THEOREM 6.8

Let $t\colon \Sigma^* \longrightarrow \Sigma^*$ be a computable function. Then there is a Turing machine F for which $t(\langle F \rangle)$ describes a Turing machine equivalent to F. Here we'll assume that if a string isn't a proper Turing machine encoding, it describes a Turing machine that always rejects immediately.

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

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 has been known to be true only since Andrew Wiles proved it in 1994. Finally, statement 3 says that infinitely many prime pairs[1] exist. Known as the *twin prime conjecture*, it remains unsolved.

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

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.

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

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

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 $R_1, \ldots, R_k$ 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 $R_i(x_1, \ldots, x_k)$ 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

[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.

is an assignment of specific relations to the relation symbols. As we described in Section 0.2 (page 9), 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.

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 $\forall x\, \forall y\, \big[\, R_1(x, y) \lor R_1(y, x)\, \big]$. Let model $\mathcal{M}_1 = (\mathcal{N}, \leq)$ be the model whose universe is the natural numbers and that 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 $\forall x\, \forall y\, \big[\, x{\leq}y \lor y{\leq}x\, \big]$. ■

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 $\mathit{PLUS}(a, b, c) = \text{TRUE}$ whenever $a + b = c$. Then $\mathcal{M}_2$ is a model of $\psi = \forall y\, \exists x\, \big[\, R_1(x, x, y)\, \big]$. 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 $\forall y\, \exists x\, \big[\, x + x = y\, \big]$ in place of $\forall y\, \exists x\, \big[\, R_1(x, x, y)\, \big]$ 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.

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

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

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 represents 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 as

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

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

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

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 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

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

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 $\mathrm{Th}(\mathcal{N}, +)$. 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 $\mathrm{Th}(\mathcal{N}, +)$ 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 $b_j \in \{\texttt{0},\texttt{1}\}$ is a bit of the number a_j, it nondeterministically guesses $z \in \{\texttt{0},\texttt{1}\}$ 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 a_{i+1} corresponding to suppressed leading 0s in a_1 through a_i by nondeterministically branching using ε-transitions 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 220). 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 $(\mathcal{N}, +, \times)$ 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 that the start configuration for M on w is correct, that each configuration legally follows from the one preceding it, and that the last configuration is accepting.

PROOF OF THEOREM 6.13 We give a mapping reduction from A_{TM} to $\text{Th}(\mathcal{N}, +, \times)$. The reduction constructs the formula $\phi_{M,w}$ from the input $\langle M, w\rangle$ by 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 $\text{Th}(\mathcal{N}, +, \times)$ 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, 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 $\langle S \rangle$ via the recursion theorem.
2. Construct the sentence $\psi = \neg\exists c\,\left[\,\phi_{S,0}\,\right]$, using Lemma 6.14.
3. Run algorithm P from the proof of Theorem 6.15 on input ψ.
4. If stage 3 accepts, *accept*."

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.