sxx*, *Sxy*, *Syy*, *Syz*, and *Szz* from the definition of S. So $a \neq x, b \neq x, b \neq y, c \neq y$, and $c \neq z$. So we have *Swa* and $a < w - 4$, *Sxb* and $b < x - 4$, and *Syc* and $c < y - 4$. If the nth triple is (r, s, t), then we also have Swr, *Sxs*, *Syt* and $r < w - 4, s < x - 4, t < y - 4$. So by (3) above we must have $r = a, s = b$, and $t = c$. So (a, b, c) is the nth triple, and the proof is complete.

Problems

21.1 Prove Lemma 21.8 directly, without deriving it from Lemma 21.9.

21.2 Show that the estimates 2^k and $2^k \cdot r$ in Lemmas 21.8 and 21.9 cannot be improved.

21.3 What happens if constants are added to monadic logic with identity?

21.4 The language of set theory has a single nonlogical symbol and two-place predicate $\in$. **ZFC** is a certain theory in this language, of which it was asserted towards the end of section 17.1 that it is 'adequate for formalizing essentially all accepted mathematical proofs'. What is the bearing of this fact on Theorem 21.4?

22

Second-Order Logic

Suppose that, in addition to allowing quantifications over the elements of a domain, as in ordinary first-order logic, we allow also quantification over relations and functions on the domain. The result is called second-order logic. *Almost all the major theorems we have established for first-order logic fail spectacularly for second-order logic, as is shown in the present short chapter. This chapter and those to follow generally presuppose the material in section 17.1. (They are also generally independent of each other, and the results of the present chapter will not be presupposed by later ones.)*

Let us begin by recalling some of the major results we have established for first-order logic.

The compactness theorem: If every finite subset of a set of sentences has a model, the whole set has a model.

The (downward) Löwenheim–Skolem theorem: If a set of sentences has a model, it has an enumerable model.

The upward Löwenheim–Skolem theorem: If a set of sentences has an infinite model, it has a nonenumerable model.

The (abstract) Gödel completeness theorem: The set of valid sentences is semirecursive.

All of these results fail for *second-order logic*, which involves an extended notion of sentence, with a corresponding extension of the notion of truth of a sentence in an interpretation. In introducing these extended notions, we stress at the outset that we change neither the definition of *language* nor the definition of *interpretation*: a language is still an enumerable set of nonlogical symbols, and an interpretation of a language is still a domain together with an assignment of a denotation to each nonlogical symbol in the language. The only changes will be that we add some new clauses to the definition of what it is to be a sentence of a language, and correspondingly some new clauses to the definition of what it is for a sentence of a language to be true in an interpretation.

What is a second-order sentence? Let us refer to what we have been calling 'variables' as *individual* variables. We now introduce some new kinds of variable: *relation* variables and *function* variables. Just as we have one-, two-, three-, and more-place predicates or relation symbols and function symbols, we have one-, two-, three-, and more-place relation variables and function variables. (Since one-place relations are just sets, one-place relation variables may be called *set* variables.) We suppose that no symbol of any sort is also a symbol of any other sort. We extend the definition of formula by allowing relation or function variables to occur in those positions in formulas where previously only relation symbols (a.k.a. predicates) or function symbols (respectively!) could occur, and also by allowing the new kinds of variable to occur after $\forall$ and $\exists$ in quantifications. Free and bound occurrences are defined for the new kinds of variable exactly as they were for defined for individual variables. Sentences, as always, are formulas in which no variables (individual, relation, or function) occur free. A second-order formula, then, is a formula that contains at least one occurrence of a relation or function variable, and a second-order sentence is a second-order formula that is a sentence. A formula or sentence of a language, whether first- or second-order, is, as before, one whose nonlogical symbols all belong to the language.

22.1 Example (Second-order sentences). (In the following examples we use u as a one-place function variable, and X as a one-place relation variable.)

In first-order logic we could identify a particular function as the identity function: $\forall x\, f(x) = x$. But in second-order logic we can assert the existence of the identity function: $\exists u\, \forall x\, u(x) = x$.

Similarly, where in first-order logic we could assert that two particular indviduals share a property (Pc & Pd), in second-order logic we can assert that every two individuals share some property or other: $\forall x \forall y \exists X(Xx \,\&\, Xy)$.

Finally, in first-order logic we can assert that if two particular individuals are identical, then they must either both have or both lack a particular property: $c = d \rightarrow (Pc \leftrightarrow Pd)$. But in second-order logic we can *define* identity through *Leibniz's law* of the *identity of indiscernibles:* $c = d \leftrightarrow \forall X\, (Xc \leftrightarrow Xd)$.

Each of the three second-order sentences above is valid: true in each of its interpretations.

When is a second-order sentence S true in an interpretation $\mathcal{M}$? We answer this question by adding four more clauses (for universal and existential quantifications involving relation and function variables) to the definition of truth in an interpretation given in section 9.3. For a universal quantification $\forall XF(X)$ involving a relation variable, the clause reads as follows. First we define what it is for a relation R (of the appropriate number of places) on the domain of $\mathcal{M}$ to *satisfy* $F(X)$: R does so if, on expanding the language by adding a new relation symbol P (of the appropriate number of places) to the language, and expanding the interpretation $\mathcal{M}$ to an interpretation $\mathcal{M}_R^P$ of the expanded language by taking R as the denotation of P, the sentence $F(P)$ becomes true. Then we define $\forall XF(X)$ to be true in $\mathcal{M}$ if and only if every relation R (of the appropriate number of places) on the domain of $\mathcal{M}$ satisfies $F(X)$. The clauses for existential quantifications and for function symbols are similar. The definitions of validity, satisfiability, and implication are also unchanged for second-order sentences. Any sentence, first- or second-order, is valid if and only if true in all its interpretations, and satisfiable if and only if true in at least one of them. A set Γ of sentences implies a sentence D if and only if there is no interpretation in which all the sentences in Γ are true but D false.

(The foregoing gives the *standard* notion of interpretation and truth for second-order logic. In the literature *nonstandard* notions, euphemistically called 'general', are sometimes considered, where an interpretation has separate domains of individuals and of relations and functions. These will not be considered here.)

22.2 Example (The definition of identity). The Leibniz definition of identity in Example 22.1 is unnecessarily complicated, since the following simpler *Whitehead–Russell* definition will do:

$$c = d \leftrightarrow \forall X(Xc \rightarrow Xd)$$

We don't *need* a biconditional on the right!

Proof: $\sim Pc \vee Pd$ or $Pc \rightarrow Pd$ is true in an interpretation just in case the set P denotes either fails to contain the individual c denotes or contains the one d denotes. Hence a set R satisfies $Xc \rightarrow Xd$ just in case it either fails to contain the individual c denotes or contains the one d denotes. Hence $\forall X(Xc \rightarrow Xd)$ is true just in case every set either fails to contain the individual c denotes or contains the one d denotes. If c and d denote the same individual, this must be so for every set, while if c and d do not denote the same individual, then it will fail to be so for the set whose one and only element is the individual c denotes. Thus $\forall X(Xc \rightarrow Xd)$ is true just in case c and d denote the same individual, which is to say, if and only if $c = d$ is true. (Intuitively, the Whitehead–Russell definition is valid because among the properties of a is the property of *being identical with a*; hence if the individual b is to have *all* the properties of a, it must in particular have the property of being identical with a.)

22.3 Example (The 'axiom' of enumerability). Let Enum be the sentence

$$\exists z \exists u \forall X((Xz \,\&\, \forall x(Xx \rightarrow Xu(x))) \rightarrow \forall x Xx).$$

Then Enum is true in an interpretation if and only if its domain is enumerable.

Proof: First suppose Enum is true in an interpretation $\mathcal{M}$. This means there exists an individual a in $|\mathcal{M}|$ and a one-place function f on $|\mathcal{M}|$ that satisfy

$$\forall X((Xz \,\&\, \forall x(Xx \rightarrow Xu(x))) \rightarrow \forall x Xx).$$

Thus, if we add a constant $\mathbf{0}$ and let it denote a, and a one-place function symbol $'$ and let it denote f, then

$$\forall X((X\mathbf{0} \,\&\, \forall x(Xx \rightarrow Xx')) \rightarrow \forall x Xx)$$

is true. This means every subset A of $|\mathcal{M}|$ satisfies

$$(X\mathbf{0} \,\&\, \forall x(Xx \rightarrow Xx')) \rightarrow \forall x Xx.$$

In particular this is so for the enumerable subset A of $|\mathcal{M}|$ whose elements are all and only a, $f(a)$, $f(f(a))$, $f(f(f(a)))$, and so on. Thus if we add a one-place predicate $\mathbf{N}$ and let it denote A, then

$$(\mathbf{N0} \,\&\, \forall x(\mathbf{N}x \rightarrow \mathbf{N}x')) \rightarrow \forall x \mathbf{N}x$$

is true. But $\mathbf{N0}$ is true, since the individual a that is the denotation of $\mathbf{0}$ is in the set A that is the denotation of $\mathbf{N}$, and $\forall x(\mathbf{N}x \to \mathbf{N}x')$ is true, since if any individual is in A, so is the value obtained when the function f that is the denotation of $'$ is applied to that individual as argument. Hence $\forall x\mathbf{N}x$ must be true, and this means that *every* individual in the domain is in A, so the domain, being just A, is enumerable.

Conversely, suppose that the domain of an interpretation $\mathcal{M}$ is enumerable. Fix an enumeration of its elements: m_0, m_1, m_2, and so on. Let a be m_0, and let f be the function that given m_i as argument yields m_{i+1} as value, and add a constant $\mathbf{0}$ and a one-place function symbol $'$ to denote a and f. Given any subset A of the domain, suppose we add a one-place predicate $\mathbf{N}$ to denote A. Then if $\mathbf{N0}$ is true, $a = m_0$ must belong to A, and if $\forall x(\mathbf{N}x \to \mathbf{N}x')$ is true, then whenever m_i belongs to A, $f(m_i) = m_{i+1}$ must belong to A. So if both are true, every element $m_0, m_1, m_2, \ldots$ of the domain must belong to A, and therefore $\forall x\mathbf{N}x$ is true. Thus

$$(\mathbf{N0}\,\&\,\forall x(\mathbf{N}x \to \mathbf{N}x')) \to \forall x\mathbf{N}x$$

is true if $\mathbf{N}$ is taken to denote A, and therefore A satisfies

$$(X\mathbf{0}\,\&\,\forall x(Xx \to Xx')) \to \forall xXx$$

and since this is true for any A,

$$\forall X((X\mathbf{0}\,\&\,\forall x(Xx \to Xx')) \to \forall xXx)$$

is true, and therefore

$$\exists z\exists u\forall X((Xz\,\&\,\forall x(Xx \to Xu(x))) \to \forall xXx)$$

or Enum is true in $\mathcal{M}$.

22.4 Example (The ‘axiom’ of infinity). Let Inf be the sentence

$$\exists z\exists u(\forall xz \neq u(x)\,\&\,\forall x\forall y(u(x) = u(y) \to x = y)).$$

Then Inf is true in an interpretation if and only if its domain is infinite. The proof is left to the reader.

22.5 Proposition. The downward and upward Löwenheim–Skolem theorems both fail for second-order logic.

Proof: Inf & ∼Enum and Inf & Enum are both second-order sentences having infinite but no finite models. The former has only nonenumerable models, contrary to the downward Löwenheim–Skolem theorem; the latter only denumerable models, contrary to the upward Löwenheim–Skolem theorem.

It is an immediate consequence of the downward and upward Löwenheim–Skolem theorems that if a first-order sentence or set of such sentences has an infinite model, then it has nonisomorphic infinite models. Even this corollary of the Löwenheim–Skolem theorems fails for second-order logic, as the next example shows.

22.6 Example (Second-order arithmetic). Let $\mathbf{P}^{\mathrm{II}}$ be the conjunction of the axioms of $\mathbf{Q}$ (as in section 16.2) with the following sentence Ind, called the *axiom of induction*:

$$\forall X((X\mathbf{0} \,\&\, \forall x(Xx \to Xx')) \to \forall x Xx).$$

Then an interpretation of the language of arithmetic is a model of $\mathbf{P}^{\mathrm{II}}$ if and only if it is isomorphic to the standard interpretation.

Proof: We have already in effect seen in the proof of Example 22.3 that in any model of Ind, the domain will consist precisely of the denotations of the terms $\mathbf{0}, \mathbf{0}', \mathbf{0}'', \ldots$, which is to say, of the numerals $\mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots$, as we usually abbreviate those terms. We have also seen in section 16.2 that in any model of the axioms of $\mathbf{Q}$, all the following will be true for natural numbers m, n, and p:

$$\begin{array}{rll}
\mathbf{m} \neq \mathbf{n} & \text{if} & m \neq n \\
\mathbf{m} < \mathbf{n} & \text{if} & m < n \\
\sim\mathbf{m} < \mathbf{n} & \text{if} & m \geq n \\
\mathbf{m} + \mathbf{n} = \mathbf{p} & \text{if} & m + n = p \\
\mathbf{m} + \mathbf{n} \neq \mathbf{p} & \text{if} & m + n \neq p \\
\mathbf{m} \cdot \mathbf{n} = \mathbf{p} & \text{if} & m \cdot n = p \\
\mathbf{m} \cdot \mathbf{n} \neq \mathbf{p} & \text{if} & m \cdot n \neq p.
\end{array}$$

Now let $\mathcal{M}$ be a model of $\mathbf{P}^{\mathrm{II}}$. Every element of $|\mathcal{M}|$ is the denotation of at least one $\mathbf{m}$, because $\mathcal{M}$ is a model of Ind, and of at most one $\mathbf{m}$, because $\mathcal{M}$ is a model of the axioms of $\mathbf{Q}$ and therefore of $\mathbf{m} \neq \mathbf{n}$ whenever $m \neq n$, by the first fact on the list above. We can therefore define a function j from $|\mathcal{M}|$ to the natural numbers by letting the value of j for the argument that is the denotation of $\mathbf{m}$ by m. By the other six facts on the list above, j will be an isomorphism between $\mathcal{M}$ and the standard interpretation.

Conversely, $\mathbf{P}^{\mathrm{II}}$ is easily seen to be true in the standard interpretation, and the proof of the isomorphism theorem (Proposition 12.5) goes through essentially unchanged for second-order logic, so any interpretation isomorphic to the standard interpretation will also be an model of $\mathbf{P}^{\mathrm{II}}$.

22.7 Proposition. The compactness theorem fails for second-order logic.

Proof: As in the construction of a nonstandard model of first-order arithmetic, add a constant c to the language of arithmetic and consider the set

$$\Gamma = \{\mathbf{P}^{\mathrm{II}}, c \neq \mathbf{0}, c \neq \mathbf{1}, c \neq \mathbf{2}, \ldots\}.$$

Every finite subset Γ_0 has a model obtained by expanding the standard interpretation to assign a suitable denotation to c—any number bigger than all those mentioned in Γ_0 will do. But Γ itself does not, because in any model of $\mathbf{P}^{\mathrm{II}}$ every element is the denotation of one of the terms $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$, and so on.

22.8 Proposition. The (abstract) Gödel completeness theorem fails for second-order logic: The set of valid sentences of second-order logic is not semirecursive (or even arithmetical).

Proof: A first-order sentence A of the language of arithmetic is true in the standard interpretation if and only if it is true in all interpretations isomorphic to the standard one, and hence by the preceding example if and only if it is true in all models of $\mathbf{P}^{\mathrm{II}}$, or equivalently, if and only if $\mathbf{P}^{\mathrm{II}} \rightarrow A$ is valid. The function taking (the code number of) a first-order sentence A to (the code number of) the second-order sentence $\mathbf{P}^{\mathrm{II}} \rightarrow A$ is clearly recursive. (Compare the proof of Theorem 17.6.) Hence if the set of (code numbers of) valid second-order sentences were semirecursive, the set of (code numbers of) sentences of the language of arithmetic true in the standard interpretation would be also. But the latter set is not arithmetical (by Theorem 17.3) and *a fortiori* not semirecursive.

Proposition 22.8 is sometimes formulated as follows: ‘Second-order logic is incomplete’. A more accurate formulation would be: ‘No sound proof procedure for second-order logic is complete’. (After all, it’s not the logic that’s incomplete, but candidate proof procedures.)

We conclude this chapter with a preview of the next. Recall that a set S of natural numbers is *arithmetically definable*, or simply *arithmetical*, if there is a first-order formula $F(x)$ of the language of arithmetic such that S consists of just those m for which $F(\mathbf{m})$ is true in the standard interpretation, or equivalently, just those m that satisfy $F(x)$ in the standard interpretation. A set S of natural numbers is *analytically definable* or *analytical* if there is a first- or second-order formula $\phi(x)$ of the language of arithmetic such that S consists of just those m that satisfy $\phi(x)$ in the standard interpretation. Let us, for the space of this discussion, use the word *class* for sets of sets of natural numbers. Then a *class* Σ of sets of natural numbers is *arithmetical* if there is a second-order formula $F(X)$ *with no bound relation or function variables* such that Σ consists of just those sets M that satisfy $F(X)$ in the standard intepretation. A class Σ of sets of natural numbers is *analytical* if there is a second-order formula $\phi(X)$ such that Σ consists of just those sets M that satisfy $\phi(X)$ in the standard interpretation. We have seen that recursive and semirecursive sets are arithmetical, but that the set of (code numbers of) first-order sentences of the language of arithmetic that are true in the standard interpretation is not arithmetical. It can similarly be shown that the set of first- and second-order sentences true in the standard interpretation is not analytical.

However, the set V of (code numbers of) *first*-order sentences true in the standard interpretation *is* analytical. This follows from the fact, to be proved in the next chapter, that the class $\{V\}$ of sets of natural numbers whose one and only member is the set V is arithmetical. The latter result means that there is a second-order formula $F(X)$ with no bound relation or function variables such that V is the one and only set that satisfies $F(X)$ in the standard interpretation. From this it follows that V is precisely the set of m that satisfy $\exists X(F(X) \,\&\, Xx)$; and this shows that, as asserted, V is analytical. It will also be shown that the class of arithmetical sets of natural numbers is not arithmetical. (Again, this class can be shown to be analytical.) In order to keep the next chapter self-contained and independent of this one, a different definition of arithmetical class will be given there, not presupposing familiarity with second-order logic. However, the reader who *is* familiar with second-order logic should have no difficulty recognizing that this definition is equivalent to the one given here.

Problems

22.1 Does it follow from the fact that $\exists xFx \,\&\, \exists x{\sim}Fx$ is satisfiable that $\exists X(\exists xXx \,\&\, \exists x{\sim}Xx)$ is valid?

22.2 Let us write R^*ab to abbreviate

$$\forall X[Xa \,\&\, \forall x\forall y((Xx \,\&\, Rxy) \to Xy) \to Xb].$$

Show that the following are valid:

(a) R^*aa

(b) $Rab \to R^*ab$

(c) $(R^*ab \,\&\, R^*bc) \to R^*ac$

Suppose Rab if and only if a is a child of b. Under what conditions do we have R^*ab?

22.3 (*A theorem of Frege*) Show that (a) and (b) imply (c):

(a) $\forall x\forall y\forall z[(Rxy \,\&\, Rxz) \to y = z]$

(b) $\exists x(R^*xa \,\&\, R^*xb)$

(c) $(R^*ab \vee a = b \vee R^*ba)$.

22.4 Write $\Diamond(R)$ to abbreviate

$$\forall x\forall y(\exists w(Rwx \,\&\, Rwy) \to \exists z(Rxz \,\&\, Ryz)).$$

Show that $\Diamond(R) \to \Diamond(R^*)$ is valid.

22.5 (*The principle of Russell's paradox*) Show that $\exists X{\sim}\exists y\forall x(Xx \leftrightarrow Rxy)$ is valid.

22.6 (*A problem of Henkin*) Let Q1 and Q2 be as in section 16.2, and let I be the induction axiom of Example 22.6. Which of the eight combinations $\{(\sim)\text{Q1}, (\sim)\text{Q2}, (\sim)I\}$, where on each of the three sentences the negation sign may be present or absent, are satisfiable?

22.7 Show that the set of (code numbers of) second-order sentences true in the standard model of arithmetic is not analytical.

22.8 Show that $\mathbf{P}^{\mathrm{II}}$ is not logically equivalent to any first-order sentence.

22.9 Show that for any first- or second-order sentence A of the language of arithmetic, either $\mathbf{P}^{\mathrm{II}} \,\&\, A$ is equivalent to $\mathbf{P}^{\mathrm{II}}$, or $\mathbf{P}^{\mathrm{II}} \,\&\, A$ is equivalent to $\mathbf{0} \neq \mathbf{0}$.

22.10 Show that the set of (code numbers of) second-order sentences that are equivalent to first-order sentences is not analytical.

22.11 Prove the Craig interpolation theorem for second-order logic.

23

Arithmetical Definability

Tarski's theorem tells us that the set V of (code numbers of) first-order sentences of the language in arithmetic that are true in the standard interpretation is not arithmetically definable. In section 23.1 we show that this negative result is poised, so to speak, between two positive results. One is that for each n the set V_n of sentences of the language of arithmetic of degree of complexity n that are true in the standard interpretation is arithmetically definable (in a sense of degree of complexity to be made precise). The other is that the class {V} of sets of natural numbers whose one and only member is V is arithmetically definable (in a sense of arithmetical definability for classes to be made precise). In section 23.2 we take up the question whether the class of arithmetically definable sets of numbers is an arithmetically definable class of sets. The answer is negative, according to Addison's theorem. *This result is perhaps most interesting on account of its method of proof, which is a comparatively simple application of the method of* forcing *originally devised to prove the independence of the continuum hypothesis in set theory (as alluded to in the historical notes to Chapter 18).*

23.1 Arithmetical Definability and Truth

Throughout this chapter we use L and $\mathcal{N}$ for the language of arithmetic and its standard interpretation (previously called L^* and $\mathcal{N}^*$), and V for the set of code numbers of first-order sentences of L ture in $\mathcal{N}$. It will be convenient to work with a version of logic in which the only operators are $\sim$ and $\vee$ and $\exists$ (& and $\forall$ being treated as unofficial abbreviations). We measure the 'complexity' of a sentence by the number of occurrences of logical operators $\sim$ and $\vee$ and $\exists$ in it. (Our results do, however, go through for other reasonable notions of measures of complexity: see the problems at the end of the chapter). By V_n we mean the set of code numbers of first-order sentences of L of complexity $\leq n$ that are true in $\mathcal{N}$.

We are going to be discussing natural numbers, sets of natural numbers, and sets of sets of natural numbers. To keep the levels straight, we generally use *numbers* for the natural numbers, *sets* for the sets of natural numbers, and *classes* for the sets of sets of natural numbers. We write L^c for the expansion of L by adding a constant c, and $\mathcal{N}^c_a$ for the expansion of $\mathcal{N}$ that assigns c as denotation the number a. Then a set S of numbers is arithmetically definable if and only if there is a sentence $F(c)$ of L^c such that S is precisely the set of a for which $F(c)$ is true in $\mathcal{N}^c_a$. Analogously, we write L^G for the expansion of L by adding a one-place predicate G, and $\mathcal{N}^G_A$ for the expansion of $\mathcal{N}$ that assigns G as denotation the set A. And we say a class Σ of sets of numbers is *arithmetically definable* if and only if there is a sentence $F(G)$ of L^G such that Σ is precisely the set of A for which $F(G)$ is true in $\mathcal{N}_A^G$.

The following two results contrast with Tarski's theorem to the effect that V is not arithmetically definable.

23.1 Theorem. For each n, V_n is arithmetically definable.

23.2 Theorem. The class $\{V\}$ whose one and only member is V is arithmetically definable.

This entire section will be devoted to the proofs. We are going to need certain facts about recursiveness (or the 'arithmetization of syntax'):

(1) The set S of code numbers of the sentences of L is recursive.

(2) For each n, the set S_n of code numbers of sentences of L with no more than n occurrences of logical operators is recursive.

(3) There exists a recursive function ν such that if B is a sentence of L with code number b, then $\nu(b)$ is the code number of $\sim B$.

(4) There exists a recursive function δ such that if B and C are sentences of L with code numbers b and c, then $\delta(b, c)$ is the code number of $(B \vee C)$.

(5) There exists a recursive function η such that if v is a variable with code numbers q and $F(v)$ is a formula with code number p, then $\eta(p, q)$ is the code number of $\exists v F(v)$.

(6) There exists a recursive function σ such that if v, $F(v)$, q, p are as in (5), then for any m, $\sigma(p, q, m)$ is the code number of $F(\mathbf{m})$.

