In this theorem, t plays the role of the transformation, and F is the fixed point.

PROOF Let F be the following Turing machine.

```
F = "On input w:
    1. Obtain, via the recursion theorem, own description ⟨F⟩.
    2. Compute t(⟨F⟩) to obtain the description of a TM G.
    3. Simulate G on w."
```

Clearly, $\langle F\rangle$ and $t(\langle F\rangle) = \langle G\rangle$ describe equivalent Turing machines because F simulates G.

6.2 DECIDABILITY OF LOGICAL THEORIES

Mathematical logic is the branch of mathematics that investigates mathematics itself. It addresses questions such as: What is a theorem? What is a proof? What is truth? Can an algorithm decide which statements are true? Are all true statements provable? We'll touch on a few of these topics in our brief introduction to this rich and fascinating subject.

We focus on the problem of determining whether mathematical statements are true or false and investigate the decidability of this problem. The answer depends on the domain of mathematics from which the statements are drawn. We examine two domains: one for which we can give an algorithm to decide truth and another for which this problem is undecidable.

First we need to set up a precise language to formulate these problems. Our intention is to be able to consider mathematical statements such as

1. $\forall q\, \exists p\, \forall x,y\ \big[p{>}q \wedge (x,y{>}1 \rightarrow xy{\neq}p)\big]$,
2. $\forall a,b,c,n\ \big[(a,b,c{>}0 \wedge n{>}2) \rightarrow a^n{+}b^n{\neq}c^n\big]$, and
3. $\forall q\, \exists p\, \forall x,y\ \big[p{>}q \wedge (x,y{>}1 \rightarrow (xy{\neq}p \wedge xy{\neq}p{+}2))\big]$.

Statement 1 says that infinitely many prime numbers exist, which has been known to be true since the time of Euclid, about 2,300 years ago. Statement 2 is *Fermat's last theorem*, which been known to be true only since Andrew Wiles proved it a few years ago. Finally, statement 3 says that infinitely many prime pairs[1] exist. Known as the *twin prime conjecture*, it remains unsolved.

To consider whether we could automate the process of determining which of these statements are true, we treat such statements merely as strings and define a language consisting of those statements that are true. Then we ask whether this language is decidable.

[1]***Prime pairs*** are primes that differ by 2.

To make this a bit more precise, let's describe the form of the alphabet of this language:

$$\{\wedge, \vee, \neg, (,), \forall, x, \exists, R_1, \ldots, R_k\}.$$

The symbols $\wedge$, $\vee$, and $\neg$, are called ***Boolean operations***; "(" and ")" are the ***parentheses***; the symbols $\forall$ and $\exists$ are called ***quantifiers***; the symbol x is used to denote ***variables***;[2] and the symbols $R_1, \ldots, R_l$ are called ***relations***.

A ***formula*** is a well-formed string over this alphabet. For completeness, we'll sketch the technical but obvious definition of a ***well-formed formula*** here, but feel free to skip this part and go on to the next paragraph. A string of the form $R_i(x_1, \ldots, x_j)$ is an ***atomic formula***. The value j is the ***arity*** of the relation symbol R_i. All appearances of the same relation symbol in a well-formed formula must have the same arity. Subject to this requirement a string ϕ is a formula if it

1. is an atomic formula,
2. has the form $\phi_1 \wedge \phi_2$ or $\phi_1 \vee \phi_2$ or $\neg\phi_1$, where ϕ_1 and ϕ_2 are smaller formulas, or
3. has the form $\exists x_i\,[\,\phi_1\,]$ or $\forall x_i\,[\,\phi_1\,]$, where ϕ_1 is a smaller formula.

A quantifier may appear anywhere in a mathematical statement. Its ***scope*** is the fragment of the statement appearing within the matched pair of parentheses or brackets following the quantified variable. We assume that all formulas are in ***prenex normal form***, where all quantifiers appear in the front of the formula. A variable that isn't bound within the scope of a quantifier is called a ***free variable***. A formula with no free variables is called a ***sentence*** or ***statement***.

EXAMPLE 6.9

Among the following examples of formulas, only the last one is a sentence.

1. $R_1(x_1) \wedge R_2(x_1, x_2, x_3)$
2. $\forall x_1\,\big[\,R_1(x_1) \wedge R_2(x_1, x_2, x_3)\,\big]$
3. $\forall x_1\,\exists x_2\,\exists x_3\,\big[\,R_1(x_1) \wedge R_2(x_1, x_2, x_3)\,\big]$.

Having established the syntax of formulas, let's discuss their meanings. The Boolean operations and the quantifiers have their usual meanings, but to determine the meaning of the variables and relation symbols we need to specify two items. One is the ***universe*** over which the variables may take values. The other is an assignment of specific relations to the relation symbols. As we described in Section 0.2 (page 8), a relation is a function from k-tuples over the universe to $\{\text{TRUE}, \text{FALSE}\}$. The arity of a relation symbol must match that of its assigned relation.

[2]If we need to write several variables in a formula, we use the symbols w, y, z, or x_1, x_2, x_3, and so on. We don't list all the infinitely many possible variables in the alphabet to keep the alphabet finite. Instead, we list only the variable symbol x, and use strings of x's to indicate other variables, as in xx for x_2, xxx for x_3, and so on.

A universe together with an assignment of relations to relation symbols is called a ***model***.[3] Formally we say that a model $\mathcal{M}$ is a tuple $(U, P_1, \ldots, P_k)$, where U is the universe and P_1 through P_k are the relations assigned to symbols R_1 through R_k. We sometimes refer to the ***language of a model*** to be the collection of formulas that use only the relation symbols the model assigns and that use each relation symbol with the correct arity. If ϕ is a sentence in the language of a model, ϕ is either true or false in that model. If ϕ is true in a model $\mathcal{M}$, we say that $\mathcal{M}$ is a model of ϕ.

If you feel overwhelmed by these definitions, concentrate on our objective in stating them. We want to set up a precise language of mathematical statements so that we can ask whether an algorithm can determine which are true and which are false. The following two examples should be helpful.

EXAMPLE 6.10

Let ϕ be the sentence $\forall x \, \forall y \, [\, R_1(x, y) \lor R_1(y, x) \,]$. Let model $\mathcal{M}_1 = (\mathcal{N}, \leq)$ be the model whose universe is the natural numbers and which assigns the "less than or equal" relation to the symbol R_1. Obviously, ϕ is true in model $\mathcal{M}_1$ because either $a \leq b$ or $b \leq a$ for any two natural numbers a and b. However, if $\mathcal{M}_1$ assigned "less than" instead of "less than or equal" to R_1, then ϕ would not be true because it fails when x and y are equal.

If we know in advance which relation will be assigned to R_i, we may use the customary symbol for that relation in place of R_i with infix notation rather than prefix notation if customary for that symbol. Thus with model $\mathcal{M}_1$ in mind, we could write ϕ as $\forall x \, \forall y \, [\, x \leq y \lor y \leq x \,]$.

EXAMPLE 6.11

Now let $\mathcal{M}_2$ be the model whose universe is the real numbers $\mathcal{R}$ and that assigns the relation *PLUS* to R_1, where $PLUS(a, b, c) = \text{TRUE}$ whenever $a + b = c$. Then $\mathcal{M}_2$ is a model of $\psi = \forall y \, \exists x \, [\, R_1(x, x, y) \,]$. However, if $\mathcal{N}$ were used for the universe instead of $\mathcal{R}$ in $\mathcal{M}_2$, the sentence would be false.

As in Example 6.10, we may write ψ as $\forall y \, \exists x \, [\, x + x = y \,]$ in place of $\forall y \, \exists x \, [\, R_1(x, x, y) \,]$ when we know in advance that we will be assigning the addition relation to R_1.

As Example 6.11 illustrates, we can represent functions such as the addition function by relations. Similarly, we can represent constants such as 0 and 1 by relations.

Now we give one final definition in preparation for the next section. If $\mathcal{M}$ is a model, we let the ***theory of $\mathcal{M}$***, written $\text{Th}(\mathcal{M})$, be the collection of true sentences in the language of that model.

[3] A model is also variously called an ***interpretation*** or a ***structure***.

A DECIDABLE THEORY

Number theory is one of the oldest branches of mathematics and also one of its most difficult. Many innocent looking statements about the natural numbers with the plus and times operations have confounded mathematicians for centuries, such as the twin prime conjecture mentioned earlier.

In one of the celebrated developments in mathematical logic, Alonzo Church, building on the work of Kurt Gödel, showed that no algorithm can decide in general whether statements in number theory are true or false. Formally, we write $(\mathcal{N}, +, \times)$ to be the model whose universe is the natural numbers[4] with the usual $+$ and $\times$ relations. Church showed that $\text{Th}(\mathcal{N}, +, \times)$, the theory of this model, is undecidable.

Before looking at this undecidable theory, let's examine one that is decidable. Let $(\mathcal{N}, +)$ be the same model, without the $\times$ relation. Its theory is $\text{Th}(\mathcal{N}, +)$. For example, the formula $\forall x\, \exists y\ \big[\, x + x = y \,\big]$ is true and is therefore a member of $\text{Th}(\mathcal{N}, +)$, but the formula $\exists y \forall x\ \big[\, x + x = y \,\big]$ is false and is therefore not a member.

THEOREM 6.12

$\text{Th}(\mathcal{N}, +)$ is decidable.

PROOF IDEA This proof is an interesting and nontrivial application of the theory of finite automata that we presented in Chapter 1. One fact about finite automata that we use appears in Problem 1.32 (page 88) where you were asked to show that they are capable of doing addition if the input is presented in a special form. The input describes three numbers in parallel, by representing one bit of each number in a single symbol from an eight-symbol alphabet. Here we use a generalization of this method to present i-tuples of numbers in parallel using an alphabet with 2^i symbols.