(7) The set V_0 of atomic sentences of L that are true in $\mathcal{N}$ is recursive.

[We may suppose that $\nu, \delta, \eta, \sigma$ take the value 0 for inappropriate arguments; for instance, if b is not the code number for a sentence, then $\nu(b) = 0$.]

In every case, intuitively it is more or less clear that the set or function in question is effectively decidable or computable, so according to Church's thesis they should all be recursive. Proofs not depending on appeal to Church's thesis have been given for (1) and (3)–(6) in Chapter 15; and the proof for (2) is very similar and could easily have been included there as well (or placed among the problems at the end of that chapter).

As for (7), perhaps the simplest proof is to note that V_0 consists of the elements of the recursive set S_0 that are theorems of **Q**, and equivalently whose negations are *not* theorems of **Q** (since **Q** proves all true atomic sentences and disproves all false ones). But we know from Chapter 15 that the set of theorems of **Q** or any axiomatizable theory is a semirecursive set, and the set of sentences whose negations are *not* theorems is the complement of a semirecursive set. It follows that V_0 is both a semirecursive set and the complement of one, and is therefore recursive.

The sets above all being recursive, they are definable in arithmetic and indeed in **Q**, and the functions are representable. Let $S(x)$, $S^0(x)$, $S^1(x)$, $S^2(x)$, ..., $\mathrm{Nu}(x, y)$, $\mathrm{Delta}(x, y, z)$, $\mathrm{Eta}(x, y, z)$, $\mathrm{Sigma}(x, y, z, w)$, and $V^0(x)$ be defining or representing formulas for S, S_0, S_1, S_2, ..., ν, δ, η, σ, and V_0. This machinery will be used in the proofs of both theorems.

Proof of Theorem 23.1: A sentence A that contains $n+1$ logical operators is either the negation $\sim B$ of a sentence B containing n operators, or the disjunction $B \vee C$ of two sentences B and C each containing at most n operators, or else an existential quantification $\exists v F(v)$ of a formula $F(v)$ containing n operators. In the last case, for each m, the sentence $F(\mathbf{m})$ also contains n operators. In the first case A will be true if and only if B is not true. In the second case, A will be true if and only if B is true or C is true. In the third case, A will be true if and only if, for some m, $F(\mathbf{m})$ is true.

In terms of V_n, we can therefore characterize V_{n+1} as the set of those numbers a in S_{n+1} such that either k is in V_n; or for some b, $a = \nu(b)$ and b is not in V_n; or for some b and c, $a = \delta(b, c)$ and either b is in V_n or c is in V_n; or finally, for some p and q, $a = \eta(p, q)$, and for some m, $\sigma(p, q, m)$ is in V_n. So if $V^n(x)$ arithmetically defines V_n, the following formula $V^{n+1}(x)$ arithmetically defines V_{n+1}:

$$\begin{aligned} S^{n+1}(x)\,\&\,\{V^n(x) \vee \exists y[&\mathrm{Nu}(y, x)\,\&\sim V^n(y)] \\ &\vee \exists y \exists z[\mathrm{Delta}(y, z, x)\,\&\,(V^n(y) \vee V^n(z))] \\ &\vee \exists y \exists z[\mathrm{Eta}(y, z, x)\,\&\,\exists u \exists w(\mathrm{Sigma}(y, z, u, w)\,\&\,V^n(w))]\}. \end{aligned}$$

Since we know V_0 is arithmetically definable, it follows by induction that V_n is arithmetically definable for all n.

Proof of Theorem 23.2: The set of sentences true in $\mathcal{N}$ can be characterized as the unique set Γ such that:

Γ contains only sentences of L.

For any atomic sentence A, A is in Γ if and only if A is a true atomic sentence.

For any sentence B, $\sim B$ is in Γ if and only if B is not in Γ.

For any sentences B and C, $(B \vee C)$ is in Γ if and only if B is in Γ or C is in Γ.

For any variable v and formula $F(v)$, $\exists v F(v)$ is in Γ if and only if, for some m, $F(\mathbf{m})$is in Γ.

The set V of code numbers of sentences true in $\mathcal{N}$ can therefore be characterized as the unique set M such that:

For all b, if b is in M, then b is in S.

For all a, if a is in S_0, then a is in M if and only if a is in V_0.

For all b, if $\nu(b)$ is in S, then $\nu(b)$ is in M if and only if b is not in M.

For all b and c, if $\delta(b, c)$ is in S, then $\delta(b, c)$ is in M if and only if either b is in M or c is in M.

For all p and q, if $\eta(p, q)$ is in S, then $\eta(p, q)$ is in M if and only if, for some m, $\sigma(p, q, m)$ is in M.

So on expanding L be adding the one-place predicate G, if we let $F(G)$ be the conjunction

$$\begin{aligned} &\forall x(Gx \rightarrow S(x))\,\& \\ &\quad \forall x(S^0(x) \rightarrow (Gx \leftrightarrow V^0(x)))\,\& \\ &\quad \forall x \forall y((\mathrm{Nu}(x, y)\,\&\,S(y)) \rightarrow (Gy \leftrightarrow \sim Gx))\,\& \\ &\quad \forall x \forall y \forall z((\mathrm{Delta}(x, y, x)\,\&\,S(z)) \rightarrow (Gz \leftrightarrow (Gx \vee Gy)))\,\& \\ &\quad \forall x \forall y \forall z((\mathrm{Eta}(x, y, z)\,\&\,S(z)) \rightarrow (Gz \leftrightarrow \exists u \exists w(\mathrm{Sigma}(x, y, u, w)\,\&\,Gw))) \end{aligned}$$

then the only way to expand $\mathcal{N}$ to get a model of $F(G)$ is to take V as the denotation of G.

23.2 Arithmetical Definability and Forcing

We retain the terminology and notation of the preceding section. This entire section will be devoted to the proof of the following result.

23.3 Theorem (Addison's theorem). The class of arithmetically definable sets of numbers is not an arithmetically definable class of sets.

The first notion we need is that of a *condition*, by which we mean a finite, consistent set of sentences of the language L^G each either of the form $G\mathbf{m}$ or $\sim G\mathbf{m}$. The empty set $\varnothing$ is a condition. Other examples are $\{G\mathbf{17}\}$, $\{G\mathbf{17}, \sim G\mathbf{59}\}$, and

$$\{G\mathbf{0}, G\mathbf{1}, G\mathbf{2}, \ldots, G\mathbf{999\,999}, G\mathbf{1\,000\,000}\}.$$

We use p, q, r as variables for conditions. We say a condition q *extends* or is an *extension* of a condition p if p is a subset of q. Thus every condition extends itself and extends $\varnothing$.

Forcing is a relation between certain conditions and certain sentences of L^G. We write $p \Vdash S$ to mean that condition p forces sentence S. The relation of forcing is inductively defined by the following five stipulations:

(1) If S is an atomic sentence of L, then $p \Vdash S$ if and only if $\mathcal{N} \models S$.

(2) If t is a term of L and m is the denotation of t in $\mathcal{N}$, then if S is the sentence Gt, then $p \Vdash S$ if and only if $G\mathbf{m}$ is in p.

(3) If S is a disjunction $(B \vee C)$, then $p \Vdash S$ if and only if either $p \Vdash B$ or $p \Vdash C$.

(4) If S is an existential quantification $\exists x B(x)$, then $p \Vdash S$ if and only if, for some n, $p \Vdash B(\mathbf{n})$.

(5) If S is a negation $\sim B$, then $p \Vdash S$ if and only if, for every q that extends p, it is not the case that $q \Vdash S$.

The last clause bears repeating: a condition forces the negation of a sentence if and only if no extension forces the sentence. It follows that no condition forces some sentence and its negation, and also that either a condition forces the negation of a sentence or some extension forces the sentence. (It will soon be shown that if a condition forces a sentence, so does every extension of it.)

It follows from (2) and (5) that $p \Vdash \sim G\mathbf{m}$ if and only if $\sim G\mathbf{m}$ is in p. For if $\sim G\mathbf{m}$ is not in p, then $p \cup \{G\mathbf{m}\}$ is an extension of p that forces $G\mathbf{m}$. So if $p \Vdash \sim G\mathbf{m}$, that is, if no extension of p forces $G\mathbf{m}$, then $\sim G\mathbf{m}$ must be in p. Conversely, if $\sim G\mathbf{m}$ is in p, then $G\mathbf{m}$ is in no extension of p, and hence no extension of p forces $G\mathbf{m}$, and so $p \Vdash \sim G\mathbf{m}$.

Thus $\{G\mathbf{3}\}$ forces neither $G\mathbf{11}$ nor $\sim G\mathbf{11}$, and so does not force $(G\mathbf{11} \vee \sim G\mathbf{11})$. Thus a condition may imply a sentence without forcing it. (It will soon be seen that the inverse is also possible; that, for example, $\varnothing$ forces $\sim\sim\exists x Gx$, even though it does not imply it, and does not force $\exists x Gx$.)

23.4 Lemma. If $p \Vdash S$ and q extends p, then $q \Vdash S$.

Proof: Suppose that $p \Vdash S$ and q extends p. The proof that $q \Vdash S$ is by induction on complexity of S. The atomic case has two subcases. If S is an atomic sentence of L, then since $p \Vdash S$, S is true in $\mathcal{N}$, and since S is true in $\mathcal{N}$, $q \Vdash S$. If S is an atomic sentence of form Gt, then since $p \Vdash S$, $G\mathbf{m}$ is in p, where m is the denotation of t in $\mathcal{N}$, and since q extends p, $G\mathbf{m}$ is also in q and $q \Vdash Gt$. If S is $(B \vee C)$, then since $p \Vdash S$, either $p \Vdash B$ or $p \Vdash C$; so by the induction hypothesis, either $q \Vdash B$ or $q \Vdash C$, and so $q \Vdash (B \vee C)$. If S is $\exists x B(x)$, then since $p \Vdash S$, we have $p \Vdash B(\mathbf{m})$ for some m; so by the induction hypothesis, $q \Vdash B(\mathbf{m})$ and $q \Vdash \exists x B(x)$. Finally, if S is $\sim B$, then since $p \Vdash S$, no extension of p forces B; and then, since q is an extension of p, every extension of q is an extension of p, so no extension of q forces B, and so $q \Vdash \sim B$.

Two observations, not worthy of being called lemmas, follow directly from the preceding lemma. First, if $p \Vdash B$, then $p \Vdash \sim\sim B$; for any extension of p will force B, hence no extension of p will force $\sim B$. Second, if $p \Vdash \sim B$ and $p \Vdash \sim C$, then $p \Vdash \sim(B \vee C)$; for every extension of p will force both $\sim B$ and $\sim C$, and so will force neither B nor C, and so will not force $(B \vee C)$.

A more complicated observation of the same kind may be recorded here for future reference, concerning the sentence

$$(*) \qquad \sim(\sim(\sim B \vee \sim C) \vee \sim(B \vee C))$$

which is a logical equivalent of $\sim(B \leftrightarrow C)$. Suppose $p \Vdash B$ and $p \Vdash \sim C$. Then $p \Vdash (\sim B \vee \sim C)$, so by our first observation in the preceding paragraph, $p \Vdash \sim\sim(\sim B \vee \sim C)$. Also $p \Vdash (B \vee C)$, so $p \Vdash \sim\sim(B \vee C)$. Hence by our second observation, $p \Vdash (*)$. Similarly, if $p \Vdash \sim B$ and $p \Vdash C$, then again $p \Vdash (*)$.

23.5 Lemma. If S is a sentence of L, then for every p, $p \Vdash S$ if and only if $\mathcal{N} \models S$.

Proof: The proof again is by induction on the complexity of S. If S is atomic, the assertion of the lemma holds by the first clause in the definition of forcing. If S is $(B \vee C)$, then $p \Vdash S$ if and only if $p \Vdash B$ or $p \Vdash C$, which by the induction hypothesis is so if and only $\mathcal{N} \models B$ or $\mathcal{N} \models C$, which is to say, if and only if $\mathcal{N} \models (B \vee C)$. If S is $\exists x B(x)$, the proof is similar. If S is $\sim B$, then $p \Vdash S$ if and only if no extension of p forces B, which by the induction hypothesis is so if and only if it is not the case that $\mathcal{N} \models B$, which is to say, if and only if $\mathcal{N} \models \sim B$.

Forcing is a curious relation. Since $\varnothing$ does not contain any sentence $G\mathbf{n}$, for no n does $\varnothing$ force $G\mathbf{n}$, and therefore $\varnothing$ does not force $\exists x Gx$. But $\varnothing$ does force $\sim\sim\exists x Gx$! For suppose some p forces $\sim\exists x Gx$. Let n be the least number such that $\sim G\mathbf{n}$ is not in p. Let q be $p \cup \{G\mathbf{n}\}$. Then q is a condition, q extends p, and q forces $G\mathbf{n}$, so q forces $\exists x Gx$. Contradiction. Thus no p forces $\sim\exists x Gx$, which is to say, no extension of $\varnothing$ forces $\sim\exists x Gx$, so $\varnothing$ forces $\sim\sim\exists x Gx$.

We are going to need some more definitions. Let A be a set of numbers. First, we call a condition p *A-correct* if for any m, if $G\mathbf{m}$ is in p, then m is in A, while if $\sim G\mathbf{m}$ is in p, then m is not in A. In other words, p is A-correct if and only if

$\mathcal{N}_A^G$ (the expansion of the standard interpretation $\mathcal{N}$ of the language of arithmetic L to an interpretation of the language L^G in which the new predicate G is taken to denote A) is a model of p.

Further, say A FORCES S if some A-correct condition forces S. Note that the union of any two A-correct conditions is still a condition and is still A-correct. It follows that A cannot FORCE both S and $\sim S$, since the union of an A-correct condition forcing S with one forcing $\sim S$ would force both, which is impossible.

Finally, we call A *generic* if for every sentence S of L^G, either A FORCES S or A FORCES $\sim S$. If this is so at least for every sentence S with at most n occurrences of logical operators, we call A *n-generic*. Thus a set is generic if and only if it is n-generic for all n.

The first fact about generic sets that we have to prove is that they exist.

23.6 Lemma. For any p, there is a generic set A such that p is A-correct.

Proof: Let $S_0, S_1, S_2, \ldots$ be an enumeration of all sentences of L^G. Let p_0, p_1, $p_2, \ldots$ be an enumeration of all conditions. We inductively define a sequence q_0, $q_1, q_2, \ldots$ of conditions, each an extension of those that come before it, as follows:

(0) q_0 is p.

(1) If q_i forces $\sim S_i$, then q_{i+1} is q_i.

(2) If q_i does not force $\sim S_i$, in which case there must be some q extending q_i and forcing S_i, then q_{i+1} is the first such q (in the enumeration p_0, p_1, $p_2, \ldots$).

Let A be the set of m such that $G\mathbf{m}$ is in q_i for some i.

We claim that p is A-correct and that A is generic. Since $p = q_0$, and since for each i, either $q_{i+1} \Vdash S_i$ or $q_{i+1} \Vdash \sim S_i$, it will be enough to show that for each i, q_i is A-correct. And since m is in A when $G\mathbf{m}$ is in q_i, it is enough to show that if $\sim G\mathbf{m}$ is in q_i, then m is *not* in A. Well, suppose it were. Then $G\mathbf{m}$ would be in q_j for some j. Letting $k = \max(i, j)$, both $\sim G\mathbf{m}$ and $G\mathbf{m}$ would be in q_k, which is impossible. This contradiction completes the proof.

The next fact about generic sets relates FORCING and truth.

23.7 Lemma. Let S be a sentence of L^G, and A a generic set. Then A FORCES S if and only if $\mathcal{N}_A^G \models S$.

Proof: The proof will be yet another by induction on complexity, with five cases, one for each clause in the definition of forcing. We abbreviate 'if and only if' to 'iff'.

Case 1. S is an atomic sentence of L. Then A FORCES S iff some A-correct p forces S, iff (by Lemma 23.5) $\mathcal{N} \models S$, iff $\mathcal{N}_A^G \models S$.

Case 2. S is an atomic sentence Gt. Let m be the denotation of t in $\mathcal{N}$. Then A FORCES S iff some A-correct p forces Gt, iff $G\mathbf{m}$ is in some A-correct p, iff m is in A, iff $\mathcal{N}_A^G \models Gt$.

Case 3. S is $(B \vee C)$. Then A FORCES S iff some A-correct p forces $(B \vee C)$, iff some A-correct p forces B or forces C, iff either some A-correct p forces B or some A-correct p forces C, iff A FORCES B or A FORCES C, iff (by the induction hypothesis) $\mathcal{N}_A^G \models B$ or $\mathcal{N}_A^G \models C$, iff $\mathcal{N}_A^G \models (B \vee C)$.

Case 4. S is $\exists x B(x)$. Then A FORCES S iff some A-correct p forces $\exists x B(x)$, iff for some A-correct p there is an m such that p forces $B(\mathbf{m})$, iff for some m there is an A-correct p such that p forces $B(\mathbf{m})$, iff for some m, A forces $B(\mathbf{m})$, iff (by the induction hypothesis) for some m, $\mathcal{N}_A^G \models B(\mathbf{m})$, iff $\mathcal{N}_A^G \models \exists x B(x)$.

Case 5. S is $\sim B$. No set FORCES both B and $\sim B$. Since A is generic, A FORCES at least one of B or $\sim B$. Hence A FORCES $\sim B$ iff it is not the case that A FORCES B, iff (by the induction hypothesis) not $\mathcal{N}_A^G \models B$, iff $\mathcal{N}_A^G \models \sim B$.

The last fact about generic sets that we have to prove is that none of them is arithmetical.

23.8 Lemma. No generic set is arithmetical.

Proof: Suppose otherwise. Then there is a generic set A and a formula $B(x)$ of L such that for every n, n is in A if and only if $\mathcal{N} \models B(\mathbf{n})$. So $\mathcal{N}_A^G \models \forall x(Gx \leftrightarrow B(x))$ or $\mathcal{N}_A^G \models \sim\exists x F(x)$, where $F(x)$ is the following logical equivalent of $\sim(Gx \leftrightarrow B(x))$:

$$\sim(\sim(\sim Gx \vee \sim B(x)) \vee \sim(Gx \vee B(x))).$$

By Lemma 23.7, A FORCES $\sim\exists x F(x)$, so some A-correct p forces $\sim\exists x F(x)$, so for no q extending p and no n does q force $F(\mathbf{n})$, which is to say

$$(*) \qquad \sim(\sim(\sim G\mathbf{n} \vee \sim B(\mathbf{n})) \vee \sim(G\mathbf{n} \vee B(\mathbf{n}))).$$

Let k be the least number such that neither $G\mathbf{k}$ nor $\sim G\mathbf{k}$ is in p. Define a condition q extending p by letting $q = p \cup \{G\mathbf{k}\}$ if $\mathcal{N} \models \sim B(\mathbf{k})$ and letting $q = p \cup \{\sim G\mathbf{k}\}$ if $\mathcal{N} \models B(\mathbf{k})$. In the former case, $q \Vdash G\mathbf{k}$, while by Lemma 23.5 $q \Vdash \sim B(\mathbf{k})$. In the latter case, $q \Vdash \sim G\mathbf{k}$ while by Lemma 23.5 $q \Vdash B(\mathbf{k})$. In either case, $q \Vdash (*)$ by our observations following Lemma 23.4, which is to say $q \Vdash F(\mathbf{n})$. Contradiction.

Suppose that at the beginning of the proof of Lemma 23.6, instead of enumerating all sentences we enumerate those sentences of complexity $\leq n$ (that is, having no more than n occurrences of logical operators). Then the proof would establish the existence of an n-generic set rather than of a generic set. Suppose that in the hypothesis of Lemma 23.7 we only assume the set A is n-generic rather than generic. Then the proof would establish the conclusion of Lemma 23.7 for sentences of complexity $\leq n$, rather than for all sentences. But suppose that in the hypothesis of Lemma 23.8 we only assume the set A is n-generic rather than generic. Then the proof would break down entirely. And indeed, in contrast to Lemma 23.8, we have the following.

23.9 Lemma. For any n, there is an n-generic set A that is arithmetical.

Proof: The proof will be indicated only in outline. The idea is to carry out the construction in the proof of Lemma 23.6, starting from an enumeration of all sentences of complexity $\leq n$, and with $p = \varnothing$. It is necessary to show that, if code numbers are assigned in a suitable way, then various relations among code numbers connected with the construction will be arithmetical, with the result that the generic set constructed is arithmetical as well.

First note that, since we have seen in the preceding section that the set of code numbers of sentences of complexity $\leq n$ is recursive, the function enumerating the elements of that set in increasing order is recursive. That is, if we enumerate the sentences $S_0, S_1, S_2, \ldots$ in order of increasing code number, then the function taking us from i to the code number for S_i will be recursive.

We also enumerate the conditions $p_0, p_1, p_2, \ldots$ in order of increasing code number, where code numbers are assigned to finite sets of sentences—for that is what conditions are—as in section 15.2. As we observed in section 15.2, the relation 'the sentence with code number i belongs to the set with code number s' is recursive. Using this fact and the fact that the function taking m to the code number for $G\mathbf{m}$—essential the substitution function σ used in the preceding section—is recursive, it is not hard to show that the set of code numbers of conditions is recursive, and that the relation that holds between m and s if and only if s is the code number of a condition containing $G\mathbf{m}$ is recursive. We also observed in section 15.2 that the relation 'the set with code number s is a subset of the set with code number t' is recursive. Hence the relation that holds between s and t if and only if they are code numbers of conditions p and q, with q an extension of p, is also recursive. Being recursive, the various functions and relations we have mentioned are all arithmetical.

We also need one more fact: that for each n, the relation that holds between i and s if and only if i is the code number of a sentence S of complexity $\leq n$ and s is the code number of a condition p, and p forces S, is arithmetical. The proof is very similar to the proof in the preceding section that for each n the set V_n is arithmetical, and will be left to the reader.

Now returning to the construction of an n-generic set A, by the method of the proof of Lemma 23.6, we see that m is in A if and only if there exists a sequence s of conditions such that the following hold (for each i less than the length of the sequence):

(0) The 0th entry of the sequence is the empty condition $\varnothing$

(1) If the ith entry of the sequence forces the negation of the ith sentence in the enumeration of sentences, then the $(i + 1)$st entry is the same as the ith.

(2) Otherwise, the $(i + 1)$st entry is a condition that extends the ith and that forces the ith sentence in the enumeration of sentences, and is such that no condition earlier in the enumeration of conditions (that is, no condition of smaller code number) does both these things.

(3) The sentence $G\mathbf{m}$ belongs to the last entry of the sequence.

We can, of course, replace 'there exists a sequence...' by 'there exists a code number for a sequence...'. When everything is thus reformulated in terms of code numbers, what we get is a logical compound of relations that we have noted in the preceding several paragraphs to be arithmetical. It follows that A itself is arithmetical.

At last we are in a position to prove Addison's theorem.

Proof of Theorem 23.3: Suppose the theorem fails. Then there is a sentence S of L^G such that for any set $A, \mathcal{N}_A^G \models S$ if and only if A is arithmetical. Let S be of complexity n. By Lemma 23.9 there exists an n-generic set A that is arithmetical. So $\mathcal{N}_A^G \models S$. So by Lemma 23.7 (or rather, the version for n-generic sets and sentences of complexity $\leq n$, as in our remarks following Lemma 23.8), A FORCES S. So some

A-correct p forces S. By Lemma 23.6, there exists a (fully) generic set A^* such that p is A^*-correct. Since p forces S, by Lemma 23.7 (in its original version), $\mathcal{N}^G_{A^*} \models S$. But this means A^* is arithmetical, contrary to Lemma 23.8.

Problems

23.1 Use Beth's definability theorem, Tarski's theorem on the first-order indefinability of first-order arithmetic truth, and the results of section 23.1 to obtain another proof of the existence of nonstandard models of arithmetic.

23.2 Show that for each n the set of (code numbers of) true prenex sentences of the language of arithmetic that contain at most n quantifiers is arithmetical. Show the same with 'prenex' omitted.

23.3 Show that if $p \Vdash {\sim}{\sim}{\sim}B$, then $p \Vdash {\sim}B$.

23.4 Given an example of a sentence B such that the set of even numbers FORCES neither B nor ${\sim}B$.

23.5 Show that the set of pairs (i, j) such that j codes a sentence of L^G and i codes a condition that forces that sentence is not arithmetical.

23.6 Where would the proof of Addison's theorem have broken down if we had worked with $\sim$, $\&$, $\forall$ rather than $\sim$, $\vee$, $\exists$ (and made the obvious analogous stipulations in the definition of forcing)?