We give an algorithm that can determine whether its input, a sentence ϕ in the language of $(\mathcal{N}, +)$, is true in that model. Let

$$\phi = \mathsf{Q}_1 x_1\, \mathsf{Q}_2 x_2 \cdots \mathsf{Q}_l x_l\ \big[\, \psi \,\big],$$

where $\mathsf{Q}_1, \ldots, \mathsf{Q}_l$ each represent either $\exists$ or $\forall$ and ψ is a formula without quantifiers that has variables $x_1, \ldots, x_l$. For each i from 0 to l, define formula ϕ_i to be

$$\phi_i = \mathsf{Q}_{i+1} x_{i+1}\, \mathsf{Q}_{i+2} x_{i+2} \cdots \mathsf{Q}_l x_l\ \big[\, \psi \,\big].$$

Thus $\phi_0 = \phi$ and $\phi_l = \psi$.

Formula ϕ_i has i free variables. For $a_1, \ldots, a_i \in \mathcal{N}$ write $\phi_i(a_1, \ldots, a_i)$ to be the sentence obtained by substituting the constants $a_1, \ldots, a_i$ for the variables $x_1, \ldots, x_i$ in ϕ_i.

For each i from 0 to l, the algorithm constructs a finite automaton A_i that

[4]For convenience in this chapter, we change our usual definition of $\mathcal{N}$ to be $\{0, 1, 2, \ldots\}$.

recognizes the collection of strings representing i-tuples of numbers that make ϕ_i true. The algorithm begins by constructing A_l directly, using a generalization of the method in the solution to Problem 1.32. Then, for each i from l down to 1, it uses A_i to construct A_{i-1}. Finally, once the algorithm has A_0, it tests whether A_0 accepts the empty string. If it does, ϕ is true and the algorithm accepts.

PROOF For $i > 0$ define the alphabet

$$\Sigma_i = \left\{ \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 1 \end{bmatrix}, \ldots, \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} \right\}.$$

Hence Σ_i contains all size i columns of 0s and 1s. A string over Σ_i represents i binary integers (reading across the rows). We also define $\Sigma_0 = \{[\,]\}$, where $[\,]$ is a symbol.

We now present an algorithm that decides Th$(\mathcal{N}, +)$. On input ϕ where ϕ is a sentence, the algorithm operates as follows. Write ϕ and define ϕ_i for each i from 0 to l, as in the proof idea. For each such i construct a finite automaton A_i from ϕ_i that accepts strings over Σ_i^* corresponding to i-tuples $a_1, \ldots, a_i$ whenever $\phi_i(a_1, \ldots, a_i)$ is true, as follows.

To construct the first machine A_l, observe that $\phi_l = \psi$ is a Boolean combination of atomic formulas. An atomic formula in the language of Th$(\mathcal{N}, +)$ is a single addition. Finite automata can be constructed to compute any of these individual relations corresponding to a single addition and then combined to give the automaton A_l. Doing so involves the use of the regular language closure constructions for union, intersection, and complementation to compute Boolean combinations of the atomic formulas.

Next, we show how to construct A_i from A_{i+1}. If $\phi_i = \exists x_{i+1}\ \phi_{i+1}$, we construct A_i to operate as A_{i+1} operates, except that it nondeterministically guesses the value of a_{i+1} instead of receiving it as part of the input.

More precisely, A_i contains a state for each A_{i+1} state and a new start state. Every time A_i reads a symbol

$$\begin{bmatrix} b_1 \\ \vdots \\ b_{i-1} \\ b_i \end{bmatrix},$$

where every $b_i \in \{0,1\}$ is a bit of the number a_i, it nondeterministically guesses $z \in \{0,1\}$ and simulates A_{i+1} on the input symbol

$$\begin{bmatrix} b_1 \\ \vdots \\ b_{i-1} \\ b_i \\ z \end{bmatrix}.$$

Initially, A_i nondeterministically guesses the leading bits of z corresponding to suppressed leading 0s in b_1 through b_i by nondeterministically branching from its new start state to all states that A_{i+1} could reach from its start state with input strings of the symbols

$$\left\{ \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \right\}$$

in Σ_{i+1}. Clearly, A_i accepts its input $(a_1, \ldots, a_i)$ if some a_{i+1} exists where A_{i+1} accepts $(a_1, \ldots, a_{i+1})$.

If $\phi_i = \forall x_{i+1}\ \phi_{i+1}$, it is equivalent to $\neg\exists x_{i+1}\neg\ \phi_{i+1}$. Thus we can construct the finite automaton that recognizes the complement of the language of A_{i+1}, then apply the preceding construction for the $\exists$ quantifier, and finally apply complementation once again to obtain A_i.

Finite automaton A_0 accepts any input iff ϕ_0 is true. So the final step of the algorithm tests whether A_0 accepts ε. If it does, ϕ is true and the algorithm accepts; otherwise, it rejects.

AN UNDECIDABLE THEORY

As we mentioned earlier, $\text{Th}(\mathcal{N}, +, \times)$ is an undecidable theory. No algorithm exists for deciding the truth or falsity of mathematical statements, even when restricted to the language of $(\mathcal{N}, +, \times)$. This theorem has great importance philosophically because it demonstrates that mathematics cannot be mechanized. We state this theorem, but give only a brief sketch of its proof.

THEOREM 6.13

$\text{Th}(\mathcal{N}, +, \times)$ is undecidable.

Although it contains many details, the proof of this theorem is not difficult conceptually. It follows the pattern of the other proofs of undecidability presented in Chapter 4. We show that $\text{Th}(\mathcal{N}, +, \times)$ is undecidable by reducing A_{TM} to it, using the computation history method as previously described (page 192). The existence of the reduction depends on the following lemma.

LEMMA 6.14

Let M be a Turing machine and w a string. We can construct from M and w a formula $\phi_{M,w}$ in the language of $\text{Th}(\mathcal{N}, +, \times)$ that contains a single free variable x, whereby the sentence $\exists x\ \phi_{M,w}$ is true iff M accepts w.

PROOF IDEA Formula $\phi_{M,w}$ "says" that x is a (suitably encoded) accepting computation history of M on w. Of course, x actually is just a rather large integer, but it represents a computation history in a form that can be checked by using the $+$ and $\times$ operations.

The actual construction of $\phi_{M,w}$ is too complicated to present here. It extracts individual symbols in the computation history with the $+$ and $\times$ operations to check: the start configuration for M on w; that each configuration legally follows from the one preceding it; and finally that the last configuration is accepting.

PROOF OF THEOREM 6.13 We give a mapping reduction from A_{TM} to $\mathrm{Th}(\mathcal{N}, +, \times)$. The reduction constructs the formula $\phi_{M,w}$ from the input $\langle M, w\rangle$, using Lemma 6.14. Then it outputs the sentence $\exists x\ \phi_{M,w}$.

Next, we sketch the proof of Kurt Gödel's celebrated *incompleteness theorem*. Informally, this theorem says that, in any reasonable system of formalizing the notion of provability in number theory, some true statements are unprovable.

Loosely speaking, the ***formal proof*** π of a statement ϕ is a sequence of statements, $S_1, S_2, \ldots, S_l$, where $S_l = \phi$. Each S_i follows from the preceding statements and certain basic axioms about numbers, using simple and precise rules of implication. We don't have space to define the concept of proof, but for our purposes assuming the following two reasonable properties of proofs will be enough.

1. The correctness of a proof of a statement can be checked by machine. Formally, $\{\langle\phi, \pi\rangle \mid \pi \text{ is a proof of } \phi\}$ is decidable.
2. The system of proofs is *sound*. That is, if a statement is provable (i.e., has a proof), it is true.

If a system of provability satisfies these two conditions, the following three theorems hold.

THEOREM 6.15

The collection of provable statements in $\mathrm{Th}(\mathcal{N}, +, \times)$ is Turing-recognizable.

PROOF The following algorithm P accepts its input ϕ if ϕ is provable. Algorithm P tests each string as a candidate for a proof π of ϕ, using the proof checker assumed in provability property 1. If it finds that any of these candidates is a proof, it accepts.

Now we can use the preceding theorem to prove our version of the incompleteness theorem.

THEOREM 6.16

Some true statement in $\text{Th}(\mathcal{N}, +, \times)$ is not provable.

PROOF We give a proof by contradiction. We assume to the contrary that all true statements are provable. Using this assumption, we describe an algorithm D that decides whether statements are true, contradicting Theorem 6.13.

On input ϕ algorithm D operates by running algorithm P given in the proof of Theorem 6.15 in parallel on inputs ϕ and $\neg\phi$. One of these two statements is true and thus by our assumption is provable. Therefore P must halt on one of the two inputs. By provability property 2, if ϕ is provable, then ϕ is true, and if $\neg\phi$ is provable, then ϕ is false. So algorithm D can decide the truth or falsity of ϕ.

In the final theorem of this section we use the recursion theorem to give an explicit sentence in the language of $(\mathcal{N}, +, \times)$ that is true but not provable. In Theorem 6.16 we demonstrated the existence of such a sentence but didn't actually describe one, as we do now.

THEOREM 6.17

The sentence $\psi_{\text{unprovable}}$, as described in the proof of this theorem, is unprovable.

PROOF IDEA Construct a sentence that says: "This sentence is not provable," using the recursion theorem to obtain the self-reference.

PROOF Let S be a TM that operates as follows.

```
S = "On any input:
    1. Obtain own description ⟨S⟩ via the recursion theorem.
    2. Construct the sentence $\psi = \neg\exists c\, [\, \phi_{S,0} \,]$, using Lemma 6.14.
    3. Run algorithm P from the proof of Theorem 6.15 on input $\psi$.
    4. If stage 3 accepts, accept. If it halts and rejects, reject."
```

Let $\psi_{\text{unprovable}}$ be the sentence ψ described in stage 2 of algorithm S. That sentence is true iff S doesn't accept 0 (the string 0 was selected arbitrarily).