23.7 Show that the only arithmetical subsets of a generic set are its finite subsets.

23.8 Show that if A is generic, then $\{A\}$ is not arithmetical.

23.9 Show that $\{A : A \text{ is generic}\}$ is not arithmetical.

23.10 Show that every generic set contains infinitely many prime numbers.

23.11 Show that the class of generic sets is nonenumerable.

23.12 A set of natural numbers is said to have density r, where r is a real number, if r is the limit as n goes to infinity of the ratio (number of members of $A < n)/n$. Show that no generic set has a density.

24

Decidability of Arithmetic without Multiplication

Arithmetic is not decidable: the set V of code numbers of sentences of the language L of arithmetic that are true in the standard interpretation is not recursive (nor even arithmetical). But for some sublanguages L of L, if we consider the elements of V that are code numbers of sentences of L*, then the set V* of such elements* is *recursive: arithmetic without some of the symbols of its language is decidable. A striking case is* Presburger arithmetic*, or arithmetic without multiplication. The present chapter is entirely devoted to proving its decidability.*

We have used *(true) arithmetic* to mean the set of sentences of the language of arithmetic $L = \{\mathbf{0}, <, ', +, \cdot\}$ that are true in the standard interpretation $\mathcal{N}$. By *arithmetic without multiplication* we mean the set of sentences of (true) arithmetic that do not contain the symbol $\cdot$. By *arithmetic without addition* we mean the set of sentences of (true) arithmetic that do not contain the symbols $<, ', +$. In contrast to the undecidability of arithmetic stand Presburger's theorem, to the effect that arithmetic without multiplication is decidable, and Skolem's theorem, to the effect that arithmetic without addition is decidable. [Note in connection with the latter theorem that $'$ is easily definable in terms of $<$ and that $+$ is definable in terms of $'$ and $\cdot$, as follows:

$$x + y = z \leftrightarrow (x' \cdot z'')' \cdot (y' \cdot z'') = ((x' \cdot y')' \cdot (z'' \cdot z''))'.$$

That is why $<$ and $'$ have to be dropped along with $+$.] This chapter will be entirely devoted to proving the former theorem, by describing an effective procedure for determining whether or not a given sentence of the language of arithmetic not involving $\cdot$ is true in the standard interpretation.

We begin with a reduction of the problem. Let K be the language with constants $\mathbf{0}$ and $\mathbf{1}$, infinitely many one-place predicates $\mathbf{D}_2, \mathbf{D}_3, \mathbf{D}_4, \ldots$, the two-place predicate $<$, and the two-place function symbols $+$ and $-$. Let $\mathcal{M}$ be the interpretation with domain the set of *all* integers (positive, zero, negative), and with the following denotations for the nonlogical symbols. $\mathbf{0}, \mathbf{1}, <, +, -$ will denote the usual zero and unity elements, order relation, and addition and subtraction operations on integers. $\mathbf{D}_n$ will denote the set of integers divisible without remainder by n.

Given a sentence S of L without $\cdot$, replace $'$ everywhere in it by $\mathbf{+1}$, and replace every quantification $\forall x$ or $\exists x$ by a *relativized* quantification

$$\forall x((x = \mathbf{0} \vee \mathbf{0} < x) \rightarrow \cdots) \qquad \exists x((x = \mathbf{0} \vee \mathbf{0} < x)\,\&\, \ldots).$$

to obtain a sentence S^* of K. Then S will be true in $\mathcal{N}$ if and only if S^* is true in $\mathcal{M}$. Thus to prove Presburger's theorem, it will be sufficient to describe an effective procedure for determining whether or not a given sentence of K is true in $\mathcal{M}$.

For the remainder of this chapter, therefore, *term* or *formula* or *sentence* will always mean term or formula or sentence *of* K, while *denotation* or *satisfaction* or *truth* will always mean denotation or satisfaction or truth *in* $\mathcal{M}$. We call two terms r and s *coextensive* if $\forall v_1 \ldots \forall v_n\, r = s$ is true, where the v_i are all the variables occurring in r or s. We call two formulas F and G *coextensive* if $\forall v_1 \ldots \forall v_n (F \leftrightarrow G)$ is true, where the v_i are all the free variables occurring in F or G.

Given any closed term, we can effectively calculate its denotation. Given any atomic sentence, we can effectively determine its truth value; and we can therefore do the same for any quantifier-free sentence. We are going to show how one can effectively decide whether a given sentence S is true by showing how one can effectively associate to S a coextensive quantifier-free sentence T: once T is found, its truth value, which is also the truth value of S, can be effectively determined.

The method to be used for finding T, given S, is called *elimination of quantifiers*. It consists in showing how one can effectively associate to a quantifier-free formula $F(x)$, which may contain other free variables besides x, and quantifier-free G such that G is coextensive with $\exists x\, F(x)$ and G contains no additional free variables beyond the free variables in $\exists x\, F(x)$. This shown, given S, we put it in prenex form, then replace each quantification $\forall x$ by $\sim\exists x \sim$, and work from the inside outward, successively replacing existential quantifications of quantifier-free formulas by coextensive quantifier-free formulas with no additional free variables, until at last a sentence with no free variables, which is to say, a quantifier-free sentence T, is obtained.

So let $F(x)$ be a quantifier-free formula. We obtain G, coextensive with $\exists x\, F(x)$ and containing no addition free variables beyond those in $\exists x\, F(x)$, by performing, in order, a sequence of 30 operations, each of which replaces a formula by a coextensive formula with no additional free variables.

In describing the operations to be gone through, we make use of certain notational conventions. When writing of a positive integer k and a term t we allow ourselves to write

$-t$	instead of	$\mathbf{0} - t$
k	instead of	$\mathbf{1} + \mathbf{1} + \cdots + \mathbf{1}$ (k times)
kt	instead of	$t + t + \cdots + t$ (k times)

for instance. With such notation, the 30 operations are as follows:

(1) Put F into disjunctive normal form. (See section 19.1.) Thus we get a disjunction of conjunctions of atomic formulas of the forms $r = s$ or $r < s$ or $\mathbf{D}_m s$ (where r and s are terms) and negations of such.

(2) Replace each formula of form $r = s$ by $(r < s + \mathbf{1} \,\&\, s < r + \mathbf{1})$.

(3) Replace each formula of form $r \neq s$ by $(r < s \vee s < r)$.

(4) Put the result back into disjunctive normal form.

(5) Replace each formula of form $\sim r < s$ by $s < r + 1$.

(6) Replace each formula of form $\sim\mathbf{D}_m s$ by the disjunction of $\mathbf{D}_m(s + i)$ for all i with $0 < i < m$. The result is coextensive with the original, because for any number a, m divides exactly one of $a, a+1, a+2, \ldots, a+m-1$.

(7) Put the result back into disjunctive normal form.

(8) At this point we have a disjunction of conjunctions of atomic formulas of the forms $r < s$ and $\mathbf{D}_m s$. Replace each formula of form $r < s$ by $\mathbf{0} < (s - r)$.

(9) We say a term is in *normal form* if it has one of the five forms $kx, -kx, kx + t, -kx + t$, or t, wherein t is a term not containing the variable x. For every term one can effectively find a coextensive term in normal form by ordinary algebraic operations, such as regrouping and reordering summands. Replace each term in the formula that is not in normal form by a coextensive one that is.

(10) Replace each formula of form

$$\mathbf{0} < -kx, \qquad \mathbf{0} < kx + t, \quad \text{or} \quad \mathbf{0} < -kx + t$$

by

$$kx < \mathbf{0}, \qquad -t < kx, \quad \text{or} \quad kx < t$$

as the case may be.

(11) At this point all atomic formulas with predicate $<$ that contain the variable x have either the form $t < kx$ or the form $kx < t$, where k is positive and t does not contain x. We call those of the former form *lower inequalities* and those of the latter form *upper inequalties*. Rearrange the order of conjuncts in each disjunct so that all lower inequalities occur on the left.

(12) Towards reducing the number of lower inequalities occurring in any disjunct, if a conjunction of form $t_1 < k_1 x \,\&\, t_2 < k_2 x$ occurs in a disjunct, replace it by the disjunction of the following three conjunctions:

(i) $t_1 < k_1 x \;\;\&\;\; k_1 t_2 < k_2 t_1$
(ii) $t_1 < k_1 x \;\;\&\;\; k_1 t_2 = k_2 t_1$
(iii) $t_2 < k_2 x \;\;\&\;\; k_2 t_1 < k_1 t_2$.

To see that this substitution is justified (that is, to see that it produces a result coextensive with the original), note that exactly one of the second conjuncts in (i)–(iii) must hold, and that (i) or (ii) holds, then so do $k_2 t_1 < k_1 k_2 x$ and $k_1 t_2 < k_1 k_2 x$, and hence so does $t_2 < k_2 x$; while similarly (iii) yields $t_1 < k_1 x$.

(13) Eliminate any occurrences of $=$ introduced at the previous step by repeating steps (2) and (4).

(14) The effect of the preceding three steps is to reduce by one the number of lower inequalities in any disjunct where there were more than one to begin with. (Note that $k_1 t_2 < k_2 t_1$, for instance, does *not* count as a lower inequality, since it does not contain the variable x.) Repeat these three steps over and over until no disjunct has more than one lower inequality among its conjuncts.

(15) Carry out an analogous process for upper inequalities, until no disjunct has more than one lower *or* upper inequality among its conjuncts.

(16) Replace each formula of form

$$\mathbf{D}_m(kx), \qquad \mathbf{D}_m(-kx), \qquad \mathbf{D}_m(kx+t), \quad \text{or} \quad \mathbf{D}_m(-kx+t)$$

by

$$\mathbf{D}_m(kx-\mathbf{0}), \qquad \mathbf{D}_m(kx-\mathbf{0}), \qquad \mathbf{D}_m(kx-(-t)), \quad \text{or} \quad \mathbf{D}_m(kx-t)$$

as the case may be. This step is justified because for any number a, m divides a if and only if m divides $-a$.

(17) At this point all atomic formulas with $\mathbf{D}_m$ and involving x have the form $\mathbf{D}_m(kx-t)$, where k is a positive integer. Replace any formula of this form by the disjunction of all conjunctions

$$\mathbf{D}_m(kx-i) \,\&\, \mathbf{D}_m(t-i)$$

for $0 \leq i < m$. To see that this step is justified, note that m divides the difference of two numbers a and b if and only if a and b leave the same remainder on division by m, and that the remainder on dividing a (respectively, b) by m is the unique i with $0 \ \leq i < m$ such that m divides $a - i$ (respectively, $b - i$).

(18) Put the result back into disjunctive normal form.

(19) At this point all atomic formulas with $\mathbf{D}_m$ and involving x have the form $\mathbf{D}_m(kx - i)$, where k is a positive integer and $0 \ \leq i < m$. Replace any formula of this form with $k > 1$ by the disjunction of the formulas $\mathbf{D}_m(x-j)$ for all j with $0 \ \leq j < m$ such that m divides $kj - i$. This step is justified because for any number a, ka leaves a remainder of i on division by m if and only if kj does, where j is the remainder on dividing a by m.

(20) Put the result back into disjunctive normal form.

(21) At this point all atomic formulas with $\mathbf{D}_m$ and involving x have the form $\mathbf{D}_m(x-i)$, where i is a nonnegative integer. In any such case consider the prime decomposition of m; that is, write

$$m = p_1^{e_1} \cdots p_k^{e_k} \qquad \text{where} \quad p_1 < p_2 < \cdots < pk \quad \text{and all } p\text{s are primes.}$$

If $k > 1$, then let $m_1 = p_1^{e_1}, \ldots, m_k = p_k^{e_k}$, and replace $\mathbf{D}_m(x-i)$ by

$$\mathbf{D}_{m_1}(x-i) \,\&\, \ldots \,\&\, \mathbf{D}_{m_k}(x-i).$$

This step is justified because the product of two given numbers having no common factor (such as powers of distinct primes) divides a given number if and only if each of the two given numbers does.

(22) At this point all atomic formulas with $\mathbf{D}$s and involving x have the form $\mathbf{D}_m(x - i)$, where i is a nonnegative integer, and m a power of a prime. If in a given disjunct there are two conjuncts $\mathbf{D}_m(x - i)$ and $\mathbf{D}_n(x - j)$ where m and n are powers of the *same* prime, say $m = p^d$, $n = p^e$, $d \leq e$, then drop $\mathbf{D}_m(x-i)$ in favor of $\mathbf{D}_n(i - j)$, which does not involve x. This step is justified because, since

m divides n, for any number a, if a leaves remainder j on division by n, a will leave remainder i on division by m if and only if j does.

(23) Repeat the preceding step until for any two conjuncts $\mathbf{D}_m(x - i)$ and $\mathbf{D}_n(x - j)$ in a single disjunct, m and n are powers of distinct primes, and therefore have no common factors.

(24) Replace each $\mathbf{D}_m(x - i)$ by $\mathbf{D}_m(x - i^*)$, where i^* is the remainder on dividing i by m.

(25) Rewrite each disjunct so that all atomic formulas with with **D**s and involving x are on the left.

(26) At this point each disjunct has the form

$$\mathbf{D}_{m_1}(x - i_1)\ \&\ \ldots\ \&\ \mathbf{D}_{m_k}(x - i_k)\ \&\ (\text{other conjuncts})$$

where $0 \leq i_1 < m_1, \ldots, 0 \leq i_k < m_k$. Let $m = m_1 \cdot \cdots \cdot m_k$. According to the Chinese remainder theorem (see Lemma 15.5), there exists a (unique) i with $0 \leq i < m$ such that i leaves remainder i_1 on division by $m_1, \ldots, i$ leaves remainder i_k on division by m_k. Replace the conjuncts involving **D**s by the single formula $\mathbf{D}_m(x - i)$.

(27) At this point we have a disjunction $F_1 \vee \cdots \vee F_k$ each of whose disjuncts is a conjunction containing at most one lower inequality, at most one upper inequality, and at most one formula of form $\mathbf{D}_m(x - i)$. Rewrite $\exists x(F_1 \vee \cdots \vee F_k)$ as $\exists x F_1 \vee \cdots \vee \exists x F_k$.

(28) Within each disjunct $\exists xF$, rewrite the conjunction F so that any and all conjuncts involving x occur on the left, and confine the quantifier to these conjuncts, of which there are at most three; if there are none, simply omit the quantifier.

(29) At this point, the only occurrences of x are in sentences of one of the seven types listed in Table 24-1. Replace these by the sentences listed on the right.

Table 24-1. *Elimination of quantifiers*

$\exists x\, s < jx$	$\mathbf{0} < \mathbf{1}$
$\exists x\, kx < t$	$\mathbf{0} < \mathbf{1}$
$\exists x\, \mathbf{D}_m(x - i)$	$\mathbf{0} < \mathbf{1}$
$\exists x(\mathbf{D}_m(x - i)\ \&\ s < jx)$	$\mathbf{0} < \mathbf{1}$
$\exists x(\mathbf{D}_m(x - i)\ \&\ kx < t)$	$\mathbf{0} < \mathbf{1}$
$\exists x(s < jx\ \&\ kx < t)$	$\exists x(\mathbf{D}_{jk}(x - \mathbf{0})\ \&\ ks < x\ \&\ x < jt)$
$\exists x(\mathbf{D}_m\ (x - i)\ \&\ s < jx\ \&\ kx < t)$	$\exists x(\mathbf{D}_{jkm}(x - jki)\ \&\ ks < x\ \&\ x < jt)$

This step is justified in the first five cases because in these cases the sentence on the right is automatically true. (In the fourth case this is because there are arbitrarily large integers leaving a prescribed remainder i on division by m, and similarly in the fifth case.) The sixth and seventh cases are similar to each other. We discuss the latter because it is slightly more complicated. First note that the sentence on the left,

(i) $$\exists x(\mathbf{D}_m(x - i)\ \&\ s < jx\ \&\ kx < t)$$

is coextensive with

(ii) $$\exists x(\mathbf{D}_{jkm}(jkx - jki)\ \&\ ks < jkx\ \&\ jkx < jt).$$

This in turn is coextensive with

(iii) $$\exists y(\mathbf{D}_{jkm}(y - jki) \,\&\, ks < y \,\&\, y < jt)$$

which is the sentence on the right, except for relettering the variable. For if x is as in (ii), then $y = jkx$ will be as in (iii); and conversely, if y is as in (iii), then since jk divides $y - jki$, jk must divide y, which is to say that y will be of the form jkx for some x, which x will then be as in (ii).

(30) At this point, the only occurrences of x are in sentences of the form

$$\exists x(\mathbf{D}_m(x - i) \,\&\, s < x \,\&\, x < t).$$

Replace this by the disjunction of

$$\mathbf{D}_m(s + j - i) \,\&\, s + j < t$$

for all j with $1 \le j \le m$. This step is justified because, given two integers a and b, there will be an integer strictly between them that leaves the same remainder as i when divided by m if and only if one of $a + 1, \ldots, a + m$ is such an integer.

We now have eliminated x altogether, and have obtained a quantifier-free formula coextensive with our original formula and involving no additional free variables, and we are done.

Problems

24.1 Consider monadic logic without identity, and add to it a new quantifier $(\mathsf{M}x)(A(x) > B(x))$, which is to be true if and only if there are more x such that $A(x)$ than there are x such that $B(x)$. Call the result *comparative* logic. Show how to define in terms of M:

(a) $\forall$ and $\exists$ (so that these can be officially dropped and treated as mere abbreviations)

(b) 'most x such that $A(x)$ are such that $B(x)$'

24.2 Define a *comparison* to be a formula of the form $(\mathsf{M}x)(A(x) > B(x))$ where $A(x)$ and $B(x)$ are quantifier-free. Show that any sentence is equivalent to a truth-functional compound of comparisons (which then by relettering may be taken all to involve the same variable x).

24.3 As with sets of sentences of first-order logic, a set of sentences of logic with the quantifier M is (*finitely*) *satisfiable* if there is an interpretation (with a finite domain) in which all sentences in the set come out true. Show that finite satisfiability for finite sets of sentences of logic with the quantifier M is decidable. (The same is true for satisfiability, but this involves more set theory than we wish to presuppose.)

24.4 For present purposes, by an *inequality* is meant an expression of the form

$$a_1x_1 + \cdots + a_mx_m \,\S\, b$$

where the x_i are variables, the a_i and b are (numerals for) specific rational numbers, and § may be any of <, $\le$, >, $\ge$. A finite set of inequalities is *coherent* if there are rational numbers r_i that if taken for the x_i would make each inequality in the set come out true (with respect to the usual addition operation and order relation on rational numbers). Show that there is a decision procedure for the coherence of finite sets of inequalities.

24.5 In *sentential* logic the only nonlogical symbols are an enumerable infinity of *sentence letters*, and the only logical operators are negation, conjunction, and disjunction $\sim$, &, $\vee$. Let $A_1, \ldots, A_n$ be sentence letters, and consider sentences of sentential logic that contain no sentence letters, but the A_i, or equivalently, that are truth-functional compounds of the A_i. For each sequence $e = (e_1, \ldots, e_n)$ of 0s and 1s, let P_e be $(\sim)A_1 \,\&\, \ldots \,\&\, (\sim)A_n$, where for each i, $1 \leq i \leq n$, the negation sign preceding A_i is present if $e_i = 0$, and absent if $e_i = 1$. For present purposes a *probability measure* μ may be defined as an assignment of a rational number $\mu(P_e)$ to each P_e in such a way that the sum of all these numbers is 1. For a truth-functional combination A of the A_i we define $\mu(A)$ to be the sum of the $\mu(P_e)$ for those P_e that imply A, or equivalently, that are disjuncts in the full disjunctive normal form of A). The *conditional probability* $\mu(A\backslash B)$ is defined to be the quotient $\mu(A \,\&\, B)/\mu(A)$ if $\mu(A) \neq 0$, and is conventionally taken to be 1 if $\mu(A) = 0$. For present purposes, by a *constraint* is meant an expression of the form $\mu(A) \,\S\, b$ or $\mu(A\backslash B) \,\S\, b$, where A and B are sentences of sentential logic, b a nonnegative rational number, and § any of $<, \leq, >, \geq$. A finite set of constraints is *coherent* if there exists a probability measure μ that makes each constraint in the set come out true. Is the set of constraints $\mu(A\backslash B) = 3/4$, $\mu(B\backslash C) = 3/4$, and $\mu(A\backslash C) = 1/4$ coherent?

24.6 Show that there is a decision procedure for the coherence of finite sets of constraints.

25

Nonstandard Models

By a model of (true) arithmetic *is meant any model of the set of all sentences of the language L of arithmetic that are true in the standard interpretation $\mathcal{N}$. By a* nonstandard *model is meant one that is not isomorphic to $\mathcal{N}$. The proof of the existence of an (enumerable) nonstandard model of arithmetic is as an easy application of the compactness theorem (and the Löwenheim–Skolem theorem). Every enumerable nonstandard model is isomorphic to a nonstandard model $\mathcal{M}$ whose domain is the same as that of $\mathcal{N}$, namely, the set of natural numbers; though of course such an $\mathcal{M}$ cannot assign the same denotations as $\mathcal{N}$ to the nonlogical symbols of L. In section 25.1 we analyze the structure of the order relation in such a nonstandard model. A consequence of this analysis is that, though the order relation cannot be the standard one, it at least can be a recursive relation. By contrast,* Tennenbaum's theorem *tells us that it cannot happen that the addition and multiplication relations are recursive. This theorem and related results will be taken up in section 25.2. Section 25.3 is a sort of appendix (independent of the other sections, but alluding to results from several earlier chapters) concerning nonstandard models of an expansion of arithmetic called analysis.*

25.1 Order in Nonstandard Models

Let $\mathcal{M}$ be a model of (true) arithmetic not isomorphic to the standard model $\mathcal{N}$. (The existence of such models was established in the problems at the end of Chapter 12, as an application of the compactness theorem.) What does such a model look like? We'll call the objects in the domain $|\mathcal{M}|$ NUMBERS. $\mathcal{M}$ assigns as denotation to the symbol $\mathbf{0}$ some NUMBER O we'll call ZERO, and to the symbol $'$ some function † on NUMBERS we'll call SUCCESSOR. It assigns to $<$ some relation $\prec$ we'll call LESS THAN, and to $+$ and $\cdot$ some functions $\oplus$ and $\otimes$ we'll call ADDITION and MULTIPLICATION. Our main concern in this section will be to understand the LESS THAN relation.

First of all, no NUMBER is LESS THAN itself. For no (natural) number is less than itself. So $\forall x \sim x < x$ is true in $\mathcal{N}$, so it is true in $\mathcal{M}$, and so as asserted no NUMBER is LESS THAN itself. This argument illustrates our main technique for obtaining information about the 'appearance' of $\mathcal{M}$: observe that the natural numbers have a certain property, conclude that a certain sentence of L is true in $\mathcal{N}$, infer that it must also be true in $\mathcal{M}$ (since the same sentences of L are true in $\mathcal{M}$ as in $\mathcal{N}$), and decipher the sentence 'over' $\mathcal{M}$. In this way we can conclude that exactly one of any two NUMBERS is LESS THAN the other, and that if one NUMBER is LESS THAN another, which is LESS

THAN a third, then the first is LESS THAN the third. LESS THAN is a linear ordering of the NUMBERS, just as less than is a linear ordering of the numbers.

Zero is the least number, so ZERO is the LEAST NUMBER. Any number is less than its successor, and there is no number between a given number and its successor (in the sense of being greater than the former and less than the latter), so any NUMBER is LESS THAN its SUCCESSOR, and there is no NUMBER between a given NUMBER and its SUCCESSOR. In particular, $0'$ (that is, 1 or one) is the next-to-least number, and $\mathrm{O}^{\dagger}$ (which we may call I or ONE) is the next-to-LEAST NUMBER; $0''$ is next-to-next-to-least and $\mathrm{O}^{\dagger\dagger}$ is next-to-next-to-LEAST; and so on. So there is an initial segment O, $\mathrm{O}^{\dagger}$, $\mathrm{O}^{\dagger\dagger}$, ... of the relation LESS THAN that is isomorphic to the series 0, $0''$, $0''$, ... of the (natural) numbers.