If S finds a proof of $\psi_{\text{unprovable}}$, S accepts 0, and the sentence would thus be false. A false sentence cannot be provable, so this situation cannot occur. The only remaining possibility is that S fails to find a proof of $\psi_{\text{unprovable}}$ and so S doesn't accept 0. But then $\psi_{\text{unprovable}}$ is true, as we claimed.

EXERCISES

6.1 Give an example in the spirit of the recursion theorem of a program in a real programming language (or a reasonable approximation thereof) that prints itself out.

6.2 Show that any infinite subset of MIN_{TM} is not Turing-recognizable.

$^{\mathsf{A}}$6.3 Show that if $A \leq_{\mathrm{T}} B$ and $B \leq_{\mathrm{T}} C$ then $A \leq_{\mathrm{T}} C$.

6.4 Let $A_{\mathsf{TM}}{}' = \{\langle M, w\rangle \mid M$ is an oracle TM and $M^{A_{\mathsf{TM}}}$ accepts $w\}$. Show that $A_{\mathsf{TM}}{}'$ is undecidable relative to A_{TM}.

$^{\mathsf{A}}$6.5 Is the statement $\exists x\,\forall y\,\big[\,x{+}y{=}y\,\big]$ a member of Th$(\mathcal{N}, +)$? Why or why not? What about the statement $\exists x\,\forall y\,\big[\,x{+}y{=}x\,\big]$?

PROBLEMS

6.6 Describe two different Turing machines, M and N, that, when started on any input, M outputs $\langle N\rangle$ and N outputs $\langle M\rangle$.

6.7 In the fixed-point version of the recursion theorem (Theorem 6.8) let the transformation t be a function that interchanges the states q_{accept} and q_{reject} in Turing machine descriptions. Give an example of a fixed point for t.

*6.8 Show that $EQ_{\mathrm{TM}} \not\leq_{\mathrm{m}} \overline{EQ_{\mathrm{TM}}}$.

$^{\mathsf{A}}$6.9 Use the recursion theorem to give an alternative proof of Rice's theorem in Problem 5.28.

$^{\mathsf{A}}$6.10 Give a model of the sentence

$$\begin{aligned}\phi_{\text{eq}} = \quad & \forall x\,\big[\,R_1(x,x)\,\big] \\ & \land \forall x{,}y\big[\,R_1(x,y) \leftrightarrow R_1(y,x)\,\big] \\ & \land \forall x{,}y{,}z\big[\,(R_1(x,y) \land R_1(y,z)) \rightarrow R_1(x,z)\,\big].\end{aligned}$$

*6.11 Let ϕ_{eq} be defined as in Problem 6.10. Give a model of the sentence

$$\begin{aligned}\phi_{\text{lt}} = \quad & \phi_{\text{eq}} \\ & \land \forall x{,}y\,\big[\,R_1(x,y) \rightarrow \neg R_2(x,y)\,\big] \\ & \land \forall x{,}y\,\big[\,\neg R_1(x,y) \rightarrow (R_2(x,y) \oplus R_2(y,x))\,\big] \\ & \land \forall x{,}y{,}z\,\big[\,(R_2(x,y) \land R_2(y,z)) \rightarrow R_2(x,z)\,\big] \\ & \land \forall x\,\exists y\,\big[\,R_2(x,y)\,\big].\end{aligned}$$

$^{\mathsf{A}}$6.12 Let $(\mathcal{N}, <)$ be the model with universe $\mathcal{N}$ and the "less than" relation. Show that Th$(\mathcal{N}, <)$ is decidable.

6.13 For each $m > 1$ let $\mathcal{Z}_m = \{0, 1, 2, \ldots, m-1\}$ and let $\mathcal{F}_m = (\mathcal{Z}_m, +, \times)$ be the model whose universe is $\mathcal{Z}_m$ and that has relations corresponding to the $+$ and $\times$ relations computed modulo m. Show that for each m the theory $\text{Th}(\mathcal{F}_m)$ is decidable.

6.14 Show that for any two languages A and B a language J exists, where $A \leq_{\text{T}} J$ and $B \leq_{\text{T}} J$.

6.15 Show that for any language A, a language B exists, where $A \leq_{\text{T}} B$ and $B \not\leq_{\text{T}} A$.

***6.16** Prove that there exist two languages A and B that are Turing-incomparable—that is, where $A \not\leq_{\text{T}} B$ and $B \not\leq_{\text{T}} A$.

***6.17** Let A and B be two disjoint languages. Say that language C *separates* A and B if $A \subseteq C$ and $B \subseteq \overline{C}$. Describe two disjoint Turing-recognizable languages that aren't separable by any decidable language.

6.18 In Corollary 4.18 we showed that the set of all languages is uncountable. Use this result to prove that languages exist that are not recognizable by an oracle Turing machine with oracle for A_{TM}.

6.19 Recall the Post correspondence problem that we defined in Section 5.2 and its associated language PCP. Show that PCP is decidable relative to A_{TM}.

6.20 Show how to compute the descriptive complexity of strings $\text{K}(x)$ with an oracle for A_{TM}.

6.21 Use the result of Problem 6.20 to give a function f that is computable with an oracle for A_{TM}, where for each n, $f(n)$ is an incompressible string of length n.

6.22 Show that the function $\text{K}(x)$ is not a computable function.

6.23 Show that the set of incompressible strings is undecidable.

6.24 Show that the set of incompressible strings contains no infinite subset that is Turing-recognizable.

***6.25** Show that for any c, some strings x and y exist, where $\text{K}(xy) > \text{K}(x) + \text{K}(y) + c$.

SELECTED SOLUTIONS

6.3 Say that M_1^B decides A and M_2^C decides B. Use an oracle TM M_3, where M_3^C decides A. Machine M_3 simulates M_1. Every time M_1 queries its oracle about some string x, machine M_3 tests whether $x \in B$ and provides the answer to M_1. Because machine M_3 doesn't have an oracle for B and cannot perform that test directly, it simulates M_2 on input x to obtain that information. Machine M_3 can obtain the answer to M_2's queries directly because these two machines use the same oracle, C.

6.5 The statement $\exists x\, \forall y\, \big[\, x{+}y{=}y \,\big]$ is a member of $\text{Th}(\mathcal{N}, +)$ because that statement is true for the standard interpretation of $+$ over the universe $\mathcal{N}$. Recall that we use $\mathcal{N} = \{0, 1, 2, \ldots\}$ in this chapter and so we may use $x = 0$. The statement $\exists x\, \forall y\, \big[\, x{+}y{=}x \,\big]$ is not a member of $\text{Th}(\mathcal{N}, +)$ because that statement isn't true in this model. For any value of x, setting $y = 1$ causes $x{+}y{=}x$ to fail.

6.9 Assume for the sake of contradiction that some TM X decides a property P, and P satisfies the conditions of Rice's theorem. One of these conditions says that TMs A and B exist where $\langle A\rangle \in P$ and $\langle B\rangle \notin P$. Use A and B to construct TM R:

```
R = "On input w:
    1. Obtain own description ⟨R⟩ using the recursion theorem.
    2. Run X on ⟨R⟩.
    3. If X accepts ⟨R⟩, simulate B on w.
       If X rejects ⟨R⟩, simulate A on w."
```

If $\langle R\rangle \in P$, then X accepts $\langle R\rangle$ and $L(R) = L(B)$. But $\langle B\rangle \notin P$, contradicting $\langle R\rangle \in P$, because P agrees on TMs that have the same language. We arrive at a similar contradiction if $\langle R\rangle \notin P$. Therefore our original assumption is false. Every property satisfying the conditions of Rice's theorem is undecidable.

6.10 The statement ϕ_{eq} gives the three conditions of an equivalence relation. A model (A, R_1), where A is any universe and R_1 is any equivalence relation over A, is a model of ϕ_{eq}. For example, let A be the integers $\mathcal{Z}$ and let $R_1 = \{(i,i) |\, i \in \mathcal{Z}\}$.

6.12 Reduce $\text{Th}(\mathcal{N},<)$ to $\text{Th}(\mathcal{N},+)$, which we've already shown to be decidable. To do so, show how to convert a sentence ϕ_1 over the language of $\text{Th}(\mathcal{N},<)$, to a sentence ϕ_2 over the language of $\text{Th}(\mathcal{N},+)$ while preserving truth or falsity in the respective models. Replace every occurrence of $i < j$ in ϕ_1 by the formula $\exists k\big[\,(i{+}k{=}j) \land (k{+}k{\neq}k)\,\big]$ in ϕ_2, where k is a different new variable each time.

Sentence ϕ_2 is equivalent to ϕ_1 because "i is less than j" means that we can add a nonzero value to i and obtain j. Putting ϕ_2 into prenex-normal form, as required by the algorithm for deciding $\text{Th}(\mathcal{N},+)$, requires a bit of additional work. The new existential quantifiers are brought to the front of the sentence. To do so, these quantifiers must pass through Boolean operations that appear in the sentence. Quantifiers can be brought through the operations of $\land$ and $\lor$ without change. Passing through $\neg$ changes $\exists$ to $\forall$ and vice-versa. Thus $\neg\exists k\,\psi$ becomes the equivalent expression $\forall k\,\neg\psi$, and $\neg\forall k\,\psi$ becomes $\exists k\,\neg\psi$.