We call O, $\mathrm{O}^{\dagger}$, $\mathrm{O}^{\dagger\dagger}$, ... the *standard* NUMBERS. Any others are *nonstandard*. The standard NUMBERS are precisely those that can be obtained from ZERO by applying the SUCCESSOR operation a *finite* number of times. For any (natural) number n, let us write $h(n)$ for $\mathrm{O}^{\dagger\dagger\dots\dagger(n\text{ times})}$, which is the denotation of the numeral $\mathbf{n}$ or $\mathbf{0}^{\prime\prime\dots\prime(n\text{ times})}$ in $\mathcal{M}$. Then the standard NUMBERS are precisely the $h(n)$ for n a natural number. Any others are *nonstandard*. Any standard NUMBER $h(n)$ is LESS THAN any nonstandard NUMBER m. This is because, being true in $\mathcal{N}$, the sentence

$$\forall z((z \neq \mathbf{0} \,\&\, \dots \,\&\, z \neq \mathbf{n}) \rightarrow \mathbf{n} < z)$$

must be true in $\mathcal{M}$, so any NUMBER other than $h(0), \dots, h(n)$ must be GREATER THAN $h(n)$.

[It is not quite trivial to show that there must *be* some nonstandard NUMBERS in any nonstandard model $\mathcal{M}$. If there were not, then h would be a function from (natural) numbers *onto* the domain of $\mathcal{M}$. We claim that in that case, h would be an isomorphism between $\mathcal{N}$ and $\mathcal{M}$, which it cannot be if $\mathcal{M}$ is nonstandard. First, h would be one-to-one, because when $m \neq n$, $\mathbf{m} \neq \mathbf{n}$ is true in $\mathcal{N}$ and so in $\mathcal{M}$, so the denotations of $\mathbf{m}$ and $\mathbf{n}$ in $\mathcal{M}$ are distinct, that is, $h(m) \neq h(n)$. Further, when $m + n = p$, $\mathbf{m} + \mathbf{n} = \mathbf{p}$ is true in $\mathcal{N}$ and so in $\mathcal{M}$, so $h(m + n) = h(m) \oplus h(n)$. Finally, $h(m \cdot n) = h(m) \otimes h(n)$ by a similar argument.]

Any number other than zero is the successor of some unique NUMBER, so any NUMBER other than ZERO is the SUCCESSOR of some unique number. So we can define a function ‡ from NUMBERS to NUMBERS by letting $\mathrm{O}^{\ddagger} = \mathrm{O}$ and otherwise letting $m^{\ddagger}$ be the unique NUMBER of which m is the SUCCESSOR. If n is standard, then $n^{\dagger}$ and $n^{\ddagger}$ are standard, too, and if m is nonstandard, then $m^{\dagger}$ and $m^{\ddagger}$ are nonstandard. Moreover, if n is standard and m nonstandard, then n is LESS THAN $m^{\ddagger}$.

We'll now define an equivalence relation $\approx$ on NUMBERS. If a and b are NUMBERS, we'll say that $a \approx b$ if for some *standard*(!) NUMBER c, either $a \oplus c = b$ or $b \oplus c = a$. Intuitively speaking, $a \approx b$ if a and b are a *finite* distance away from each other, or in other words, if one can get from a to b by applying † or ‡ a finite number of times. Every standard NUMBER bears the relation $\approx$ to all and only the standard NUMBERS. We call the equivalence class under $\approx$ of any NUMBER a the *block* of a. Thus a's block is

$$\{\dots, a^{\ddagger\ddagger\ddagger}, a^{\ddagger\ddagger}, a^{\ddagger}, a, a^{\dagger}, a^{\dagger\dagger}, a^{\dagger\dagger\dagger}, \dots\}.$$

Note that a's block is infinite in both directions if a is nonstandard, and is ordered like the integers (negative, zero, and positive).

Suppose that a is LESS THAN b and that a and b are in different blocks. Then since $a^\dagger$ is LESS THAN or equal to b, and a and $a^\dagger$ are in the same block, $a^\dagger$ is LESS THAN b. Similarly, a is LESS THAN $b^\ddagger$. It follows that if there is even one member of a block A that is LESS THAN some member of a block B, then every member of A is LESS THAN every member of B. If this is the case, we'll say that block A is LESS THAN block B. A block is *nonstandard* if and only if it contains some nonstandard number. The standard block is the LEAST block.

There is no LEAST nonstandard block, however. For suppose that b is a nonstandard NUMBER. Then there is an a LESS THAN b such that either $a \oplus a = b$ or $a \oplus a \oplus I = b$. [Why? Because for any (natural) number b greater than zero there is an a less than b such that either $a + a = b$ or $a + a + 1 = b$.] Let's suppose $a \oplus a = b$. (The other case is similar.) If a is standard, so is $a \oplus a$. So a is nonstandard. And a is not in the same block as b: for if $a \oplus c = b$ for some standard c, then $a \oplus c = a \oplus a$, whence $c = a$, contradicting the fact that a is nonstandard. (The laws of addition that hold in $\mathcal{N}$ hold in $\mathcal{M}$.) So a's block is LESS THAN b's block. Similarly, there is no GREATEST block.

Finally, if one block A is LESS THAN another block C, then there is a third block B that A is LESS THAN, and that is LESS THAN C. For suppose a is in A and c is in C, and a is LESS THAN c. There there is an b such that a is LESS THAN b, b is LESS THAN c, and either $a \oplus c = b \oplus b$ or $a \oplus c \oplus I = b \oplus b$. (*Averages*, to within a margin of error of one-half, always exist in $\mathcal{N}$; b is the AVERAGE in $\mathcal{M}$ of a and c.) Suppose $a \oplus c = b \oplus b$. (The argument is similar in the other case.) If b is in A, then $b = a \oplus d$ for some standard d, and so $a \oplus c = a \oplus d \oplus a \oplus d$, and so $c = a \oplus d \oplus d$ (laws of addition), from which it follows, as $d \oplus d$ is standard, that c is in A. So b is not in A, and, similarly not in C either. We may thus take as the desired B the block of b.

To sum up: the elements of the domain of any nonstandard model $\mathcal{M}$ of arithmetic are going to be linearly ordered by LESS THAN. This ordering will have an initial segment that is isomorphic to the usual ordering of natural numbers, followed by a sequence of blocks, each of which is isomorphic to the usual ordering of the integers (negative, zero, and positive). There is neither an earliest nor a latest block, and between any two blocks there lies a third. Thus the ordering of the *blocks* is what was called in the problems at the end of Chapter 12 a *dense linear ordering without endpoints*, and so, as shown there, it is isomorphic to the usual ordering of the rational numbers. This analysis gives us the following result.

25.1*a* Theorem. The order relations on any two enumerable nonstandard models of arithmetic are isomorphic.

Proof: Let K be the set consisting of all natural numbers together with all pairs (q, a) where q is a rational number and a and integer. Let $<_K$ be the order on K in which the natural numbers come first, in their usual order, and the pairs afterward, ordered as follows: $(q, a) <_K (r, b)$ if and only if $q < r$ in the usual order on rational numbers, or ($q = r$ and $a < b$ in the usual order on integers). Then what we have shown above is that the order relation in any enumerable nonstandard model of arithmetic is isomorphic to the ordering $<_K$ of K. Hence the order relations in any two such models are isomorphic to each other.

This result can be extended from models of (true) arithmetic to models of the theory **P** (introduced in Chapter 16).

25.1*b* Theorem. The order relations on any two enumerable nonstandard models of **P** are isomorphic.

Proof: We indicate the proof in outline. What one needs to do in order to extend Theorem 25.1*a* from models of arithmetic to models of **P** is to replace every argument 'S must be true in $\mathcal{M}$ because S is true in $\mathcal{N}$' that occurs above, by the argument 'S must be true in $\mathcal{M}$ because S is a theorem of **P**'. To show that S is indeed a theorem of **P**, one needs to 'formalize' in **P** the ordinary, unformalized mathematical proof that S is true in $\mathcal{N}$. In some cases (for instance, laws of arithmetic) this has been done already in Chapter 16; in the other cases (for instance, the existence of averages) what needs to be done is quite similar to what was done in Chapter 16. Details are left to the reader.

Any enumerable model of arithmetic or **P** (or indeed any theory) is isomorphic to one whose domain is the set of natural numbers. Our interest in the remainder of this chapter will be in the nature of the relations and functions that such a model assigns as denotations to the nonlogical symbols of the language. A first result on this question is a direct consequence of Theorem 25.1*a*.

25.2 Corollary. There is a nonstandard model of arithmetic with domain the natural numbers in which the order relation is a recursive relation (and the successor function a recursive function).

Proof: We know the order relation on any nonstandard model of arithmetic is isomorphic to the order $<_K$ on the set K defined in the proof of Theorem 25.1*a*. The main step in the proof of the corollary will be relegated to the problems at the end of the chapter. It is to show that there is a recursive relation $\prec$ on the natural numbers that is also isomorphic to the order $<_K$ on the set K. Now, given any enumerable nonstandard model $\mathcal{M}$ of arithmetic, there is a function h from the natural numbers to $|\mathcal{M}|$ that is an isomorphism between the ordering $\prec$ on the natural numbers and the ordering $<^{\mathcal{M}}$ on $\mathcal{M}$. Much as in the proof of the canonical-domains lemma (Corollary 12.6), define an operation † on natural numbers by letting $n^\dagger$ be the (unique) m such that $h(m) = h(n)'^{\mathcal{M}}$; and define functions $\oplus$ and $\otimes$ similarly. Then the interpretation with domain the natural numbers and with $\prec, ^\dagger, \oplus, \otimes$ as the denotation of $<, ', +, \cdot$ will be isomorphic to $\mathcal{M}$. It will thus be a model of arithmetic, with the order relation $\prec$ recursive. (If one is careful, one can get the successor function † to be recursive as well.)

The Löwenheim–Skolem theorem tells us that any theory that has an infinite model has a model with domain the natural numbers. The *arithmetical Löwenheim–Skolem theorem* asserts that any *axiomatizable* theory that has an infinite model has a model with domain the natural numbers *and the denotation of every nonlogical symbol an arithmetical relation or function*. The proof of this result requires careful review of the proof of the model existence lemma in Chapter 13. It is outlined in the problems at the end of this chapter. While (true) arithmetic is not an axiomatizable theory, **P** is, and so the arithmetical Löwenheim–Skolem theorem gives us the following.

25.3 Corollary. There is a nonstandard model of **P** with domain the natural numbers in which the denotation of every nonlogical symbol is an arithmetical relation or function.

Proof: As in the proof of the existence of nonstandard models of arithmetic, add a constant ∞ to the language of arithmetic and apply the compactness theorem to the theory

$$\mathbf{P} \cup \{\infty \neq \mathbf{n} \colon n = 0, 1, 2, \dots\}$$

to conclude that it has a model (necessarily infinite, since all models of **P** are). The denotation of ∞ in any such model will be a nonstandard element, guaranteeing that the model is nonstandard. Then apply the arithmetical Löwenheim–Skolem theorem to conclude that the model may be taken to have domain the natural numbers, and the denotations of all nonlogical symbols arithmetical.

The results of the next section contrast sharply with Corollaries 25.2 and 25.3.

25.2 Operations in Nonstandard Models

Our goal in this section is to indicate the proof of two strengthenings of Tennenbaum's theorem to the effect that there is no nonstandard model of **P** with domain the natural numbers in which the addition and multiplication functions are both recursive, along with two analogues of these strengthened results. Specifically, the four results are as follows.

25.4*a* Theorem. There is no nonstandard model of (true) arithmetic with domain the natural numbers in which the addition function is arithmetical.

25.4*b* Theorem (Tennenbaum–Kreisel theorem). There is no nonstandard model of **P** with domain the natural numbers in which the addition function is recursive.

25.4*c* Theorem. There is no nonstandard model of (true) arithmetic with domain the natural numbers in which the multiplication function is arithmetical.

25.4*d* Theorem (Tennenbaum–McAloon theorem). There is no nonstandard model of **P** with domain the natural numbers in which the multiplication function is recursive.

The proof of Theorem 25.4*a* will be given in some detail. The modifications needed to prove Theorem 25.4*b* and those needed to prove Theorem 25.4*c* will both be indicated in outline. A combination of both kind of modifications would be needed for Theorem 25.4*d*, which will not be further discussed.

Throughout the remainder of this section, by *formula* we mean formula and sentence of the language of arithmetic L, and by *model* we mean an interpretation of L with domain the set of natural numbers. For the moment our concern will be with models of (true) arithmetic. Let $\mathcal{M}$ be such a model that is not isomorphic to the standard model $\mathcal{N}$, and let us use $\oplus$ and $\otimes$ as in the preceding section for the denotations it assigns to the addition and multiplication symbols.

A notational preliminary: Our usual notation for the satisfaction in a model $\mathcal{M}$ of a formula $F(x, y)$ by elements a and b of the domain has been $\mathcal{M} \models F[a, b]$. For the remainder of this chapter, rather than write, for instance, 'let $F(x, y)$ be the formula $\exists z\, x = y \cdot z$ and let $\mathcal{M} \models F[a, b]$', we are just going to write 'let $\mathcal{M} \models \exists z\, x = y \cdot z[a, b]$'. (Potentially this briefer notation is ambiguous where there is more than one free variable, since nothing in it explicitly indicates that it is a that goes with x and b with y rather than the other way around; actually, context and alphabetical order should always be sufficient to indicate what is intended.) Thus instead of writing 'Let $F(z)$ be the formula $\mathbf{n} < z$ and suppose $\mathcal{M} \models F[d]$', we just write 'Suppose $\mathcal{M} \models \mathbf{n} < z[d]$'. In this notation, a number d is a nonstandard element of $\mathcal{M}$ if and only if for for every n, $\mathcal{M} \models \mathbf{n} < z[d]$. (If d is nonstandard, $\mathcal{M} \models \mathbf{d} < z[d]$.)

We know from the previous section that nonstandard elements exist. The key to proving Theorem 25.4*a* is a rather surprising result (Lemma 25.7*a* below) asserting the existence of nonstandard elements with special properties. In the statement of this result and the lemmas needed to prove it, we write $\pi(n)$ for the nth prime (counting 2 as the zeroth, 3 as the first, and so on). In order to be able to write about π in the language of arithmetic, fix a formula $\Pi(x, y)$ representing the function π in **Q** (and hence in **P** and in arithemetic), as in section 16.2. Also, abbreviate as $x \mid y$ the rudimentary formula defining the relation 'x divides y'. Here, then, are the key lemmas.

25.5*a* Lemma. Let $\mathcal{M}$ be a nonstandard model of arithmetic. For any $m > 0$,

$$\mathcal{M} \models \forall x\, \mathbf{m} \cdot x = x + \cdots + x \qquad (m\ x\text{s}).$$

25.6*a* Lemma. Let $\mathcal{M}$ be a nonstandard model of arithmetic. Let $A(x)$ be any formula of L. Then there is a nonstandard element d such that

$$\mathcal{M} \models \exists y \forall x < z\, (\exists w(\Pi(x, w) \,\&\, w \mid y) \leftrightarrow A(x))[d].$$

25.7*a* Lemma. Let $\mathcal{M}$ be a nonstandard model of arithmetic. Let $A(x)$ be any formula of L. Then there exists a b such that for every n,

$$\mathcal{M} \models A(\mathbf{n}) \quad \text{if and only if} \quad \text{for some } a, \quad b = a \oplus \cdots \oplus a \ \ [\pi(n)\ a\text{s}].$$

Proof of Lemma 25.5a: The displayed sentence is true in $\mathcal{N}$, and hence is true in $\mathcal{M}$.

Proof of Lemma 25.6a: It is enough to show that the sentence

$$\forall z \exists y \forall x < z\, (\exists w(\Pi(x, w) \,\&\, w \mid y) \leftrightarrow A(x))$$

is true in $\mathcal{N}$, since it must then be true in $\mathcal{M}$. Now what this sentence says, interpreted over $\mathcal{N}$, is just that for every z there exists a positive y such that for all $x < z$, the xth prime divides y if and only if $A(x)$ holds. It is enough to take for y the product of the xth prime for all $x < z$ such that $A(x)$ holds.

Before giving the details of the proof of Lemma 25.7*a*, let us indicate Tennenbaum's main idea. Lemma 25.6*a* can be regarded as saying that for every z there is a y that encodes the answers to all questions $A(x)$? for x less than z. Apply this to a nonstandard element d in $\mathcal{M}$. Then there is a b that encodes the answers to all questions $\mathcal{M} \models A(x)[i]$? for all i LESS THAN d. But since d is nonstandard, the denotations of all numerals are LESS THAN d. So b codes the answers to all the infinitely many questions $\mathcal{M} \models A(\mathbf{n})$? for n a natural number.

Proof of Lemma 25.7a: Let d be as in Lemma 25.6*a*, so we have

$$\mathcal{M} \models \exists y \forall x < z\ (\exists w(\Pi(x, w)\ \&\ w \mid y) \leftrightarrow A(x))[d].$$

Let b be such that

$$\mathcal{M} \models \forall x < z\ (\exists w(\Pi(x, w)\ \&\ w \mid y) \leftrightarrow A(x))[b, d].$$

Since d is nonstandard, for every n, $\mathcal{M} \models \mathbf{n} < z[d]$. Thus we have

$$\mathcal{M} \models (\exists w(\Pi(\mathbf{n}, w)\ \&\ w \mid y) \leftrightarrow A(\mathbf{n}))[b].$$

Since Π represents π, we have

$$\mathcal{M} \models \forall w(\Pi(\mathbf{n}, w) \leftrightarrow w = \mathbf{p}_n).$$

Thus for every n,

$$\mathcal{M} \models \mathbf{p}_n | y \leftrightarrow A(\mathbf{n})[b].$$

That is,

$$\mathcal{M} \models \exists x(\mathbf{p}_n \cdot x = y) \leftrightarrow A(\mathbf{n})[b].$$

It follows that, for every n,

$$\mathcal{M} \models A(\mathbf{n}) \quad \text{if and only if} \quad \text{for some} \quad a, \quad \mathcal{M} \models \mathbf{p}_n \cdot x = y[a, b].$$

By Lemma 25.5*a* this means

$$\mathcal{M} \models A(\mathbf{n}) \quad \text{if and only if} \quad \text{for some} \quad a, \quad b = a \oplus \cdots \oplus a \ \ [\pi(n)\, a\text{s}]$$

as required to complete the proof.

Proof of Theorem 25.4a: Suppose $\oplus$ is arithmetical. Then, since it is obtainable from $\oplus$ by primitive recursion, the function f taking a to $a \oplus \cdots \oplus a(n\ a\text{s})$ is arithmetical; and then, since it is obtainable from f and π by composition, the function g taking a to $a \oplus \cdots \oplus a\ [\pi(n)\ a\text{s}]$ is arithmetical. (The proof in section 16.1 that recursive functions are arithmetical shows that processes of composition and primitive recursion applied to arithmetical functions yield arithmetical functions.) Hence the relation H given by

$$Hbn \quad \text{if and only if} \quad \text{for some } a, \quad b = a \oplus \cdots \oplus a\ [\pi(n)\, a\text{s}]$$

or in other words

$$Hbn \quad \text{if and only if} \quad \exists a\, b = g(a, n)$$

is arithmetical, being obtainable by existential quantification from the graph relation of an arithmetical function.

So let $B(x, y)$ be a formula arithmetically defining H. Let $A(x)$ be the formula $\sim B(x, x)$. Apply Lemma 25.7*a* to obtain a b such that for all n, $\mathcal{M} \models A(\mathbf{n})$ if and only if Hbn. Since the same sentences are true in $\mathcal{M}$ and $\mathcal{N}$, for all n, $\mathcal{N} \models A(\mathbf{n})$ if and only if Hbn. In particular, $\mathcal{N} \models A(\mathbf{b})$ if and only if Hbb, that is, $\mathcal{N} \models \sim B(\mathbf{b}, \mathbf{b})$ if and only if Hbb. But since B arithmetically defines H, we also have $\mathcal{N} \models B(\mathbf{b}, \mathbf{b})$ if and only if Hbb. Contradiction.

For the proof of Theorem 25.4*b*, we need extensions of the lemmas used for Theorem 25.4*a* that will apply not just to models of arithmetic but to models of **P**. We state these as Lemmas 25.5*b* through 25.7*b* below. As in the case of the extension of Theorem 25.1*a* to Theorem 25.1*b*, some 'formalizing' of the kind done in Chapter 16 is needed. What is needed for Lemma 25.6*b*, however, goes well beyond this; so, leaving other details to the reader, we give the proof of that lemma, before going on to give the derivation of Theorem 25.4*b* from the lemmas. The proof of Lemma 25.6*b* itself uses an auxiliary lemma of some interest, Lemma 25.8 below.

25.5*b* Lemma. Let $\mathcal{M}$ be a nonstandard model of **P**. For any $m > 0$,

$$\mathcal{M} \models \forall x\, \mathbf{m} \cdot x = x + \cdots + x (m\, x\text{s}).$$

25.6*b* Lemma. Let $\mathcal{M}$ be a nonstandard model of **P**. Let $A(x)$ be any formula of L. Then there is a nonstandard element d such that

$$\mathcal{M} \models \exists y \forall x < z\, (\exists w(\Pi(x, w) \,\&\, w \mid y) \leftrightarrow A(x))[d].$$

25.7*b* Lemma. Let $\mathcal{M}$ be a nonstandard model of **P**. Let $A(x)$ be any formula of L. Then there exists a b such that for every n,

$$\mathcal{M} \models A(\mathbf{n}) \quad \text{if and only if} \quad \text{for some } a, \quad b = a \oplus \cdots \oplus a \;\; [\pi(n)\, a\text{s}].$$

25.8 Lemma (Overspill principle). Let $\mathcal{M}$ be a nonstandard model of **P**. Let $B(x)$ be any formula of L that is satisfied in $\mathcal{M}$ by all standard elements. Then $B(x)$ is satisfied in $\mathcal{M}$ by some nonstandard element.

Proof of Lemma 25.8: Assume not. Then for any d that satisfies $B(x)$ in $\mathcal{M}$, d is standard, hence $d^\dagger$ is standard, and hence $d^\dagger$ satisfies $B(x)$ in $\mathcal{M}$. Thus

$$\mathcal{M} \models \forall x (B(x) \rightarrow B(x'))$$

since O, being standard, satisfies $B(x)$ in $\mathcal{M}$, $\mathcal{M} \models B(\mathbf{0})$. But also

$$\mathcal{M} \models (B(\mathbf{0}) \,\&\, \forall x(B(x) \rightarrow B(x'))) \rightarrow \forall x B(x)$$

since this is an axiom of **P**. So $\mathcal{M} \models \forall x B(x)$ and *every* element satisfies $B(x)$ in $\mathcal{M}$, contrary to assumption.

Proof of Lemma 25.6b: It is possible to formalize the proof of

$$\forall z \exists y \forall x < z\, (\exists w(\Pi(x, w) \,\&\, w \mid y) \leftrightarrow A(x))$$

in **P**, but to do so would be both extremely tedious and entirely unnecessary, since in view of the preceding lemma it is enough to show that

$$\exists y \forall x < z\,(\exists w(\Pi(x, w)\,\&\, w \mid y) \leftrightarrow A(x))$$

is satisfied by all standard elements, and for this it is enough to show that for every n, the following is a theorem of **P**:

$$(1)\qquad \exists y \forall x < \mathbf{n}\,(\exists w(\Pi(x, w)\,\&\, w \mid y) \leftrightarrow A(x)).$$

Let $n = m + 1$. First recall that the following is a theorem of **P**:

$$(2)\qquad \forall x(x < \mathbf{n} \leftrightarrow (x = \mathbf{0} \vee \cdots \vee x = \mathbf{m})).$$

Since Π represents π, writing p_i for $\pi(i)$, for all $i < n$ the following is a theorem of **P**:

$$(3)\qquad \forall w(\Pi(\mathbf{i}, w) \leftrightarrow w = \mathbf{p}_i).$$

Using (2) and (3), (1) is provably equivalent in **P** to

$$(4)\qquad \exists y((\mathbf{p}_0 \mid y \leftrightarrow A(\mathbf{0}))\,\&\, \ldots\,\&\,(\mathbf{p}_m \mid y \leftrightarrow A(\mathbf{m}))).$$

For each sequence $e = (e_0, \ldots, e_m)$ of length n of 0s and 1s, let A_e be the conjunction of all $(\sim)A(\mathbf{i})$, where the negation sign is present if $e_i = 0$ and absent if $e_i = 1$. Let $B_e(y)$ be the analogous formula with $\mathbf{p}_i \mid y$ in place of $A(\mathbf{i})$. Then the formula after the initial quantifier in (4) is logically equivalent to the disjunction of all conjunctions $A_e\,\&\,B_e(y)$. The existential quantifier may be distributed through the disjunction, and in each disjunct confined to the conjuncts that involve the variable y. Thus (4) is logically equivalent to the disjunction of all conjunctions $A_e\,\&\,\exists y B_e(y)$. Hence (1) is provably equivalent in **P** to this disjunction. But $\exists y B_e(y)$ is a true $\exists$-rudimentary sentence, and so is provable in **P**. Hence (1) is provably equivalent in **P** to the disjunction of all A_e. But this disjunction is logically valid, hence provable in **P** or any theory. So (1) is provable in **P**.