SELECTED BIBLIOGRAPHY

1. ADLEMAN, L. Two theorems on random polynomial time. In *Proceedings of the Nineteenth IEEE Symposium on Foundations of Computer Science* (1978), pp. 75–83.
2. ADLEMAN, L. M., AND HUANG, M. A. Recognizing primes in random polynomial time. In *Proceedings of the Nineteenth Annual ACM Symposium on the Theory of Computing* (1987), pp. 462–469.
3. ADLEMAN, L. M., POMERANCE, C., AND RUMELY, R. S. On distinguishing prime numbers from composite numbers. *Annals of Mathematics 117* (1983), 173–206.
4. AGRAWAL, M., KAYAL, N., AND SAXENA, N. PRIMES is in P. (2002), `http://www.cse.iitk.ac.in/news/primality.pdf`.
5. AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *Data Structures and Algorithms*. Addison-Wesley, 1982.
6. AHO, A. V., SETHI, R., AND ULLMAN, J. D. *Compilers: Principles, Techniques, Tools*. Addison-Wesley, 1986.
7. AKL, S. G. *The Design and Analysis of Parallel Algorithms*. Prentice-Hall International, 1989.
8. ALON, N., ERDÖS, P., AND SPENCER, J. H. *The Probabilistic Method*. John Wiley & Sons, 1992.
9. ANGLUIN, D., AND VALIANT, L. G. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *Journal of Computer and System Sciences 18* (1979), 155–193.
10. ARORA, S., LUND, C., MOTWANI, R., SUDAN, M., AND SZEGEDY, M. Proof verification and hardness of approximation problems. In *Proceedings of the Thirty-third IEEE Symposium on Foundations of Computer Science* (1992), pp. 14–23.
11. BAASE, S. *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley, 1978.
12. BABAI, L. E-mail and the unexpected power of interaction. In *Proceedings of the Fifth Annual Conference on Structure in Complexity Theory* (1990), pp. 30–44.
13. BACH, E., AND SHALLIT, J. *Algorithmic Number Theory, Vol. 1*. MIT Press, 1996.

14. BALCÁZAR, J. L., DÍAZ, J., AND GABARRÓ, J. *Structural Complexity I, II.* EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1988 (I) and 1990 (II).

15. BEAME, P. W., COOK, S. A., AND HOOVER, H. J. Log depth circuits for division and related problems. *SIAM Journal on Computing 15*, 4 (1986), 994–1003.

16. BLUM, M., CHANDRA, A., AND WEGMAN, M. Equivalence of free boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters 10* (1980), 80–82.

17. BRASSARD, G., AND BRATLEY, P. *Algorithmics: Theory and Practice.* Prentice-Hall, 1988.

18. CARMICHAEL, R. D. On composite numbers p which satisfy the Fermat congruence $a^{p-1} \equiv p$. *American Mathematical Monthly 19* (1912), 22–27.

19. CHOMSKY, N. Three models for the description of language. *IRE Trans. on Information Theory 2* (1956), 113–124.

20. COBHAM, A. The intrinsic computational difficulty of functions. In *Proceedings of the International Congress for Logic, Methodology, and Philosophy of Science*, Y. Bar-Hillel, Ed. North-Holland, 1964, pp. 24–30.

21. COOK, S. A. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on the Theory of Computing* (1971), pp. 151–158.

22. CORMEN, T., LEISERSON, C., AND RIVEST, R. *Introduction to Algorithms.* MIT Press, 1989.

23. EDMONDS, J. Paths, trees, and flowers. *Canadian Journal of Mathematics 17* (1965), 449–467.

24. ENDERTON, H. B. *A Mathematical Introduction to Logic.* Academic Press, 1972.

25. EVEN, S. *Graph Algorithms.* Pitman, 1979.

26. FELLER, W. *An Introduction to Probability Theory and Its Applications, Vol. 1.* John Wiley & Sons, 1970.

27. FEYNMAN, R. P., HEY, A. J. G., AND ALLEN, R. W. *Feynman lectures on computation.* Addison-Wesley, 1996.

28. GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability—A Guide to the Theory of NP-completeness.* W. H. Freeman, 1979.

29. GILL, J. T. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing 6*, 4 (1977), 675–695.

30. GÖDEL, K. On formally undecidable propositions in *Principia Mathematica* and related systems I. In *The Undecidable*, M. Davis, Ed. Raven Press, 1965, pp. 4–38.

31. GOEMANS, M. X., AND WILLIAMSON, D. P. .878-approximation algorithms for MAX CUT and MAX 2SAT. In *Proceedings of the Twenty-sixth Annual ACM Symposium on the Theory of Computing* (1994), pp. 422–431.

32. GOLDWASSER, S., AND MICALI, S. Probabilistic encryption. *Journal of Computer and System Sciences* (1984), 270–229.

33. GOLDWASSER, S., MICALI, S., AND RACKOFF, C. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing* (1989), 186–208.

34. GREENLAW, R., HOOVER, H. J., AND RUZZO, W. L. *Limits to Parallel Computation: P-completeness Theory*. Oxford University Press, 1995.

35. HARARY, F. *Graph Theory*, 2d ed. Addison-Wesley, 1971.

36. HARTMANIS, J., AND STEARNS, R. E. On the computational complexity of algorithms. *Transactions of the American Mathematical Society 117* (1965), 285–306.

37. HILBERT, D. Mathematical problems. Lecture delivered before the International Congress of Mathematicians at Paris in 1900. In *Mathematical Developments Arising from Hilbert Problems*, vol. 28. American Mathematical Society, 1976, pp. 1–34.