Proof of Theorem 25.4b: We need a fact established in the problems at the end of Chapter 8 (and in a different way in those at the end of Chapter 16): there exist disjoint semirecursive sets A and B such that there is no recursive set containing A and disjoint from B. Since the sets are semirecursive, there are $\exists$-rudimentary formulas $\exists y\alpha(x, y)$ and $\exists y\beta(x, y)$ defining them. Replacing these by

$$\exists y(\alpha(x, y))\,\&\,{\sim}\exists z \leq y\beta(x, y)) \quad \text{and} \quad \exists y(\beta(x, y)\,\&\,{\sim}\exists z \leq y\,\alpha(x, y))$$

we get $\exists$-rudimentary formulas $\alpha^*(x)$ and $\beta^*(x)$ also defining A and B, and for which ${\sim}\exists x(\alpha^*(x)\,\&\,\beta^*(x))$ is a theorem of **P**. If n is in A, then since $\alpha^*(\mathbf{n})$ is $\exists$-rudimentary and true, it is a theorem of **P**, and hence $\mathcal{M} \models \alpha^*(\mathbf{n})$; while if n is in B, then similarly $\beta^*(\mathbf{n})$ is a theorem of **P** and hence so is ${\sim}\alpha^*(\mathbf{n})$, so that $\mathcal{M} \models {\sim}\alpha^*(\mathbf{n})$.

Now by Lemma 25.7b, there are elements b^+ and b^- such that for every n,

$$\mathcal{M} \models \alpha^*(\mathbf{n}) \quad \text{if and only if} \quad \text{for some } a, \quad b^+ = a \oplus \cdots \oplus a \;\; (\pi(n)\; a\text{s})$$
$$\mathcal{M} \models {\sim}\alpha^*(\mathbf{n}) \quad \text{if and only if} \quad \text{for some } a, \quad b^- = a \oplus \cdots \oplus a \;\; (\pi(n)\; a\text{s}).$$

Let Y^+ be $\{n\colon \mathcal{M} \models \alpha^*(\mathbf{n})\}$, and let Y^- be its complement, $\{n\colon \mathcal{M} \models {\sim}\alpha^*(\mathbf{n})\}$. Then we have

$$Y^+ = \{n\colon \text{for some } a, b^+ = a \oplus \cdots \oplus a\ (\pi(n)\ a\text{s})\}.$$

If the function $\oplus$ is recursive, then (much as in the proof of Theorem 25.4*a*) since the function g taking a to $a \oplus \cdots \oplus a$ ($\pi(n)$ as) is obtainable from $\oplus$ by primitive recursion and composition with π, this g is recursive. Since

$$Y^+ = \{n\colon \exists a\, b^+ = g(a, n)\}.$$

Y^+ is semirecursive. A similar argument with b^- in place of b^+ shows that the complement Y^- of Y^+ is also semirecursive, from which it follows that Y^+ is recursive. But this is impossible, since Y^+ contains A and is disjoint from B.

For the proof of Theorem 25.4*c*, we need lemmas analogous to those used for Theorem 25.4*a*, with $\otimes$ in place of $\oplus$. We state these as Lemmas 25.5*c* through 25.7*c* below. These lemmas pertain to exponentiation. Now the notation x^y for exponentiation is not available in L, any more than the notation π for the function enumerating the primes. But we allow ourselves to use that in stating the lemmas, rather than use a more correct but more cumbersome formulation in terms of a formula *representing* the exponential function. We also write $x \downarrow y$ for 'y has an integral xth root' or 'y is the xth power of some integer'. The only real novelty comes in the proof of Lemma 25.6*c*, so we give that proof, leaving other details to the reader.

25.5*c* Lemma. Let $\mathcal{M}$ be a nonstandard model of arithmetic. For any $m > 0$,

$$\mathcal{M} \models \forall x\, x^{\mathbf{m}} = x \cdot \cdots \cdot x\ (m\ x\text{s}).$$

25.6*c* Lemma. Let $\mathcal{M}$ be a nonstandard model of arithmetic. Let $A(x)$ be any formula of L. Then there is a nonstandard element d such that

$$\mathcal{M} \models \exists y \forall x < z\, (\exists w(\Pi(x, w) \,\&\, w \downarrow y) \leftrightarrow A(x))[d].$$

25.7*c* Theorem. Let $\mathcal{M}$ be a nonstandard model of arithmetic. Let $A(x)$ be any formula of L. Then there exists a b such that for every n,

$$\mathcal{M} \models A(\mathbf{n}) \quad \text{if and only if} \quad \text{for some } a, \quad b = a \otimes \cdots \otimes a\ \ (\pi(n)\ a\text{s}).$$

Proof of Lemma 25.6c: It is enough to show that

$$\forall z \exists y \forall x < z\, (\exists w(\Pi(x, w) \,\&\, W\!\downarrow y) \leftrightarrow A(x))$$

is true in $\mathcal{N}$, since it must then be true in $\mathcal{M}$. Recall that we have shown in the proof of Lemma 25.6*b* that

$$\forall z \exists y \forall x < z\, (\exists w(\Pi(x, w) \,\&\, w \mid y) \leftrightarrow A(x))$$

is true in $\mathcal{N}$. It suffices to show, therefore, that the following is true in $\mathcal{N}$:

$$\forall y \exists v \forall w(w \downarrow v \leftrightarrow w \mid y).$$

In fact, given y, 2^y will do for v (unless $y = 0$, in which case $v = 0$ will do). For suppose w divides y, say $y = uw$. Then $2^y = 2^{uw} = (2^u)^w$, and 2^y is a wth power.

And suppose conversely 2^y is a wth power, say $2^y = t^w$. Then t cannot be divisible by any odd prime, and so must be a power of 2, say $t = 2^u$. Then $2^y = (2^u)^w = 2^{uw}$, and $y = uw$, so y is divisible by w.

25.3 Nonstandard Models of Analysis

In the language L^* of arithmetic, under its standard interpretation $\mathcal{N}^*$ (to revert to our former notation), we can directly 'talk about' natural numbers, and can indirectly, through coding, 'talk about' finite sets of natural numbers, integers, rational numbers, and more. We cannot, however, 'talk about' arbitrary sets of natural numbers or objects that might be coded by these, such as real or complex numbers. The *language of analysis* L^{**}, and its standard interpretation $\mathcal{N}^{**}$, let us do so.

This language is an example of a *two-sorted first-order language*. In two-sorted first-order logic there are two sorts of variables: a first sort $x, y, z, \ldots$, which may be called *lower* variables, and a second sort $X, Y, Z, \ldots$, which may be called *upper* variables. For each nonlogical symbol of a two-sorted language, it must be specified not only how many places that symbol has, but also which sorts of variables go into which places. An interpretation of a two-sorted language has two domains, upper and lower. A sentence $\forall x F(x)$ is true in an interpretation if every element of the lower domain satisfies $F(x)$, while a sentence $\forall X G(X)$ is true if every element of the upper domain satisfies $G(X)$. Otherwise the definitions of language, sentence, formula, interpretation, truth, satisfaction, and so forth are unchanged from ordinary or one-sorted first-order logic.

An isomorphism between two interpretations of a two-sorted language consists of a *pair* of correspondences, one between the lower domains and the other between the upper domains of the two interpretations. The proof of the isomorphism lemma (Proposition 12.5) goes through for two-sorted first-order logic, and so do the proofs of more substantial results such as the compactness theorem and the Löwenheim–Skolem theorem (including the strong Löwenheim–Skolem theorem of Chapter 19). Note that in the Löwenheim–Skolem theorem, an interpretation of a two-sorted language counts as enumerable only if *both* its domains are enumerable.

In the language of analysis L^{**} the nonlogical symbols are those of L^*, which take only lower variables, plus a further two-place predicate $\in$, which takes a lower variable in its first place but an upper in its second. Thus $x \in Y$ is an atomic formula, but $x \in y$, $X \in Y$, and $X \in y$ are not. In the standard interpretation $\mathcal{N}^{**}$ of L^*, the lower domain is the set of natural numbers and the interpretation of each symbol of L is the same as in the standard interpretation $\mathcal{N}^*$ of L. The upper domain is the class of all sets of natural numbers, and the interpretation of $\in$ is the membership or elementhood relation $\in$ between numbers and sets of numbers. As (true) arithmetic is the set of sentences of L^* true in $\mathcal{N}^*$, so *(true) analysis* is the set of all sentences of L^{**} true in $\mathcal{N}^{**}$. A model of analysis is *nonstandard* if it is not isomorphic to $\mathcal{N}^{**}$. Our aim in this section is to gain some understanding of nonstandard models of (true) analysis and some important subtheories thereof.

By the *lower part* of an interpretation of L^{**}, we mean the interpretation of L^* whose domain is the lower domain of the given interpretation, and that assigns to each nonlogical symbol of $L*$ the same denotation as does the given interpretation. Thus the lower part of $\mathcal{N}**$ is $\mathcal{N}*$. A sentence of $L*$ will be true in an interpretation of $L**$ if and only if it is true in the lower part of that interpretation. Thus a sentence of $L*$ is a theorem of (that is, is in) true arithmetic if and only if it is a theorem of true analysis.

Our first aim in this section will be to establish the existence of nonstandard models of analysis of two distinct kinds. An interpretation of $L**$ is called an $\in$-*model* if (as in the standard interpretation) the elements of the upper domain are sets of elements of the lower domain, and the interpretation of $\in$ is the membership or elementhood relation $\in$ (between elements of the lower and the upper domain). The sentence

$$\forall X \forall Y(\forall x(x{\in}X \leftrightarrow x{\in}Y) \to X = Y)$$

is called the axiom of *extensionality*. Clearly it is true in any $\in$-model and hence in any model isomorphic to an $\in$-model. Conversely, any model $\mathcal{M}$ of extensionality is isomorphic to an $\in$-model $\mathcal{M}^{\#}$. [To obtain $\mathcal{M}^{\#}$ from $\mathcal{M}$, keep the same lower domain and the same interpretations for symbols of $L*$, replace each element α of the upper domain of $\mathcal{M}$ by the set $\alpha^{\#}$ of all elements a of the lower domain such that $a \in^{\mathcal{M}} \alpha$, and interpret $\in$ not as the relation $\in^{\mathcal{M}}$ but as $\in$. The identity function on the lower domain together with the function sending α to $\alpha^{\#}$ is an isomorphism. The only point that may not be immediately obvious is that the latter function is one-to-one. To see this, note that if $\alpha^{\#} = \beta^{\#}$, then α and β satisfy $\forall x(x \in X \leftrightarrow x \in Y)$ in $\mathcal{M}$, and since (2) is true in $\mathcal{M}$, α and β must satisfy $X = Y$, that is, we must have $\alpha = \beta$.] Since we are going to be interested only in models of extensionality, we may restrict our attention to $\in$-models.

If the lower part of an $\in$-model $\mathcal{M}$ is the standard model of arithmetic, we call $\mathcal{M}$ an ω-model. The standard model of analysis is, of course, an ω-model. If an ω-model of analysis is nonstandard, its upper domain must consist of some class of sets *properly* contained in the class of *all* sets of numbers. If the lower part of an $\in$-model $\mathcal{M}$ is isomorphic to the standard interpretation $\mathcal{N}*$ of $L*$, then $\mathcal{M}$ as a whole is isomorphic to an ω-model $\mathcal{M}^{\#}$. [If j is the isomorphism from $\mathcal{N}*$ to the lower part of $\mathcal{M}$, replace each element α of the upper domain of $\mathcal{M}$ by the set of n such that $j(n) \in \alpha$, to obtain $\mathcal{M}^{\#}$.] So we may restrict our attention to models that are of one of two kinds, namely, those that either are ω-models, or have a nonstandard lower part.

Our first result is that nonstandard models of analysis of both kinds exist.

25.9 Proposition. Both nonstandard models of analysis whose lower part is a nonstandard model of arithmetic and nonstandard ω-models of analysis exist.

Proof: The existence of nonstandard models of arithmetic was established in the problems at the end of Chapter 12 by applying the compactness theorem to the theory that results upon adding to arithmetic a constant ∞ and the sentences $\infty \neq \mathbf{n}$ for all natural numbers n. The same proof, with analysis in place of arithmetic, establishes the existence of a nonstandard model of analysis whose lower parts is a nonstandard model of arithmetic. The strong Löwenheim–Skolem theorem implies the existence of an enumerable subinterpretation of the standard model of analysis that is itself a model of analysis. This must be an ω-model, but it cannot be isomorphic to the standard model, whose upper domain is nonenumerable.

The axiomatizable theory in L^* to which logicians have devoted the most attention is **P**, which consists of the sentences deducible from the following axioms:

(0) The finitely many axioms of **Q**

(1) For each formula $F(x)$ of L^*, the sentence

$$(F(\mathbf{0})\,\&\,\forall x(F(x) \rightarrow F(x'))) \rightarrow \forall x F(x).$$

It is to be understood that in (1) there may be other free variables $u, v, \ldots$ present, and that what is really meant by the displayed expression is the universal closure

$$\forall u \forall v \cdots (F(\mathbf{0}, u, v, \ldots)\,\&\,\forall x(F(x, u, v, \ldots) \rightarrow F(x', u, v, \ldots)) \\ \rightarrow \forall x F(x, u, v, \ldots)).$$

The sentence in (1) is called the *induction axiom* for $F(x)$.

The axiomatizable theory in L^{**} to which logicians have devoted the most attention is the theory **P**** consisting of the sentences deducible from the following axioms:

(0) The finitely many axioms of **Q**

(1*) $\forall X(\mathbf{0} \in X\,\&\,\forall x(x \in X \rightarrow x' \in X) \rightarrow \forall x\, x \in X)$

(2) $\forall X \forall Y(\forall x(x \in X \leftrightarrow x \in Y) \rightarrow X = Y)$

(3) For each formula $F(x)$ of L^*, the sentence

$$\exists X \forall x(x{\in}X \leftrightarrow F(x)).$$

It is to be understood that in (3) there may be other free variables $u, v, \ldots$ and/or $U, V, \ldots$ present, and that what is really meant by the displayed expression is the universal closure

$$\forall u \forall v \cdots \forall U \forall V \cdots \exists X \forall x(x \in X \leftrightarrow F(x, u, v, \ldots, U, V, \ldots)).$$

The sentence (1*) is called the *induction axiom* of **P****, the extensionality axiom (2) has already been encountered, and the sentence (3) is called the *comprehension axiom* for $F(x)$. We call **P**** *axiomatic analysis*.

Since the set of theorems of (true) arithmetic is not arithmetical, the set of theorems of (true) analysis is not arithmetical, and *a fortiori* is not semirecursive. By contrast, the set of theorems of axiomatic analysis **P**** is, like the set of theorems of any axiomatizable theory, semirecursive. There must be many theorems of (true) analysis that are not theorems of axiomatic analysis, and indeed (since the Gödel theorems apply to **P****), among these are the Gödel and Rosser sentences of **P****, and the consistency sentence for **P****.

Note that the induction axiom (1) of **P** for $F(x)$ follows immediately from the induction axiom (1) of **P**** together with the comprehension axiom (3) for $F(x)$. Thus every theorem of **P** is a theorem of **P****, and the lower part of any model of **P**** is a model of **P**. We say a model of **P** is *expandable* to a model of **P**** if it is the lower part of a model of **P****. Our second result is to establish the *non*existence of certain kinds of nonstandard models of **P****.

25.10 Proposition

Not every model of **P** can be expanded to a model of **P****.

Proof: We are not going to give a full proof, but let us indicate the main idea. Any model of **P** that can be expanded to a model of **P**** must be a model of every sentence of $L*$ that is a theorem of **P****. Let A be the consistency sentence for **P** (or the Gödel or Rosser sentence). Then A is not a theorem of **P**, and so there is a model of $\mathbf{P} \cup \{\sim A\}$. We claim such a model cannot be expanded to a model of **P****, because A is provable in **P****. The most simple-minded proof of the consistency of **P** is just this: every axiom of **P** is true, only truths are deducible from truths, and $\mathbf{0} = \mathbf{1}$ is not true; hence $\mathbf{0} = \mathbf{1}$ is not deducible from **P**. In section 23.1 we in effect produced a formula $F(X)$ of $L**$ which is satisfied in the standard model of analysis by and only by the set code numbers of sentences of $L*$ that are true in the lower part of that model (that is, in the standard model of arithmetic). Working in **P****, we can introduce the abbreviation True(x) for $\exists X(F(X) \,\&\, x \in X)$, and 'formalize' the simple-minded argument just indicated. (The work of 'formalization' required, which we are omitting, is extensive, though not so extensive as would be required for a complete proof of the second incompleteness theorem.)

Recall that if a language L_1 is contained in a language L_2, a theory T_1 in L_1 is contained in a theory T_2 in L_2, then T_2 is called a *conservative extension* of T_1 if and only if every sentence of L_1 that is a theorem of T_2 is a theorem of T_1. What is shown in the proof indicated for the preceding proposition is, in this terminology, that **P**** is not a conservative extension of **P**.

A weaker variant $\mathbf{P}^+$ allows the comprehension axioms (3) *only for formulas* $F(x)$ *not involving bound upper variables*. [There may still be, in addition to free lower variables $u, v, \ldots,$ *free* upper variables $U, V, \ldots$ in $F(X)$.] $\mathbf{P}^+$ is called *(strictly) predicative analysis*. When one specifies a set by specifying a condition that is necessary and sufficient for an object to belong to the set, the specification is called *impredicative* if the condition involves quantification over sets. Predicative analysis does not allow impredicative specifications of sets. In ordinary, unformalized mathematical argument, impredicative specifications of sets of numbers are comparatively common: for instance, in the first section of the next chapter, an ordinary, unformalized mathematical proof of a principle about sets of natural numbers called the 'infinitary Ramsey's theorem' will be presented that is a typical example of a proof that can be 'formalized' in **P**** but not in $\mathbf{P}^+$.

An innocent-looking instance of impredicative specification of a set is implicitly involved whenever we define a set S of numbers as the union $S_0 \cup S_1 \cup S_2 \cup \cdots$ of a sequence of sets that is defined inductively. In an inductive definition, we specify a condition $F^0(u)$ such that u belongs to S_0 if and only if $F^0(u)$ holds, and specify a condition $F'(u, U)$ such that for all i, u belongs to S_{i+1} if and only if $F'(u, S_i)$ holds. Such an inductive definition can be turned into a direct definition, since $x \in S$ if and only if

there exists a finite sequence of sets $U_0, \ldots, U_n$ such that
    for all u, $u \in U_0$, if and only if $F^0(u)$
        for all $i < n$, for all u, $u \in U_{i+1}$ if and only if $F'(u, U_i)$
    $x \in U_n$.

But while the quantification 'there exists a finite sequence of sets' can by suitable coding be replaced by a quantification 'there exists a set', in general the latter quantification cannot be eliminated. The inductive definition implicitly involves—what the corresponding direct definition explicitly involves—an impredicative specification of a set. In general, one cannot 'formalize' in $\mathbf{P}^+$ arguments involving this kind of inductive specification of sets, *even if* the conditions F^0 and F' involve no bound upper variables.

Also, one cannot 'formalize' in $\mathbf{P}^+$ the proof of the consistency sentence for **P** indicated in the proof of the preceding proposition. [One can indeed introduce the abbreviation True(x) for $\exists X(F(X) \,\&\, x \in X)$, but one cannot in **P** prove the existence of $\{x\colon \text{True}(x)\}$, and so cannot apply the induction axiom to prove assertions involving the abbreviation True(x).] So the proof indicated for the preceding proposition fails for $\mathbf{P}^+$ in place of $\mathbf{P}^*$. In fact, not only is the consistency sentence for **P** not an example of a sentence of L^* that is a theorem of $\mathbf{P}^+$ and not of **P**, but actually there can be *no* example of such sentence: $\mathbf{P}^+$ is a conservative extension of **P**.

Our last result is a proposition immediately implying the fact just stated.

25.11 Proposition. Every model of **P** can be expanded to a model of $\mathbf{P}^+$.

Proof: Let $\mathcal{M}$ be a model of **P**. Call a subset S of the domain $|\mathcal{M}|$ *parametrically definable* over $\mathcal{M}$ if there exist a formula $F(x, y_1, \ldots, y_m)$ of L^* and elements $a_1, \ldots, a_m$ of $|\mathcal{M}|$ such that

$$S = \{b\colon \mathcal{M} \models F[b, a_1, \ldots, a_m]\}.$$

Expand $\mathcal{M}$ to an interpretation of L^{**} by taking as upper domain the class of all parametrically definable subsets of $\mathcal{M}$, and interpreting $\in$ as $\in$.We claim the expanded model $\mathcal{M}^+$ is a model of $\mathbf{P}^+$. The axioms that need checking are induction (1) and comprehension (3) (with F having no bound upper variables). Leaving the former to the reader, we consider an instance of the latter:

$$\forall u_1 \forall u_2 \forall U_1 \forall U_2 \exists X \forall x(x \in X \leftrightarrow F(x, u_1, u_2, U_1, U_2)).$$

(In general, there could be more than two us and more than two Us, but the proof would be no different.) To show the displayed axiom is true in $\mathcal{M}^+$, we need to show that for any elements s_1, s_2 of $|\mathcal{M}|$ and any parametrically definable subsets S_1, S_2 of $|\mathcal{M}|$ there is a parametrically definable subset T of $|\mathcal{M}|$ such that

$$\mathcal{M}^+ \models \forall x(x \in X \leftrightarrow F(x, u_1, u_2, U_1, U_2))[s_1, s_2, S_1, S_2, T].$$

Equivalently, what we must show is that for any such s_1, s_2, S_1, S_2, the set

$$T = \{b\colon \mathcal{M}^+ \models F(x, u_1, u_2, U_1, U_2)[s_1, s_2, S_1, S_2, b]\}$$

is parametrically definable. To this end, consider parametric definitions of U_1, U_2:

$$U_1 = \{b\colon \mathcal{M} \models G_1[b, a_{11}, a_{12}]\}$$

$$U_2 = \{b\colon \mathcal{M} \models G_2[b, a_{21}, a_{22}]\}.$$

(In general, there could be more than two as for each U, but the proof would be no different.) Now let

$$H(x, u_1, u_2, v_{11}, v_{12}, v_{21}, v_{22})$$

be the result of replacing any subformula of form $U_i(w)$ by $G_i(w, v_{i1}, v_{i2})$. Then

$$T = \{b: \mathcal{M} \models H[b, s_1, s_2, a_{11}, a_{12}, a_{21}, a_{22}]\}$$

and is parametrically definable as required.

Problems

25.1 Show how the proof of the existence of averages can be formalized in **P**, in the style of Chapter 16.

25.2 Show that there is a recursive relation $\prec$ on the natural numbers that is also isomorphic to the order $<_K$ on the set K defined in the proof of Theorem 25.1.

25.3 Show that the successor function † associated with $\prec$ may also be taken to be recursive.

25.4 Show that in an $\in$-model that is not an ω-model, the upper domain cannot contain *all* subsets of the lower domain.

The remaining problems outline the proof of the arithmetical Löwenheim–Skolem theorem, and refer to the alternative proof of the model existence lemma in section 13.5 *and the problems following it.*

25.5 Assuming Church's thesis, explain why, if Γ is a recursive set of (code numbers of) sentences in a recursive language, the set Γ^* obtained by adding (the code numbers of) the Henkin sentences to Γ is still recursive (assuming a suitable coding of the language with the Henkin constants added).

25.6 Explain why, if Δ is an arithmetical set of sentences, then the relation

i codes a finite set of sentences Θ,
j codes a sentence D,
and $\Delta \cup \Theta$ implies D

is also arithmetical.

25.7 Suppose Γ^* is a set of sentences in a language L^* and $i_0, i_1, \ldots$ an enumeration of all the sentences of L^*, and suppose we form $\Gamma^\#$ as the union of sets Γ_n, where $\Gamma_0 = \Gamma^*$ and $\Gamma_{n+1} = \Gamma_n$ if Γ_n implies $\sim i_n$, while $\Gamma_{n+1} = \Gamma_n \cup \{i_n\}$ otherwise. Explain why, if Γ^* is arithmetical, then $\Gamma^\#$ is arithmetical.