38. HOFSTADTER, D. R. *Goedel, Escher, Bach: An Eternal Golden Braid.* Basic Books, 1979.

39. HOPCROFT, J. E., AND ULLMAN, J. D. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

40. JOHNSON, D. S. The NP-completeness column: Interactive proof systems for fun and profit. *Journal of Algorithms 9*, 3 (1988), 426–444.

41. KARP, R. M. Reducibility among combinatorial problems. In *Complexity of Computer Computations* (1972), R. E. Miller and J. W. Thatcher, Eds., Plenum Press, pp. 85–103.

42. KARP, R. M., AND LIPTON, R. J. Turing machines that take advice. *ENSEIGN: L'Enseignement Mathematique Revue Internationale 28* (1982).

43. LAWLER, E. L. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1991.

44. LAWLER, E. L., LENSTRA, J. K., RINOOY KAN, A. H. G., AND SHMOYS, D. B. *The Traveling Salesman Problem.* John Wiley & Sons, 1985.

45. LEIGHTON, F. T. *Introduction to Parallel Algorithms and Architectures: Array, Trees, Hypercubes.* Morgan Kaufmann, 1991.

46. LEVIN, L. Universal search problems (in Russian). *Problemy Peredachi Informatsii 9*, 3 (1973), 115–116.

47. LEWIS, H., AND PAPADIMITRIOU, C. *Elements of the Theory of Computation.* Prentice-Hall, 1981.

48. LI, M., AND VITANYI, P. *Introduction to Kolmogorov Complexity and its Applications.* Springer-Verlag, 1993.

49. LICHTENSTEIN, D., AND SIPSER, M. GO is PSPACE hard. *Journal of the ACM* (1980), 393–401.

50. LUBY, M. *Pseudorandomness and Cryptographic Applications.* Princeton University Press, 1996.

51. LUND, C., FORTNOW, L., KARLOFF, H., AND NISAN, N. Algebraic methods for interactive proof systems. *Journal of the ACM 39*, 4 (1992), 859–868.

52. MILLER, G. L. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences 13* (1976), 300–317.

53. NIVEN, I., AND ZUCKERMAN, H. S. *An Introduction to the Theory of Numbers*, 4th ed. John Wiley & Sons, 1980.

54. PAPADIMITRIOU, C. H. *Computational Complexity.* Addison-Wesley, 1994.

55. PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial Optimization (Algorithms and Complexity).* Prentice-Hall, 1982.

56. PAPADIMITRIOU, C. H., AND YANNAKAKIS, M. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences 43*, 3 (1991), 425–440.

57. POMERANCE, C. On the distribution of pseudoprimes. *Mathematics of Computation 37*, 156 (1981), 587–593.

58. PRATT, V. R. Every prime has a succinct certificate. *SIAM Journal on Computing 4*, 3 (1975), 214–220.

59. RABIN, M. O. Probabilistic algorithms. In *Algorithms and Complexity: New Directions and Recent Results*, J. F. Traub, Ed. Academic Press, 1976, pp. 21–39.

60. REINGOLD, O. Undirected st-connectivity in log-space. (2004), `http://www.eccc.uni-trier.de/eccc-reports/2004/TR04-094`.

61. RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. A method for obtaining digital signatures and public key cryptosytems. *Communications of the ACM 21*, 2 (1978), 120–126.

62. ROCHE, E., AND SCHABES, Y. *Finite-State Language Processing.* MIT Press, 1997.

63. SCHAEFER, T. J. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences 16*, 2 (1978), 185–225.

64. SEDGEWICK, R. *Algorithms*, 2d ed. Addison-Wesley, 1989.

65. SHAMIR, A. IP = PSPACE. *Journal of the ACM 39*, 4 (1992), 869–877.

66. SHEN, A. IP = PSPACE: Simplified proof. *Journal of the ACM 39*, 4 (1992), 878–880.

67. SHOR, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing 26*, (1997), 1484–1509.

68. SIPSER, M. Lower bounds on the size of sweeping automata. *Journal of Computer and System Sciences 21*, 2 (1980), 195–202.

69. SIPSER, M. The history and status of the P versus NP question. In *Proceedings of the Twenty-fourth Annual ACM Symposium on the Theory of Computing* (1992), pp. 603–618.

70. STINSON, D. R. *Cryptography: Theory and Practice.* CRC Press, 1995.

71. TARJAN, R. E. *Data structures and network algorithms*, vol. 44 of *CBMS-NSF Reg. Conf. Ser. Appl. Math.* SIAM, 1983.

72. TURING, A. M. On computable numbers, with an application to the Entscheidungsproblem. In *Proceedings, London Mathematical Society*, (1936), pp. 230–265.

73. ULLMAN, J. D., AHO, A. V., AND HOPCROFT, J. E. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

74. VAN LEEUWEN, J., Ed. *Handbook of Theoretical Computer Science A: Algorithms and Complexity.* Elsevier, 1990.