25.8 Suppose we have a language with relation symbols and enumerably many constants $c_0, c_1, \ldots$, but function symbols and identity are absent. Suppose $\Gamma^\#$ is arithmetical and has the closure properties required for the construction of section 13.2. In that construction take as the element $c_i^{\mathcal{M}}$ associated with the constant c_i the number i. Explain why the relation $R^{\mathcal{M}}$ associated with any relation symbol R will then be arithmetical.

25.9 Suppose we have a language with relation symbols and enumerably many constants $c_0, c_1, \ldots$, but that function symbols are absent, though identity may be present. Suppose $\Gamma^\#$ is arithmetical has the closure properties required for the construction of section 13.3. Call i *minimal* if there is no $j < i$ such that $c_i = c_j$ is in $\Gamma^{\#}$. Show that the function δ taking n to the nth number i such that c_i is minimal is arithmetical.

25.10 Continuing the preceding problem, explain why for every constant c there is a unique n such that $c = \delta(n)$ is in $\Gamma^{\#}$, and that if in the construction of section 13.3 we take as the element $c_i^{\mathcal{M}}$ associated with the constant c_i this number n, then the relation $R^{\mathcal{M}}$ associated with any relation symbol R will be arithmetical.

25.11 Explain how the arithmetical Löwenheim–Skolem theorem for the case where function symbols are absent follows on putting together the preceding six problems, and indicate how to extend the theorem to the case where they are present.

26

Ramsey's Theorem

Ramsey's theorem is a combinatorial result about finite sets with a proof that has interesting logical features. To prove this result about finite sets, we are first going to prove, in section 26.1, an analogous result about infinite sets, and are then going to derive, in section 26.2, the finite result from the infinite result. The derivation will be an application of the compactness theorem. Nothing in the proof of Ramsey's theorem to be presented requires familiarity with logic beyond the statement of the compactness theorem, but at the end of the chapter we indicate how Ramsey theory provides an example of a sentence undecidable in **P** *that is more natural mathematically than any we have encountered so far.*

26.1 Ramsey's Theorem: Finitary and Infinitary

There is an old puzzle about a party attended by six persons, at which any two of the six either like each other or dislike each other: the problem is to show that at the party there are three persons, any two of whom like each other, or there are three persons, any two of whom dislike each other.

The solution: Let a be one of the six. Since there are five others, either there will be (at least) three others that a likes or there will be three others that a dislikes. Suppose a likes them. (The argument is similar if a dislikes them.) Call the three b, c, d. Then if (case 1) b likes c or b likes d or c likes d, then a, b, and c, or a, b, and d, or a, c, and d, respectively, are three persons any two of whom like each other; but if (case 2) b dislikes c, b dislikes d, and c dislikes d, then b, c, and d are three persons, any two of whom dislike each other. And either case 1 or case 2 must hold.

The number six cannot in general be reduced; if only five persons, a, b, c, d, e are present, then the situation illustrated in Figure 26-1 can arise. (A broken line means 'likes'; a solid line, 'dislikes'.) In this situation there are no three of a, b, c, d, e any two of whom like each other (a 'clique') and no three, any two of whom dislike each other (an 'anticlique').

A harder puzzle of the same type is to prove that at any party such as the previous one at which eighteen persons are present, either there are four persons, any two of whom like each other, or four persons, any two of whom dislike each other. (This puzzle has been placed among the problems at the end of this chapter.) It is known that the number eighteen cannot be reduced.

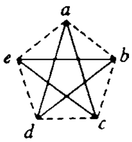


Figure 26-1. A party with no clique or anticlique of three.

We are going to prove a theorem that bears on these puzzles. Recall that by a *partition* of a nonempty set we mean a family of nonempty subsets thereof, called the *classes* of the partition, such that every element of the original set belongs to exactly one of these classes. By a *size-k set* we mean a *set with exactly k elements*.

26.1 Theorem (Ramsey's theorem). Let r, s, n be positive integers with $n \geq r$. Then there exists a positive integer $m \geq n$ such that for $X = \{0, 1, \ldots, m-1\}$, no matter how the size-r subsets of X are partitioned into s classes, there will always be a size-n subset Y of X such that all size-r subsets of Y belong to the same class.

A set Y all of whose size-r subsets belong to the same one of the s classes is called a *homogeneous set* for the partition. Note that if the theorem holds as stated, then it clearly holds for any other size-m set in place of $\{0, 1, \ldots, m-1\}$.

For instance, it holds for the set of partiers at a party where m persons are present. In the puzzles, the size-2 subsets of the set of persons at the party were partitioned into two classes, one consisting of the pairs of persons who like each other, the other, of the pairs of persons who dislike each other. So in both problems $r = s = 2$. In the first, where $n = 3$, we showed how to prove that $m = 6$ is large enough to guarantee the existence of a homogeneous set of size n—a clique of three who like each other, or an anticlique of three who dislike each other. We also showed that 6 is the *least* number m that is large enough. In the second problem, where $n = 4$, we reported that $m = 18$ is large enough, and that 18 is in fact the *least* value of m that is large enough.

In principle, since there are only finitely many size-r subsets of $\{0, \ldots, m-1\}$, and only finitely many ways to partition these finitely many subsets into s classes, and since there are only finitely many size-n subsets, we could set a computer to work searching through all partitions, and for each looking for a homogeneous set. If some partition were found without a homogeneous set, the computer could go on to do a similar check for $\{0, \ldots, m\}$. Continuing in this way, in a finite amount of time it would find the least m that is large enough to guarantee the existence of the required homogeneous set.

In practice, the numbers of possibilities to be checked are so large that such a procedure is hopelessly infeasible. We do not at present have sufficient theoretical insight into the problem to be able to reduce the number of possibilities that would have to be checked to the point where a computer could feasibly be employed in surveying them in order to pinpoint the least m. And it is entirely conceivable that because of the such physical limitations as those imposed by the speed of light, the atomic character of matter, and the short amount of time before the universe becomes unable to sustain life, we are never going to know exact what the value of the least m is, even for some quite small values of r, s, and n.

So let us set aside the difficult problem of finding the *least m* that is large enough, and turn to proving that there is *some m* that is large enough. The proof of Theorem 26.1 that we are going to present will make a 'detour through the infinite'. First we prove the following *infinitary* analogue:

26.2 Theorem (Infinitary Ramsey's theorem). Let r, s be positive integers. Then no matter how the size-r subsets of the set $X = \{0, 1, 2, \dots\}$ are partitioned into s classes, there will always be an infinite subset Y of X such that all size-r subsets of Y belong to the same class.

Note that if the theorem holds as stated, then it clearly holds for any other enumerably infinite set in place of $\{0, 1, 2, \dots\}$. (If Zeus threw a party for an enumerable infinity of guests, any two of whom either liked each other or disliked each other, there would either be infinitely many guests, any two of whom liked each other, or infinitely many, any two of whom disliked each other.) In fact Theorem 26.2 holds for any infinite set X, because any such set has an enumerably infinite subset (though it requires the axiom of choice to prove this, and we are not going to go into the matter).

The proof of Theorem 26.2 will be given in this section, and the derivation of Theorem 26.1 from it—which will involve an interesting application of the compactness theorem—in the next. Before launching into the proof, let us introduce some notation that will be useful for both proofs.

A partition of a set Z into s classes may be represented by a function f whose arguments are the elements of Z and whose values are elements of $\{1, \dots, s\}$: the ith class in the partition is just the set of those z in Z with $f(z) = i$. Let us write $f : Z \to W$ to indicate that f is a function whose arguments are the elements of Z and whose values are elements of W. Our interest is in the case where Z is the collection of all the size-r subsets of some set X. Let us denote this collection $[X]^r$. Finally, let us write ω for the set of natural numbers. Then the infintary version of Ramsey's theorem may be restated as follows: If $f : [\omega]^r \to \{1, \dots, s\}$, then there is an infinite subset Y of ω and a j with $1 \leq j \leq s$ such that $f : [Y]^r \to \{j\}$ (that is, f takes the value j for any size-r subset of Y as argument).

Proof of Theorem 26.2: Our proof will proceed as follows. For any fixed $s > 0$, we show by induction on r that for any $r > 0$ we can define an operation Φ such that if $f : [\omega]^r \to \{1, \dots, s\}$, then $\Phi(f)$ is a pair (j, Y) with $f : [Y]^r \to \{j\}$.

Basis step: $r = 1$. In this case the definition of $\Phi(f) = (j, Y)$ is easy. For each of the infinitely many size-1 sets $\{b\}$, $f(\{b\})$ is one of the finitely many positive integers $k \leq s$. We can thus define j as the least $k \leq s$ such that $f(\{b\}) = k$ for infinitely many b, and define Y as $\{b : f(\{b\}) = j\}$.

Induction step: We assume as induction hypothesis that Φ has been suitably defined for all $g : [\omega]^r \to \{1, \dots, s\}$. Suppose $f : [\omega]^{r+1} \to \{1, \dots, s\}$. In order to define $\Phi(f) = (j, Y)$, we define, for each natural number i, a natural number b_i, infinite sets Y_i, Z_i, W_i, a function $f_i : [\omega]^r \to \{1, \dots, s\}$, and a positive integer $j_i \leq s$. Let $Y_0 = \omega$. We now suppose Y_i has been defined, and show how to define b_i, Z_i, f_i, j_i, W_i, and Y_{i+1}.

Let b_i be the least member of Y_i.

Let $Z_i = Y_i - \{b_i\}$. Since Y_i is infinite, so is Z_i. Let the members of Z_i in increasing order be $a_{i0}, a_{i1}, \ldots$.

For any size-r set x of natural numbers, where $x = \{k_1, \ldots, k_r\}$, with $k_1 < \cdots < k_r$, let $f_i(x) = f(\{b_i, a_{ik_1}, \ldots, a_{ik_r}\})$. Since b_i is not one of the a_{ik} and f is defined on all size-$(r+1)$ sets of natural numbers, f_i is well defined.

By the induction hypothesis, for some positive integer $j_i \leq s$ and some infinite set W_i, $\Phi(f_i) = (j_i, W_i)$ and for every size-r subset x of W_i, we have $f_i(x) = j_i$. We have thus defined j_i and W_i, and we define $Y_{i+1} = \{a_{ik} : k \in W_i\}$.

Since W_i is infinite, Y_{i+1} is infinite. $Y_{i+1} \subseteq Z_i \subseteq Y_i$, and thus if $i_1 \leq i_2$, then $Y_{i_2} \subseteq Y_{i_1}$. And since b_i is less than every member of Y_{i+1}, we have $b_i < b_{i+1}$, which is the least member of Y_{i+1}. Thus if $i_1 < i_2$ then $b_{i_1} < b_{i_2}$.

For each positive integer $k \leq s$, let $E_k = \{i\colon j_i = k\}$. As in the basis step, some E_k is infinite, and we let j be the least k such that E_k is infinite, and let $Y = \{b_i\colon i \in E_j\}$. This completes the definition of Φ.

Since $b_{i_1} < b_{i_2}$ if $i_1 < i_2$, Y is infinite. In order to complete the proof, we must show that if y is a size-$(r+1)$ subset of Y, then $f(y) = j$. So suppose that $y = \{b_i, b_{i_1}, \ldots, b_{i_r}\}$, with $i < i_1 < \cdots < i_r$ and $i, i_1, \ldots, i_r$ all in E_j. Since the Y_i are nested, all of $b_{i_1}, \ldots, b_{i_r}$ are in Y_i. For each m, $1 \leq m \leq r$, let k_m be the unique member of W_i such that $b_{i_m} = a_{ik_m}$. And let $x = \{k_1, \ldots, k_r\}$. Then x is a subset of W_i, and since $i_1 < \cdots < i_r$, we have $b_{i_1} < \cdots < b_{i_r}$, $a_{ik_1} < \cdots < a_{ik_r}$, and $k_1 < \cdots < k_r$, and thus x is a size-r subset of W_i. But $\Phi(f_i) = (j_i, W_i)$ and thus $f_i(x) = j_i$. Since i is in E_j, $j_i = j$. Thus

$$f(y) = f\big(\{b_i, b_{i_1}, \ldots, b_{i_r}\}\big) = f\big(\{b_i, a_{ik_1}, \ldots, a_{ik_r}\}\big) = f_i(x) = j$$

as required.

Before moving on to the next section and the proof of Theorem 26.3, let us point out that the following strengthening of Theorem 26.2 is simply false: Let s be a positive integer. Then no matter how the finite sets of natural numbers are partitioned into s classes, there will always be an infinite set Y of natural numbers such that all positive integers r, all size-r subsets of Y belong to the same one of the s classes. Indeed, this fails for $s = 2$. Let $f(x) = 1$ if the finite set x contains the number that is the number of members in x; and $f(x) = 2$ otherwise. Then there is no infinite set Y such that for every r, either $f(y) = 1$ for all size-r subsets y of Y or $f(y) = 2$ for all such y. For if r is a positive integer that belongs to Y and $b_1, \ldots, b_r$ are r other members of Y, then $f(\{r, b_2, \ldots, b_r\}) = 1$, while $f(\{b_1, \ldots, b_r\}) = 2$.

26.2 König's Lemma

In order to derive the original, finitary version of Ramsey's theorem from the infinitary version, we will establish a principle known as *König's lemma*, concerning objects called trees. For present purposes, a *tree* consists of: (i) a nonempty set T of elements, called the *nodes* of the tree; (ii) a partition of T into finitely or infinitely many sets

$$T = T_0 \cup T_1 \cup T_2 \cup \cdots$$

called the *levels* of the tree; and (iii) a two-place relation R subject to the following conditions:

(1) Rab never holds for b in T_0.
(2) For b in T_{n+1}, Rab holds for exactly one a, and that a is in T_n.

When Rab holds, we say a is immediately *below* b, and b is immediately *above* a.

Figure 26-2 is a picture of a finite tree with ten nodes and four levels. Line segments connect nodes immediately below and above each other.

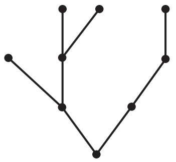


Figure 26-2. A finite tree.

A *branch* through a tree is a sequence of nodes $b_0, b_1, b_2, \ldots$ with each b_n immediately below b_{n+1}. Obviously, an infinite tree none of whose levels is infinite must have infinitely many nonempty levels. The following is not so obvious.

26.3 Lemma (König's lemma). An infinite tree none of whose levels is infinite must have an infinite branch.

Postponing the proof of this result, let us see how it can be used as a bridge between the finite and the infinite.

Proof of Theorem 26.1: Suppose that Theorem 26.1 fails. Then for some positive integers r, s, n, with $n \geq r$, for every $m \geq n$ there exists a partition

$$f : [\{0, 1, \ldots, m-1\}]^r \rightarrow \{1, \ldots, s\}$$

having no size-n homogeneous set Y. Let T be the set of all such partitions without size-n homogeneous sets for all m, and let T_k be the subset of T consisting of those f with $m = n + k$. Let Rfg hold if and only if for some k

$$f : [\{0, 1, \ldots, n+k-1\}]^r \rightarrow \{1, \ldots, s\}$$
$$g : [\{0, 1, \ldots, n+k\}]^r \rightarrow \{1, \ldots, s\}$$

and g *extends* f, in the sense that g assigns the same value as does f to any argument in the domain of f. It is easily seen that for any g in T_{k+1} there is exactly one f in T_k that g extends, so what we have defined is a tree.

There are only finitely many functions from a given finite set to a given finite set, so there are only finitely many nodes f in any level T_k. But our initial supposition was that for every $m = n + k$ there exists a partition f in T_k, so the level T_k is nonempty for all k, and the tree is infinite. König's lemma then tells us there will be an infinite branch $f_0, f_1, f_2, \ldots$, which is to say, an infinite sequence of partitions, each extending the one before, and none having a size-n homogenous set. We can then define a partition

$$F : [\omega]^r \rightarrow \{1, \ldots, s\}$$

as follows. For any size-r subset x of ω, let p be its largest element. Then for any k large enough that $p < n + k$, x will be in the domain of f_k, and we have

$$f_k(x) = f_{k+1}(x) = f_{k+2}(x) = \cdots.$$

Let $F(x)$ be this common value.

By the infinitary version of Ramsey's theorem, F has an infinite homogeneous set. That is, there is an infinite Y and a j with $1 \leq j \leq s$ such that F: $[Y]^r \rightarrow \{j\}$. Let Z be the set of the first n elements of Y, and take k large enough that the largest element of Z is less than $n + k$. Then Z will be a size-n subset of $\{0, \ldots, n + k - 1\}$, with $f_k(x) = F(x) = j$ for all size-r subsets x of Z. In other words, Z will be a size-n homogeneous set for f_k, which is impossible, since f_k is in T. This contradiction completes the proof.

Proof of Lemma 26.3: To prove König's lemma we are going to use the compactness theorem. Let L_T be the language with one one-place predicate **B** and with one constant **t** for each node t in the tree T. Let Γ consist of the following quantifier-free sentences:

(1) $$\mathbf{Bs}_1 \vee \cdots \vee \mathbf{Bs}_k$$

where $s_1, \ldots, s_k$ are all the nodes in T_0;

(2) $$\sim(\mathbf{Bs}\ \&\ \mathbf{Bt})$$

for all pairs of nodes s, t belonging to the same level; and

(3) $$\sim\mathbf{Bs} \vee \mathbf{Bu}_1 \vee \cdots \vee \mathbf{Bu}_m$$

for every node s, where $u_1, \ldots, u_m$ are all the nodes immediately above s. [If there are no nodes above s, the sentence (3) is just $\sim$**Bs**.]

We first show that if Γ has a model $\mathcal{M}$, then T has an infinite branch. By (1) there will be at least one node r in T_0 such that **Br** is true in $\mathcal{M}$. By (2) there will in fact be *exactly* one such node, call it r_0. By (3) applied with r_0 as s, there will be at least one node r immediately above r_0 such that **Br** is true in $\mathcal{M}$. By (2) there will in fact be *exactly* one such node, call it r_1. Repeating the process, we obtain $r_0, r_1, r_2, \ldots$, each immediately above the one before, which is to say that we obtain an infinite branch.

We next show that Γ does have a model. By the compactness theorem, it is enough to show that any finite subset Δ of Γ has a model. In fact, we can show that for any k, the set Γ_k containing the sentence (1), all the sentences (2), and all the sentences (3) *for s of level less than* k has a model. We can then, given a finite Δ, apply this fact to the least k such that all **s** occurring in Δ are of level $<k$, to conclude that Δ has a model.

To obtain a model of Γ_k, let the domain be T, and let the denotation of each constant **r** be the node r. It remains to assign a denotation to **B**. Take any t_k at level T_k, and take as the denotation of **B** the set consisting of t_k, the node t_{k-1} at level T_{k-1} immediately below t_k, the node t_{k-2}at level T_{k-2} immediately below t_{k-1}, and so on down until we reach a node t_0 at level 0.

The presence of t_0 in the denotation of **B** will make (1) true. Since we have included in the denotation of **B** only one node t_i at each level T_i for $i \leq k$ and none at higher levels, (2) will be true. For a sentence of form (3) with s of level $i < k$, the first disjunct will be true unless s is t_i, in which case the $\mathbf{B}t_{i+1}$ will be among the other disjuncts, and will be true. In either case, then, (3) will be true for every sentence of this type in Γ_k. [The sentence of form (3) with t_k as s will be false, but that sentence is not in Γ_k.]

Before indicating the connection of Ramsey's theorem with the kind of logical phenomena we have been concerned with in this book, we digress a moment to present a pretty application of Ramsey's theorem.

26.4 Corollary (Schur's theorem). Suppose that each natural number is 'painted' exactly one of some finite number of 'colors'. Then there are positive integers x, y, z all the same color such that $x + y = z$.

Proof: Suppose the number of colors is s. Paint each size-2 set $\{i, j\}$, $i < j$, the same color as the natural number $j - i$. By Ramsey's theorem ($r = 2, n = 3$), there are a positive integer $m \geq 3$ and a size-3 subset $\{i, j, k\}$ of $\{0, 1, \ldots, m - 1\}$ with $i < j < k$, such that $\{i, j\}$, $\{j, k\}$ and $\{i, k\}$ are all the same color. Let $x = j - i$, $y = k - j$, and $z = k - i$. Then x, y, z are positive integers all the same color, and $x + y = z$.

Ramsey's theorem is, in fact, just the starting point for a large body of results in combinatorial mathematics. It is possible to add some bells and whistles to the basic statement of the theorem. Call a nonempty set Y of natural numbers *glorious* if Y has more than p elements, where p is the least element of Y. Since every infinite set is automatically glorious, it would add nothing to the infinitary version of Ramsey's theorem to change 'infinite homogeneous set' to 'glorious infinite homogeneous set'. It does, however, add something to the statement of the original Ramsey's theorem to change 'size-n homogeneous set' to 'glorious size-n homogeneous set'.

Let us call the result of this change the *glorified* Ramsey's theorem. Essentially the same proof we have given for Ramsey's theorem proves the glorified Ramsey's theorem. (At the beginning, take T to be the set of partitions without *glorious* size-n homogeneous sets, and towards the end, take Z to be the set of the first q elements of Y, where q is the *maximum* of n and p, p being the least element of Y.) There is, however, an interesting difference in logical status between the two.

While the proof we have presented for Ramsey's theorem involved a detour through the infinite, F. P. Ramsey's original proof of Ramsey's theorem did not. Using a reasonable coding of finite sets of natural numbers by natural numbers, Ramsey's theorem can be expressed in the language of arithmetic, and by 'formalizing' Ramsey's proof, it can be proved in **P**. By contrast, the glorified Ramsey's theorem, though it can be expressed in the language of arithmetic, can*not* be proved in **P**.

It is an example of a sentence undecidable in **P** that is far more natural, mathematically speaking, than any we have encountered so far. (The sentences involved in Gödel's theorem or Chaitin's theorem, for instance, are '*meta*mathematical', being about provability and computability, not ordinary mathematical notions on the order of those occurring in Ramsey's theorem.) Unfortunately, the *Paris–Harrington theorem*, which tells us that glorified Ramsey's theorem is undecidable in **P**, requires a deeper analysis of nonstandard models than that undertaken in the preceding chapter, and is beyond the scope of a book such as this one.

Problems

26.1 Show that at a party attended by at least nine persons, any two of whom either like each other or dislike each other, either there are four, any two of whom like each other, or there are three, any two of whom dislike each other.

26.2 Show that at a party attended by at least eighteen persons, any two of whom either like each other or dislike each other, either there are four, any two of whom like each other, or there are four, any two of whom dislike each other.

26.3 A finite set of points in the plane, none lying on the line between any other two, is said to be *convex* if no point lies in the interior of the triangle formed by any three other points, as on the left in Figure 26-3. It is not hard to show that given any set of five points in the plane, none lying on the line between any other two, there is a convex subset of four points. The *Erdös–Szekeres* theorem states that, more generally, for any number $n > 4$ there exists a number m such that given a set of (at least) m points in the plane, none lying on the line between any other two, there is a convex subset of (at least) n points. Show how this theorem follows from Ramsey's theorem.

Figure 26-3. Convex and concave sets of points.

26.4 Show that the general case of Ramsey's theorem follows from the special case with $s = 2$, by induction on s.

26.5 For $r = s = 2$ and $n = 3$, each node in the tree used in the proof of Theorem 26.1 in section 26.2 can be represented by a picture in the style of Figure 26-1. How many such nodes will there be in the tree?

26.6 Prove König's lemma directly, that is, without using the compactness theorem, by considering the subtree T^* of T consisting of all nodes that have infinitely many nodes above them (where *above* means either immediately above, or immediately above something immediately above, or . . .).

27

Modal Logic and Provability

Modal logic extends 'classical' logic by adding new logical operators $\Box$ and $\Diamond$ for 'necessity' and 'possibility'. Section 27.1 is an exposition of the rudiments of (sentential) modal logic. Section 27.2 indicates how a particular system of modal logic **GL** *is related to the kinds of questions about provability in* **P** *we considered in Chapters 17 and 18. This connection motivates the closer examination of* **GL** *then undertaken in section 27.3.*

27.1 Modal Logic

Introductory textbooks in logic devote considerable attention to a part of logic we have not given separate consideration: *sentential logic*. In this part of logic, the *only* nonlogical symbols are an enumerable infinity of *sentence letters*, and the only logical operators are negation, conjunction, and disjunction: $\sim$, &, $\vee$. Alternatively, the operators may be taken to be the constant false ($\bot$) and the conditional ($\rightarrow$). The syntax of sentential logic is very simple: sentence letters are sentences, the constant $\bot$ is a sentence, and if A and B are sentences, so is $(A \rightarrow B)$.

The semantics is also simple: an interpretation is simply an assignment ω of truth values, true (represented by 1) or false (represented by 0), to the sentence letters. The valuation is extended to formulas by letting $\omega(\bot) = 0$, and letting $\omega(A \rightarrow B) = 1$ if and only if, if $\omega(A) = 1$, then $\omega(B) = 1$. In other words, $\omega(A \rightarrow B) = 1$ if $\omega(A) = 0$ or $\omega(B) = 1$ or both, and $\omega(A \rightarrow B) = 0$ if $\omega(A) = 1$ and $\omega(B) = 0$. $\sim A$ may be considered an abbreviation for $(A \rightarrow \bot)$, which works out to be true if and only if A is false. $(A \,\&\, B)$ may similarly be taken to be an abbreviation for $\sim(A \rightarrow \sim B)$, which works out to be true if and only if A and B are both true, and $(A \vee B)$ may be taken to be an abbreviation for $(\sim A \rightarrow B)$.

Validity and implication are defined in terms of interpretations: a sentence D is implied by a set of sentences Γ if it is true in every interpretation in which all sentences in Γ are true, and D is valid if it is true in all interpretations. It is decidable whether a given sentence D is valid, since whether D comes out true on an interpretation ω depends only on the values ω assigns to the finitely many sentence letters that occur in D. If there are only k of these, this means that only a finite number of interpretations, namely 2^k of them, need to be checked to see if they make D true. Similar remarks apply to implication.

What is done in introductory textbooks that we have not done here is to work out many particular examples of valid and invalid sentences, and implications and nonimplications among sentences. We are simply going to presume a certain facility with recognizing sentential validity and implication.

Modal sentential logic adds to the apparatus of ordinary or 'classical' sentential logic one more logical operator, the box $\Box$, read 'necessarily' or 'it must be the case that'. One more clause is added to the definition of sentence: if A is a sentence, so is $\Box A$. The diamond $\Diamond$, read 'possibly' or 'it may be the case that', is treated as an abbreviation: $\Diamond A$ abbreviates $\sim\Box\sim A$.

A modal sentence is said to be a *tautology* if it can be obtained from a valid sentence of nonmodal sentential logic by substituting modal sentences for sentence letters. Thus, since $p \vee \sim p$ is valid for any sentence letter p, $A \vee \sim A$ is a tautology for any modal sentence A. Analogously, *tautological consequence* for modal logic is definable in terms of implication for nonmodal sentential logic. Thus since q is implied by p and $p \rightarrow q$ for any sentence letters p and q, B is a tautologous consequence of A and $A \rightarrow B$ for any modal sentences A and B. The inference from A and $A \rightarrow B$ to B is traditionally called *modus ponens*.

There is no single accepted view as to what modal sentences are to be considered modally valid, beyond tautologies. Rather, there are a variety of systems of modal logic, each with its own notion of a sentence being demonstrable.

The *minimal* system of modal sentential logic, **K**, may be described as follows. The *axioms* of **K** include all tautologies, and all sentences of the form

$$\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B).$$

The *rules* of **K** allow one to pass from earlier sentences to any sentence that is a tautologous consequence of them, and to pass

$$\text{from } A \text{ to } \Box A.$$

The latter rule is called the rule of *necessitation*. A *demonstration* in **K** is a sequence of sentences, each of which either is an axiom or follows from earlier ones by a rule. A sentence is then *demonstrable* in **K**, or a *theorem* of **K**, if it is the last sentence of some demonstration. Given a finite set $\Gamma = \{C_1, \ldots, C_n\}$, we write $\wedge C$ for the conjunction of all its members, and say Γ is *inconsistent* if $\sim\wedge C$ is a theorem. We say a sentence D is *deducible* from Γ if $\wedge C \rightarrow D$ is a theorem. The usual relationships hold.

Stronger systems can be obtained by adding additional classes of sentences as axioms, resulting in a larger class of theorems. The following are among the candidates:

(A1) $\Box A \rightarrow A$

(A2) $A \rightarrow \Box\Diamond A$

(A3) $\Box A \rightarrow \Box\Box A$

(A4) $\Box(\Box A \rightarrow A) \rightarrow \Box A$.

For any system **S** we write $\vdash_{\mathbf{S}} A$ to mean that A is a theorem of **S**.

There is a notion of *interpretation* or *model* for **K**. We are going to be interested only in *finite* models, so we build finiteness into the definition. A *model* for **K** will be a triple $\mathcal{W} = (W, >, \omega)$, where W is a nonempty finite set, $>$ a two-place relation on it, and ω a valuation or assignment of truth values true or false (represented by 1 or 0) not to sentence letters but to *pairs* (w, p) consisting of an element w of W and a sentence letter p. The notion $\mathcal{W}, w \models A$ of a sentence A being *true* in a model $\mathcal{W}$ and an element w is defined by induction on complexity. The clauses are as follows:

$$\begin{array}{lll}
\mathcal{W}, w \models p \text{ for } p \text{ a sentence letter} & \text{iff} & \omega(w, p) = 1 \\
\text{not}\, \mathcal{W},\ w \models \bot & & \\
\mathcal{W}, w \models (A \to B) & \text{iff} & \text{not}\, \mathcal{W},\ w \models A \ \ or\ \ \mathcal{W},\ w \models B \\
\mathcal{W}, w \models \Box A & \text{iff} & \mathcal{W}, v \models A \ \text{ for all } v < w.
\end{array}$$

(We have written $v < w$ for $w > v$.) Note that the clauses for $\bot$ and $\to$ are just like those for nonmodal sentential logic. We say a sentence A is *valid* in the model $\mathcal{W}$ if $\mathcal{W}, w \models A$ for all w in W.

Stronger notions of model of can be obtained by imposing conditions that the relation $>$ must fulfill, resulting in a smaller class of models. The following are among the candidates.

$(W1)$	*Reflexivity*:	for all w,	$w > w$
$(W2)$	*Symmetry*:	for all w and v,	if $w > v$, then $v > w$
$(W3)$	*Transitivity*:	for all w, v, and u,	if $w > v > u$, then $w > u$
$(W4)$	*Irreflexivity*:	for all w,	not $w > w$.

(We have written $w > v > u$ for $w > v$ and $v > u$.) For any class Σ of models, we say A is *valid* in Σ, and write $\models_\Sigma A$, if A is valid in all $\mathcal{W}$ in Σ.

Let **S** be a system obtained by adding axioms and Σ a class obtained by imposing conditions on $>$. If whenever $\vdash_{\mathbf{S}} A$ we have $\models_\Sigma A$, we say **S** is *sound* for Σ. If whenever $\models_\Sigma A$ we have $\vdash_{\mathbf{S}} A$, we say **S** is *complete* for Σ. A soundness and completeness theorem relating the system **S** to a class of models Σ generally tells us that the (set of theorems of) the system **S** is decidable: given a sentence A, to determine whether or not A is a theorem, one can simultaneously run through all demonstrations and through all finite models, until one finds either a demonstration of A or a model of $\sim A$. A large class of such soundness and completeness theorems are known, of which we state the most basic as our first theorem.

27.1 Theorem (Kripke soundness and completeness theorems). Let **S** be obtained by adding to **K** a subset of {(A1), (A2), (A3)}. Let Σ be obtained by imposing on $<_W$ the corresponding subset of {(W1), (W2), (W3)}. Then **S** is sound and complete for Σ.

Since there are eight possible subsets, we have eight theorems here. We are going to leave most of them to the reader, and give proofs for just two: the case of the empty set, and the case of the set {(A3)} corresponding to {(W3)}: **K** is sound and complete for the class of all models, and ***K*** + (A3) is sound and complete for the class of transitive models. Before launching into the proofs we need a couple of simple facts.

27.2 Lemma. For any extension **S** of **K**, if $\vdash_{\mathbf{S}} A \to B$, then $\vdash_{\mathbf{S}} \Box A \to \Box B$.

Proof: Suppose we have a proof of $A \to B$. Then we can then extend it as follows:

(1)	$A \to B$	G
(2)	$\Box(A \to B)$	N(1)
(3)	$\Box(A \to B) \to (\Box A \to \Box B)$	A
(4)	$\Box A \to \Box B$	T(2), (3)

The annotations mean: G[iven], [by] N[ecessitation from step] (1), A[xiom], and T[autological consequence of steps] (2), (3).

27.3 Lemma. $\vdash_{\mathbf{K}} (\Box A \,\&\, \Box B) \leftrightarrow \Box(A \,\&\, B)$, and similarly for more conjuncts.

Proof:

(1)	$(A \,\&\, B) \to A$	T
(2)	$\Box(A \,\&\, B) \to \Box A$	25.2(1)
(3)	$\Box(A \,\&\, B) \to \Box B$	S(2)
(4)	$A \to (B \to (A \,\&\, B))$	T
(5)	$\Box A \to \Box(B \to (A \,\&\, B))$	25.2(4)
(6)	$\Box(B \to (A \,\&\, B)) \to (\Box B \to \Box(A \,\&\, B))$	A
(7)	$(\Box A \,\&\, \Box B) \leftrightarrow \Box(A \,\&\, B)$	T(2), (3), (5), (6)

The first three annotations mean: T[autology], [by Lemma] 25.2 [from] (1), and S[imilar to] (2).

Proof of Theorem 27.1: There are four assertions to be proved.

K *is sound for the class of all models*. Let $\mathcal{W}$ be any model, and write $w \models A$ for $\mathcal{W}$, $w \models A$. It will be enough to show that if A is an axiom, then for all w we have $w \models A$, and that if A follows by a rule from $B_1, \ldots, B_n$, and for all w we have $w \models B_i$ for each i, then for all w we have $w \models A$.

Axioms. If A is tautologous, the clauses of the definition of $\models$ for $\bot$ and $\to$ guarantee that $w \models A$. As for axioms of the other kind, if $w \models \Box(A \to B)$ and $w \models \Box A$, then for any $v < w$, $v \models A \to B$ and $v \models A$. Hence $v \models B$ for any $v < w$, and $w \models \Box B$. So $w \models \Box(A \to B) \to (\Box A \to \Box B)$.

Rules. If A is a tautologous consequence of the B_i and $w \models B_i$ for each i, then again the clauses of the definition of $\models$ for $\bot$ and $\to$ guarantee that $w \models A$. For the other rule, if $w \models A$ for all w, then *a fortiori* for any w and any $v < w$, we have $v \models A$. So $w \models \Box A$.

K *is complete for the class of all models*. Suppose A is not a theorem. We construct a model in which A is not valid. We call a sentence a *formula* if it is either a subsentence of A or the negation of one. We call a consistent set of formulas *maximal* if for every formula B it contains one of every pair of formulas B, $\sim B$. First note that $\{\sim A\}$ is consistent: otherwise $\sim\sim A$ is a theorem, and hence A is, as a tautologous consequence. Further, note that every consistent set Γ is a subset of some maximal set: $\bigwedge\Gamma$ is equivalent to some nonempty disjunction each of whose conjuncts is a conjunction of formulas that contains the members of Γ and contains every formula exactly once, plain or negated. Further, note that a maximal set contains any formula deducible from it: otherwise it would contain the *negation* of that formula; but a set that contains the negation of a formula deducible from it is inconsistent.

Let W be the set of all maximal sets. W is not empty, since $\{\sim A\}$ is consistent and therefore a subset of some maximal set. W is finite: if there are only k subsentences of A, there are at most 2^k maximal sets. Define a relation $>$ on W by letting $w > v$ if and only if whenever a formula $\Box A$ is in w, the formula A is in v. Finally, for w in W and sentence letter p, let $\omega(w, p) = 1$ if p is in w, and $\omega(w, p) = 0$ if not. Let $\mathcal{W} = (W, >, \omega)$. We are going to show by induction on complexity that for any w in W and any formula B we have $\mathcal{W}, w \models B$ if and only if B is in w. Since there is a w containing $\sim A$ rather than A, it follows that A is not valid in $\mathcal{W}$.

For the base step, if B is a sentence letter p, then p is in w iff $\omega(w, p) = 1$ iff $w \models p$. If B is $\bot$, then $\bot$ is not in w, since w is consistent, and also it is not the case that $w \models \bot$. For the induction step, if B is $C \to D$, then C and D are subsentences of A, and $\sim B \leftrightarrow (C \,\&\, \sim D)$ is a theorem, being tautologous. Thus B is not in w iff (by maximality) $\sim B$ is in w, iff C and $\sim D$ are in w, iff (by the induction hypothesis) $w \models C$ and not $w \models D$, iff not $w \models C \to D$. If B is $\Box C$, the induction hypothesis is that for any v, $v \models C$ iff C is in v. We want to show that $w \models \Box C$ iff $\Box C$ is in w. For the 'if' direction, suppose $\Box C$ is in w. Then for any $v < w$, C is in v and so $v \models C$. It follows that $w \models \Box C$.

For the 'only if' direction, suppose $w \models \Box C$. Let

$$V = \{D_1, \ldots, D_m, \sim C\}$$

where the $\Box D_i$ for $1 \leq i \leq m$ are all the formulas in w that begin with $\Box$. Is V consistent? If it is, then it is contained in some maximal v. Since all D_i are in v, we have $v < w$. Since $\sim C$ is in v, not $v \models C$, which is impossible, since $w \models \Box C$. So V is inconsistent, and it follows that

$$(D_1 \,\&\cdots\&\, D_m) \to C$$

is a theorem. By Lemma 27.2,

$$\Box(D_1 \,\&\cdots\&\, D_m) \to \Box C$$

is a theorem, and so by Lemma 27.3,

$$(\Box D_1 \,\&\cdots\&\, \Box D_m) \to \Box C$$

is a theorem. Hence, since each $\Box D_i$ is in w, $\Box C$ is in w.

K + (A3) *is sound for transitive models.* If $w \models \Box A$, then for any $v < w$ it is the case that for any $u < v$ we have by transitivity $u < w$, and so $u \models A$. Thus $v \models \Box A$ for any $v < w$, and $w \models \Box\Box A$. Thus $w \models \Box A \to \Box\Box A$.

K + (A3) *is complete for transitive models.* The construction used to prove **K** complete for the class of all models needs to be modified. Define $w > v$ if and only if whenever a formula $\Box B$ is in w, the formulas $\Box B$ are both B in v. Then $>$ will be transitive. For if $w > v > u$, then whenever $\Box A$ is in w, $\Box A$ and A will be in v, and since the former is in v, both will also by in u, so $w > u$.

The only other part of the proof that needs modification is the proof that if $w \models \Box C$, then $\Box C$ is in w. So suppose $w \models \Box C$, and let

$$V = \{\Box D_1, D_1, \ldots, \Box D_m, D_m, {\sim}C\}$$

where the $\Box D_i$ are all the formulas in w that begin with $\Box$. If V is consistent and v is a maximal set containing it, then $w > v$ and $v \models {\sim}C$, which is impossible. It follows that

$$\Box D_1 \mathbin{\&} D_1 \mathbin{\&} \cdots \mathbin{\&} \Box D_m \mathbin{\&} D_m \to C$$
$$\Box(\Box D_1 \mathbin{\&} D_1 \mathbin{\&} \cdots \mathbin{\&} \Box D_m \mathbin{\&} D_m) \to \Box C$$
$$(\Box\Box D_1 \mathbin{\&} \Box D_1 \mathbin{\&} \cdots \mathbin{\&} \Box\Box D_m \mathbin{\&} \Box D_m) \to \Box C$$

are theorems, and hence any tautologous consequence of the last of these and the axioms $\Box D_i \to \Box\Box D_i$ is a theorem, and this includes

$$(\Box D_1 \mathbin{\&} \cdots \mathbin{\&} \Box D_m) \to \Box C$$

from which it follows that $w \models \Box C$.

Besides its use in proving decidability, the preceding theorem makes it possible to prove syntactic results by semantic arguments. Let us give three illustrations. In both the first and the second, A and B are arbitrary sentences, q a sentence letter not contained in either, $F(q)$ any sentence, and $F(A)$ and $F(B)$ the results of substituting A and B respectively for any and all occurrences of q in F. In the second and third, $\boxdot A$ abbreviates $\Box A \mathbin{\&} A$. In the third, $\bullet A$ is the result of replacing $\Box$ by $\boxdot$ throughout A.

27.4 Proposition. If $\vdash_{\mathbf{K}} A \leftrightarrow B$, then $\vdash_{\mathbf{K}} F(A) \leftrightarrow F(B)$.

27.5 Proposition. $\vdash_{\mathbf{K}+(\mathrm{A3})} \boxdot(A \leftrightarrow B) \to \boxdot(F(A) \leftrightarrow F(B))$.

27.6 Proposition. If $\vdash_{\mathbf{K}+(\mathrm{A1})+(\mathrm{A3})} A$, then $\vdash_{\mathbf{K}+(\mathrm{A3})} \bullet A$.

Proof: For Proposition 27.4, it is easily seen (by induction on complexity of F) that if $\mathcal{W} = (W, >, \omega)$ and we let $\mathcal{W}' = (W, >, \omega')$, where ω' is like ω except that for all w

$$\omega'(w, q) = 1 \quad \text{if and only} \quad \text{if } \mathcal{W}, w \models A$$

then for all w, we have

$$\mathcal{W}, w \models F(A) \quad \text{if and only} \quad \text{if } \mathcal{W}', w \models F(q).$$

But if $\vdash_{\mathbf{K}} A \leftrightarrow B$, then by soundness for all w we have

$$\mathcal{W}, w \models A \quad \text{if and only} \quad \text{if } \mathcal{W}, w \models B$$

and hence

$$\mathcal{W}, w \models F(B) \quad \text{if and only} \quad \text{if } \mathcal{W}', w \models F(q)$$
$$\mathcal{W}, w \models F(A) \quad \text{if and only} \quad \text{if } \mathcal{W}, w \models F(B).$$

So by completeness we have $\vdash_{\mathbf{K}} F(A) \leftrightarrow F(B)$.

For Proposition 27.5, it is easily seen (by induction on complexity of A) that since each clause in the definition of truth at w mentions only w and those v with $w > v$, for any $\mathcal{W} = (W, >, \omega)$ and any w in W, whether $\mathcal{W}, w \models A$ depends only on the values of $\omega(v, p)$ for those v such that there is a sequence

$$w = w_0 > w_1 > \cdots > w_n = v.$$

If $>$ is transitive, these are simply those v with $w \geq v$ (that is, $w = v$ or $w > v$). Thus for any transitive model $(W, >, \omega)$ and any w, letting $W_w = \{v\colon w \geq v\}$ and $\mathcal{W}_w = (W_w, >, \omega)$, we have

$$\mathcal{W}, w \models A \quad \text{if and only if} \quad \mathcal{W}_w, w \models A.$$

Now

$$\mathcal{W}, w \models \boxdot C \quad \text{if and only if} \quad \text{for all } v \leq w \quad \text{we have} \quad \mathcal{W}, v \models C.$$

Thus if $\mathcal{W}, w \models \boxdot(A \leftrightarrow B)$, then $\mathcal{W}_w, v \models A \leftrightarrow B$ for all v in W_w. Then, arguing as in the proof of Proposition 27.4, we have $\mathcal{W}_w, v \models F(A) \leftrightarrow F(B)$ for all such v, and so $\mathcal{W}, w \models \boxdot(F(A) \leftrightarrow F(B))$. This shows

$$\mathcal{W}, w \models \boxdot(A \leftrightarrow B) \to \boxdot(F(A) \leftrightarrow F(B))$$

for all transitive $\mathcal{W}$ and all w, from which the conclusion of the proposition follows by soundness and completeness.

For Proposition 27.6, for any model $\mathcal{W} = (W, >, \omega)$, let $\bullet\mathcal{W} = (W, \geq, \omega)$. It is easily seen (by induction on complexity) that for any A and any w in W

$$\mathcal{W}, w \models A \quad \text{if and only if} \quad \bullet\mathcal{W}, w \models \bullet A.$$

$\bullet\mathcal{W}$ is always reflexive, is the same as $\mathcal{W}$ if $\mathcal{W}$ was already reflexive, and is transitive if and only if $\mathcal{W}$ was transitive. It follows that A is valid in all transitive models if and only if $\bullet A$ is valid in all reflexive transitive models. The conclusion of the proposition follows by soundness and completeness.

The conclusion of Proposition 27.4 actually applies to *any system containing* **K** in place of **K**, and the conclusions of Propositions 27.5 and 27.6 to *any system containing* **K** + (A3) in place of **K** + (A3). We are going to be especially interested in the system **GL** = **K** + (A3) + (A4). The soundness and completeness theorems for **GL** are a little tricky to prove, and require one more preliminary lemma.

27.7 Lemma. If $\vdash_{\mathbf{GL}} (\Box A \,\&\, A \,\&\, \Box B \,\&\, B \,\&\, \Box C) \to C$, then $\vdash_{\mathbf{GL}} (\Box A \,\&\, \Box B) \to \Box C$, and similarly for any number of conjuncts.

Proof: The hypothesis of the lemma yields

$$\vdash_{\mathbf{GL}} (\Box A \,\&\, A \,\&\, \Box B \,\&\, B) \to (\Box C \to C).$$

Then, as in the proof of the completeness of **K** + (A3) for transitive models, we get

$$\vdash_{\mathbf{GL}} (\Box A \,\&\, \Box B) \to \Box(\Box C \to C).$$

From this and the axiom $\Box(\Box C \to C) \to \Box C$ we get as a tautologous consequence the conclusion of the lemma.

27.8 Theorem (Segerberg soundness and completeness theorems). **GL** is sound and complete for transitive, irreflexive models.

Proof: *Soundness.* We need only show, in addition to what has been shown in the proof of the soundness of **K** + (A3) for transitive models, that if a model is also irreflexive, then $w \models \Box(\Box B \to B) \to \Box B$ for any w. To show this we need a notion of *rank*.

First note that if $>$ is a transitive, irreflexive relation on a nonempty set W, then whenever $w_0 > w_1 > \cdots > w_m$, by transitivity we have $w_i > w_j$ whenever $i < j$, and hence by irreflexivity $w_i \neq w_j$ whenever $i \neq j$. Thus if W has only m elements, we can never have $w_0 > w_1 > \cdots > w_m$. Thus in any transitive, irreflexive model, there is for any w a greatest natural number k for which there exists elements $w = w_0 > \cdots > w_k$. We call this k the *rank* $\mathrm{rk}(w)$ of w. If there is no $v < w$, then $\mathrm{rk}(w) = 0$. If $v < w$, then $\mathrm{rk}(v) < \mathrm{rk}(w)$. And if $j < \mathrm{rk}(w)$, then there is an element $v < w$ with $\mathrm{rk}(v) = j$. (If $w = w_0 > \cdots > w_{\mathrm{rk}(w)}$, then $w_{\mathrm{rk}(w)-j}$ is such a v.)

Now suppose $w \models \Box(\Box B \to B)$ but not $w \models \Box B$. Then there is some $v < w$ such that not $v \models B$. Take such a v *of lowest possible rank*. Then for all $u < v$, by transitivity $u < w$, and since $\mathrm{rk}(u) < \mathrm{rk}(v)$, $u \models B$. This shows $v \models \Box B$, and since not $v \models B$, not $v \models \Box B \to B$. But that is impossible, since $v < w$ and $w \models \Box(\Box B \to B)$. Thus if $w \models \Box(\Box B \to B)$ then $w \models \Box B$, so for all w, $w \models \Box(\Box B \to B) \to \Box B$.

Completeness. We modify the proof of the completeness of **K** + (A3) by letting W be not the set of all maximal w, but only of those for which not $w > w$. This makes the model irreflexive.

The only other part of the proof that needs modification is the proof that if $w \models \Box C$, then $\Box C$ is in w. So suppose $w \models \Box C$, and let

$$V = \{\Box D_1, D_1, \ldots, \Box D_m, D_m, \Box C, \sim C\}$$

where the $\Box D_i$ are all the formulas in w that begin with $\Box$. If V is consistent and v is a maximal set containing it, then since $\Box C$ is in v but C cannot be in v, we have not $v > v$, and v is in W. Also $w > v$ and $v \models \sim C$, which is impossible. It follows that

$$\Box D_1 \,\&\, D_1 \,\&\, \cdots \,\&\, \Box D_m \,\&\, D_m \,\&\, \Box C \to C$$

is a theorem, and hence by the preceding lemma so is

$$(\Box D_1 \,\&\, \cdots \,\&\, \Box D_m) \to \Box C$$

from which it follows that $w \models \Box C$.

27.2 The Logic of Provability

Let us begin by explaining why the system **GL** is of special interest in connection with the matters with which we have been concerned through most of this book. Let L

be the language of arithmetic, and ϕ a function assigning to sentence letters sentences of L. We associate to any modal sentence A a sentence A^ϕ of L as follows:

$$\begin{aligned} p^\phi &= \phi(p) \qquad \text{for } p \text{ a sentence letter} \\ \bot^\phi &= \mathbf{0} = \mathbf{1} \\ (B \to C)^\phi &= B^\phi \to C^\phi \\ (\Box B)^\phi &= \mathrm{Prv}(\ulcorner B^\phi \urcorner) \end{aligned}$$

where Prv is a provability predicate for **P**, in the sense of chapter 18. Then we have the following relationship between **GL** and **P**:

27.9 Theorem (Arithmetical soundness theorem). If $\vdash_{\mathbf{GL}} A$, then for all ϕ, $\vdash_{\mathbf{P}} A^\phi$.

Proof: Fix any ϕ. It is sufficient to show that $\vdash_{\mathbf{P}} A^\phi$ for each axiom of **GL**, and that if B follows by rules of **GL** from $A_1, \ldots, A_m$ and $\vdash_{\mathbf{P}} A_i^\phi$ for $1 \le i \le m$, then $\vdash_{\mathbf{P}} B^\phi$. This is immediate for a tautologous axioms, and for the rule permitting passage to tautologous consequences, so we need only consider the three kinds of modal axioms, and the one modal rule, necessitation. For necessitation, what we want to show is that if $\vdash_{\mathbf{P}} B^\phi$, then $\vdash_{\mathbf{P}} (\Box B)^\phi$, which is to say $\vdash_{\mathbf{P}} \mathrm{Prv}(\ulcorner B^\phi \urcorner)$. But this is precisely property (P1) in the definition of a provability predicate in Chapter 18 (Lemma 18.2). The axioms $\Box(B \to C) \to (\Box B \to \Box C)$ and $\Box B \to \Box\Box B$ correspond in the same way to the remaining properties (P2) and (P3) in that definition.

It remains to show that $\vdash_{\mathbf{P}} A^\phi$ where A is an axiom of the form

$$\Box(\Box B \to B) \to \Box B.$$

By Löb's theorem it suffices to show $\vdash_{\mathbf{P}} \mathrm{Prv}(\ulcorner A^\phi \urcorner) \to A^\phi$. To this end, write S for B^ϕ, so that A^ϕ is

$$\mathrm{Prv}(\ulcorner \mathrm{Prv}(\ulcorner S \urcorner) \to S \urcorner) \to \mathrm{Prv}(\ulcorner S \urcorner).$$

By (P2)

$$\mathrm{Prv}(\ulcorner A^\phi \urcorner) \to [\mathrm{Prv}(\ulcorner \mathrm{Prv}(\ulcorner \mathrm{Prv}(\ulcorner S \urcorner) \to S \urcorner)\urcorner) \to \mathrm{Prv}(\ulcorner \mathrm{Prv}(\ulcorner S \urcorner)\urcorner)]$$

$$\mathrm{Prv}(\ulcorner \mathrm{Prv}(\ulcorner S \urcorner) \to S \urcorner) \to [\mathrm{Prv}(\ulcorner \mathrm{Prv}(\ulcorner S \urcorner)\urcorner) \to \mathrm{Prv}(\ulcorner S \urcorner)]$$

are theorems of **P**, and by (P3)

$$\mathrm{Prv}(\ulcorner \mathrm{Prv}(\ulcorner S \urcorner) \to S \urcorner) \to \mathrm{Prv}(\ulcorner \mathrm{Prv}(\ulcorner \mathrm{Prv}(\ulcorner S \urcorner) \to S \urcorner)\urcorner)$$

is also a theorem of **P**. And therefore

$$\mathrm{Prv}(\ulcorner A^\phi \urcorner) \to [\mathrm{Prv}(\ulcorner \mathrm{Prv}(\ulcorner S \urcorner) \to S \urcorner) \to \mathrm{Prv}(\ulcorner S \urcorner)]$$

which is to say $\mathrm{Prv}(\ulcorner A^\phi \urcorner) \to A^\phi$, being a tautological consequences of these three sentences, is a theorem of **P** as required.

The converse of Theorem 27.9 is the *Solovay completeness theorem*: if for all ϕ, $\vdash_{\mathbf{P}} A^\phi$, then $\vdash_{\mathbf{GL}} A$. The proof of this result, which will not be needed in what follows, is beyond the scope of a book such as this.

Theorem 27.9 enables us to establish results about provability in **P** by establishing results about **GL**. The remainder of this section will be devoted to the statement of two results about **GL**, the *De Iongh–Sambin fixed point theorem* and a *normal form theorem for letterless sentences*, with an indication of their consequences for **P**. The proofs of these two results are deferred to the next section. Before stating the theorems, a few preliminary definitions will be required.

We call a sentence A *modalized* in the sentence letter p if every occurrence of p in A is part of a subsentence beginning with $\Box$. Thus if A is modalized in p, then A is a truth-functional compound of sentences $\Box B_i$ and sentence letters other than p. (Sentences not containing p at all count *vacuously* as modalized in p, while $\bot$ and truth-functional compounds thereof count *conventionally* as truth-functional compounds of *any* sentences.) A sentence is a *p-sentence* if it contains no sentence letter but p, and *letterless* if it contains no sentence letters at all.

So for example $\Box p \to \Box{\sim}p$ is a p-sentence modalized in p, as is (vacuously and conventionally) the letterless sentence ${\sim}\bot$, whereas $q \to \Box p$ is not a p-sentence but is modalized in p, and ${\sim}p$ is a p-sentence not modalized in p, and finally $q \to p$ is neither a p-sentence nor modalized in p.

A sentence H is a *fixed point* of A (with respect to p) if H contains only sentence letters contained in A, H does not contain p, and

$$\vdash_{\mathbf{GL}} \boxdot(p \leftrightarrow A) \to (p \leftrightarrow H).$$

For any A, $\Box^0 A = A$ and $\Box^{n+1} A = \Box\Box^n A$. A letterless sentence H is in *normal form* if it is a truth-functional compound of sentences $\Box^n \bot$. Sentences B and C are *equivalent* in **GL** if $\vdash_{\mathbf{GL}} (B \leftrightarrow C)$.

27.10 Theorem (Fixed point theorem). If A is modalized in p, then there exists a fixed point H for A relative to p.

Several proofs along quite different lines are known. The one we are going to give (Sambin's and Reidhaar-Olson's) has the advantage that it explicitly and effectively associates to any A modalized in p a sentence $A^{\S}$, which is then proved to be a fixed point for A.

27.11 Theorem (Normal form theorem). If B is letterless, then there exists a letterless sentence C in normal form equivalent to B in **GL**.

Again the proof we give will effectively associate to any letterless B a sentence $B^{\#}$ that in normal form equivalent to B in **GL**.

27.12 Corollary. If A is a p-sentence modalized in p, then there exists a letterless sentence H in normal form that is a fixed point for A relative to p.

The corollary follows at once from the preceding two theorems, taking as H the sentence $A^{\S\#}$. Some examples of the H thus associated with certain A are given in Table 27-1.

What does all this tell us about **P**? Suppose we take some formula $\alpha(x)$ of L 'built up from' Prv using truth functions and applying the diagonal lemma to obtain

Table 27-1. *Fixed points in normal form*

A	$\Box p$	$\sim\Box p$	$\Box\sim p$	$\sim\Box\sim p$	$\sim\Box\Box p$	$\Box p \to \Box\sim p$
H	$\sim\bot$	$\sim\Box\bot$	$\Box\bot$	$\bot$	$\sim\Box\Box\bot$	$\Box\Box\bot \to \Box\bot$

a sentence γ such that $\vdash_{\mathbf{P}} \pi_\alpha \leftrightarrow \alpha(\ulcorner \pi_\alpha \urcorner)$. Let us call such a sentence π a sentence of *Gödel type*. Then $\alpha(x)$ corresponds to a p-sentence $A(p)$, to which we may apply Corollary 27.12 in order to obtain a fixed point H in normal form. This H will in turn correspond to a truth-functional compound η of the sentences

$$\mathbf{0} = \mathbf{1}, \qquad \mathrm{Prv}(\ulcorner \mathbf{0} = \mathbf{1} \urcorner), \qquad \mathrm{Prv}(\ulcorner \mathrm{Prv}(\ulcorner \mathbf{0} = \mathbf{1} \urcorner) \urcorner), \ldots$$

and we get $\vdash_{\mathbf{P}} \pi_\alpha \leftrightarrow \eta$. Since moreover the association of A with H is effective, so is the association of α with η. Since the sentences in the displayed sequence are all false (in the standard interpretation), we can effectively determine the truth value of η and so of π_α. In other words, there is a *decision procedure* for sentences of Gödel type.

27.13 Example (‘Cashing out’ theorems about **GL** as theorems about **P**). When $\alpha(x)$ is $\mathrm{Prv}(x)$, then π_α is the Henkin sentence, $A(p)$ is $\Box p$, and H is (according to Table 27-1) $\sim\bot$, so η is $\mathbf{0} \neq \mathbf{1}$, and since $\vdash_{\mathbf{P}} \pi_\alpha \leftrightarrow \mathbf{0} \neq \mathbf{1}$, we get the result that the Henkin sentence is true—and moreover that it is a theorem of **P**, which was Löb’s answer to Henkin’s question. When $\alpha(x)$ is $\sim\mathrm{Prv}(x)$, then π_α is the Gödel sentence, $A(p)$ is $\sim\Box p$, and H is (according to Table 27-1) $\sim\Box\bot$, so η is the consistency sentence $\sim\mathrm{Prv}(\ulcorner \mathbf{0} = \mathbf{1} \urcorner)$, and since $\vdash_{\mathbf{P}} \pi_\alpha \leftrightarrow \sim\mathrm{Prv}(\ulcorner \mathbf{0} = \mathbf{1} \urcorner)$, we get the result that the Gödel sentence is true, which is something that we knew—and moreover that *the Gödel sentence is provably equivalent in* **P** *to the consistency sentence*, which is a connection between the first and second incompleteness theorems that we did *not* know of before.

Each column in Table 27-1 corresponds to another such example.

27.3 The Fixed Point and Normal Form Theorems

We begin with the normal form theorem.

Proof of Theorem 27.11: The proof is by induction on the complexity of B. (Throughout we make free tacit use of Proposition 27.4, permitting substitution of demonstrably equivalent sentences for each other.) It clearly suffices to show how to associate a letterless sentence in normal form equivalent to $\Box C$ with a letterless sentence C in normal form.

First of all, put C in conjunctive normal form, that is, rewrite C as a conjunction $D_1 \mathbin{\&} \cdots \mathbin{\&} D_k$ of disjunctions of sentences $\Box^i\bot$ and $\sim\Box^i\bot$. Since $\Box$ distributes over conjunction by Lemma 27.3, it suffices to find a suitable equivalent for $\Box D$ for any disjunction D of $\Box^i\bot$ and $\sim\Box^i\bot$. So let D be

$$\Box^{n_1}\bot \vee \cdots \vee \Box^{n_p}\bot \vee \sim\Box^{m_1}\bot \vee \cdots \vee \sim\Box^{m_q}\bot.$$

We may assume D has at least one plain disjunct: if not, just add the disjunct $\Box^0\bot = \bot$, and the result will be equivalent to the original.

Using the axiom $\Box B \to \Box\Box B$ and Lemma 27.2, we see $\vdash_{\mathbf{GL}} \Box^i B \to \Box^{i+1} B$ for all i, and hence

$$(*)\quad \vdash_{GL} \Box^i B \to \Box^j B \ \text{ and } \ \vdash_{GL} \sim\Box^j B \to \sim\Box^i B \qquad \text{whenever} \quad i \leq j.$$

So we may replace D by $\Box^n\bot \vee \sim\Box^m\bot$, where $n = \max(n_1, \ldots, n_p)$ and $m = \min(m_1, \ldots, m_q)$. If there were no negated disjuncts, this is just $\Box^n\bot$, and we are done. Otherwise, D is equivalent to $\Box^m\bot \to \Box^n\bot$. If $m \leq n$, then this is a theorem, so we may replace D by $\sim\bot$.

If $m > n$, then $n+1 \leq m$. We claim in this case $\vdash_{\mathbf{GL}} \Box D \leftrightarrow \Box^{n+1}\bot$. In one direction we have

$$\begin{array}{lll}
(1) & \Box^n\bot \to \Box^{n+1}\bot & (*) \\
(2) & (\Box^m\bot \to \Box^n\bot) \to (\Box^m\bot \to \Box^{n+1}\bot) & \mathrm{T}(1) \\
(3) & \Box(\Box^m\bot \to \Box^n\bot) \to \Box(\Box^m\bot \to \Box^{n+1}\bot) & 27.2(2) \\
(4) & \Box(\Box^{n+1}\bot \to \Box^n\bot) \to \Box^{n+1}\bot & \mathrm{A} \\
(5) & \Box(\Box^m\bot \to \Box^n\bot) \to \Box^{n+1}\bot & \mathrm{T}(3), (4) \\
(6) & \Box^n\bot \to (\Box^m\bot \to \Box^n\bot) & \mathrm{T} \\
(7) & \Box^{n+1}\bot \to \Box(\Box^m\bot \to \Box^n\bot) & 27.2(6) \\
(8) & \Box(\Box^m\bot \to \Box^n\bot) \leftrightarrow \Box^{n+1}\bot. & \mathrm{T}(5), (7)
\end{array}$$

And (8) tells us $\vdash_{\mathbf{GL}} \Box D \leftrightarrow \Box^{n+1}\bot$.

Turning to the proof of Theorem 27.10, we begin by describing the transform $A^\S$.Write $\top$ for $\sim\bot$. Let us say that a sentence A is of *grade n* if for some distinct sentence letters $q_1, \ldots, q_n$ (where possibly $n = 0$), and some sentence $B(q_1, \ldots, q_n)$ not containing p but containing all the q_i, and some sequence of distinct sentences $C_1(p), \ldots, C_n(p)$ all containing p, A is the result $B(\Box C_1(p), \ldots, \Box C_n(p))$ of substituting for each q_i in B the sentence $\Box C_i$. If A is modalized in p, then A is of grade n for some n.

If A is of grade 0, then A does not contain p, and is a fixed point of itself. In this case, let $A^\S = A$. If

$$A = B(\Box C_1(p), \ldots, \Box C_{n+1}(p))$$

is of grade $n+1$, for $1 \leq i \leq n+1$ let

$$A_i = B(\Box C_1(p), \ldots, \Box C_{i-1}(p), \top, \Box C_{i+1}(p), \ldots, \Box C_{n+1}(p)).$$

Then A_i is of grade n, and supposing § to be defined for sentences of grade n, let

$$A^\S = B(\Box C_1(A_1^\S), \ldots, \Box C_n(A_{n+1}^\S)).$$

27.14 Examples (Calculating fixed points). We illustrate the procedure by working out $A^\S$ in two cases (incidentally showing how substitution of demonstrably equivalent sentences for each other can result in simplifications of the form of $A^\S$).

Let $A = \Box{\sim}p$. Then $A = B(\Box C_1(p))$, where $B(q_1) = q_1$ and $C_1(p) = {\sim}p$. Now $A_1 = B$ $(\top) = \top$ is of grade 0, so $A_1^{\S} = A_1 = \top$, and $A^{\S} = B(\Box C_1(A_1^{\S})) = \Box{\sim}\top$, which is equivalent to $\Box\bot$, the H associated with this A in Table 27-1.

Let $A = \Box(p \to q) \to \Box{\sim}p$. Then $A = B(\Box C_1(p), \Box C_2(p))$, where $B(q_1,\ q_2) = (q_1 \to q_2)$, $C_1(p) = (p \to q)$, $C_2(p) = {\sim}p$. Now $A_1 = (\top \to \Box{\sim}p)$, which is equivalent to $\Box{\sim}p$, and $A_2 = \Box(p \to q) \to \top$, which is equivalent to $\top$. By the preceding example, $A_1^{\S} = \Box{\sim}\top$, and $A_2^{\S}$ is equivalent to $\top$. So $A^{\S}$ is equivalent to $B(\Box C_1(\Box\bot), \Box \sim C_2(\top)) = \Box(\Box{\sim}\top \to q) \to \Box{\sim}\top$, or $\Box(\Box\bot \to q) \to \Box{\sim}\bot$.

To prove the fixed-point theorem, we show by induction on n that $A^{\S}$ is a fixed point of A for all formulas A modalized in p of grade n. The base step $n = 0$, where $A^{\S} = A$, is trivial. For the induction step, let A, B, C_i be as in the definition of §, let i range over numbers between 1 and $n+1$, write H for $A^{\S}$ and H_i for $A_i^{\S}$, and assume as induction hypothesis that H_i is a fixed point for A_i. Let $\mathcal{W} = (W, >, \omega)$ be a model, and write $w \models D$ for $\mathcal{W}$, $w \models D$. In the statements of the lemmas, w may be any element of W.

27.15 Lemma. Suppose $w \models \boxdot\ (p \leftrightarrow A)$ and $w \models \Box C_i(p)$. Then $w \models C_i(p) \leftrightarrow C_i(H_i)$ and $w \models \Box C_i(p) \leftrightarrow \Box C_i(H_i)$.

Proof: Since $w \models \Box C_i(p)$, by axiom (A3) $w \models \Box\Box C_i(p)$; hence for all $v \leq w$, $v \models \Box C_i(p)$. It follows that $w \models \boxdot(C_i(p) \leftrightarrow \top)$. By Proposition 27.5, $w \models \boxdot(A \leftrightarrow A_i)$, whence by Lemma 27.5 again $w \models \boxdot(p \leftrightarrow A_i)$, since $w \models \boxdot(p \leftrightarrow A)$. Since H_i is a fixed point for A_i, $w \models \boxdot(p \leftrightarrow H_i)$. The conclusion of the lemma follows on applying Proposition 27.5 twice (once to C_i, once to $\Box C_i$).

27.16 Lemma. $w \models \boxdot(p \leftrightarrow A) \to \boxdot(\Box C_i(p) \to \Box C_i(H_i))$.

Proof: Suppose $w \models \boxdot(p \leftrightarrow A)$. By Proposition 27.6, $\boxdot D \to \boxdot\,\boxdot D$ is a theorem, so $w \models \boxdot\boxdot\,(p \leftrightarrow A)$, and if $w \geq v$, then $v \models \boxdot(p \leftrightarrow A)$. Hence if $v \models \Box C_i(p)$, then $v \models \Box C_i(p) \leftrightarrow \Box C_i(H_i)$ by the preceding lemma, and so $v \models \Box C_i(H_i)$. Thus if $w \geq v$, then $v \models \Box C_i(p) \leftrightarrow \Box C_i(H_i)$, and so $w \models \boxdot(\Box C_i(p) \to \Box C_i(H_i))$.

27.17 Lemma. $w \models \boxdot(p \leftrightarrow A) \to \boxdot(\Box C_i(H_i) \to \Box C_i(p))$.

Proof: Suppose $w \models \boxdot(p \leftrightarrow A)$, $w \geq v$, and $v \models {\sim}\Box C_i(p)$. Then there exist u with $v \geq u$ and therefore $w \geq u$ with $u \models {\sim}C_i(p)$. Take $u \leq v$ of least rank among those such that $u \models {\sim}C_i(p)$. Then for all t with $u > t$, we have $t \models C_i(p)$. Thus $u \models \Box C_i(p)$. As in the proof of Lemma 27.16, $u \models \boxdot(p \leftrightarrow A)$, and so by that lemma, $u \models C_i(p) \leftrightarrow C_i(H_i)$ and $u \models {\sim}C_i(H_i)$. Thus $v \models {\sim}\Box C_i(H_i)$ and $v \models \Box C_i(H_i) \to \Box C_i(p)$ and $w \models \boxdot(\Box C_i(H_i) \to \Box C_i(p))$.

The last two lemmas together tell us that

$$\boxdot(p \leftrightarrow A) \to \boxdot(\Box C_i(H_i) \leftrightarrow \Box C_i(p))$$

is a theorem of **GL**. By repeated application of Proposition 27.5, we successively see that $\boxdot(p \leftrightarrow A) \to \boxdot(A \leftrightarrow D)$ and therefore $\boxdot(p \leftrightarrow A) \to \boxdot(p \leftrightarrow D)$ is a theorem of

GL for all the following sentences D, of which the first is A and the last H:

$$\begin{aligned}
&B(\Box C_1(p), \Box C_2(p), \ldots, \Box C_{n+1}(p))\\
&B(\Box C_1(H_1), \Box C_2(p), \ldots, \Box C_{n+1}(p))\\
&B(\Box C_1(H_1), \Box C_2(H_2), \ldots, \Box C_{n+1}(p))\\
&\vdots\\
&B(\Box C_1(H_1), \Box C_2(H_2), \ldots, \Box C_{n+1}(H_{n+1})).
\end{aligned}$$

Thus $\boxdot(p \leftrightarrow A) \to (p \leftrightarrow H)$ is a theorem of **GL**, to complete the proof of the fixed point theorem.

The normal form and fixed point theorems are only two of the many results about **GL** and related systems that have been obtained in the branch of logical studies known as *provability logic*.

Problems

27.1 Prove the cases of Theorem 27.1 that were 'left to the reader'.

27.2 Let $\mathbf{S5} = \mathbf{K} + (\text{A1}) + (\text{A2}) + (\text{A3})$. Introduce an alternative notion of model for **S5** in which a model is just a pair $\mathcal{W} = (W, \omega)$ and $\mathcal{W}, w \models \Box A$ iff $\mathcal{W}, v \models A$ for all v in W. Show that **S5** is sound and complete for this notion of model.

27.3 Show that in **S5** every formula is provably equivalent to one such that in a subformula of form $\Box A$, there are no occurrences of $\Box$ in A.

27.4 Show that there is an *infinite* transitive, irreflexive model in which the sentence $\Box(\Box p \to p) \to \Box p$ is *not* valid.

27.5 Verify the entries in Table 27-1.

27.6 Suppose for A in Table 27-1 we took $\Box({\sim}p \to \Box\bot) \to \Box(p \to \Box\bot)$. What would be the corresponding H?

27.7 To prove that the Gödel sentence is not provable in **P**, we have to assume the consistency of **P**. To prove that the *negation* of the Gödel sentence is not provable in **P**, we assumed in Chapter 17 the ω-consistency of **P**. This is a stronger assumption than is really needed for the proof. According to Table 27-1, what assumption is just strong enough?

Annotated Bibliography

General Reference Works

BARWISE, JON (1977) (ed.), *Handbook of Mathematical Logic* (Amsterdam: North Holland). A collection of survey articles with references to further specialist literature, the last article being an exposition of the Paris–Harrington theorem.

GABBAY, DOV, and GUENTHNER, FRANZ (1983) (eds.), *Handbook of Philosophical Logic* (4 vols.) (Dordrecht: Reidel). A collection of survey articles covering classical logic, modal logic and allied subjects, and the relation of logical theory to natural language. Successive volumes of an open-ended, much-expanded second edition have been appearing since 2001.

VAN HEIJENOORT, JEAN (1967) (ed.), *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931* (Cambridge, Massachusetts: Harvard University Press). A collection of classic papers showing the development of the subject from the origins of truly modern logic through the incompleteness theorems.

Textbooks and Monographs

ENDERTON, HERBERT (2001), *A Mathematical Introduction to Logic*, 2nd ed. (New York: Harcourt/ Academic Press). An undergraduate textbook directed especially to students of mathematics and allied fields.

KLEENE, STEVEN COLE (1950), *Introduction to Metamathematics* (Princeton: D. van Nostrand). The text from which many of the older generation first learned the subject, containing many results still not readily found elsewhere.

SHOENFIELD, JOSEPH R. (1967), *Mathematical Logic* (Reading, Massachusetts: Addison-Wesley). The standard graduate-level text in the field.

TARSKI, ALFRED, MOSTOWSKI, ANDRZEJ, and ROBINSON, RAPHAEL (1953), *Undecidable Theories* (Amsterdam: North Holland). A treatment putting Gödel's first incompleteness theorem in its most general formulation.

By the Authors

BOOLOS, GEORGE S. (1993), *The Logic of Provability* (Cambridge, U.K.: Cambridge University Press). A detailed account of work on the modal approach to provability and unprovability introduced in the last chapter of this book.

JEFFREY, RICHARD C. (1991), *Formal Logic: Its Scope and Limits*, 4th ed. (Indianapolis: Hackett). An introductory textbook, supplying more than enough background for this book.

Index