Chapter 10

Applications of Turing Machines: Theory of Computation

In Chapter 9, we have considered Turing machines (TMs) as language acceptors by analogy with other language acceptors, such as finite and pushdown automata, discussed earlier in this book. In this chapter, we make use of TMs to show the fundamentals of *theory of computation*, which is primarily oriented toward determining the theoretical limits of computation in general. This orientation comes as no surprise: by the Church–Turing thesis, which opened Section IV of this book, every procedure can be formalized by a TM, so any computation beyond the power of TMs is also beyond the computer power in general.

This two-section chapter gives an introduction to two key areas of the theory of computation—computability and decidability. In Section 10.1, regarding computability, we view TMs as computers of functions over nonnegative integers and show the existence of functions whose computation cannot be specified by any procedure. Then, regarding decidability, we formalize algorithms that decide problems by using TMs that halt on every input in Section 10.2, which is divided into five subsections. In Section 10.2.1, we conceptualize this approach to decidability. In Section 10.2.2, we formulate several important problems concerning the language models discussed in Chapters 3 and 6, such as finite automata (FAs) and context-free grammars (CFGs), and construct algorithms that decide them. In Section 10.2.3, more surprisingly, we present problems that are algorithmically undecidable, and in Section 10.2.4, we approach undecidability from a more general viewpoint. Finally, in Section 10.2.5, we reconsider algorithms that decide problems in terms of their computational complexity measured according to time and space requirements. Perhaps most importantly, we point out that although some problems are decidable in principle, they are intractable for unreasonably high computational requirements of the algorithms that decide them.

10.1 Computability

Considering the TM model as the formalization of an effective procedure, we show the existence of functions whose computation cannot be specified by any procedure, so they can never be computed by any computer. As a matter of fact, the existence of these uncomputable functions immediately follows from the following counting argument. Consider the set of all functions that map $\mathbb{N}$ onto {0, 1} and the set of all procedures. Whereas the former is uncountable, the latter is countable under our assumption that every procedure has a finite description (see Section 9.1). Thus, there necessarily exist functions with no procedures to compute them. In this section, based on the following TM-based formalization, we take a more specific look at functions whose computation can or, in contrast, cannot be specified by any procedure.

Definition 10.1 Let $M \in {}_{TM}\Psi$. The *function computed by M*, symbolically denoted by *M-f*, is defined over Δ^* as $M\text{-}f = \{(x, y) | \, x, y \in \Delta^*, \triangleright\blacktriangleright x \triangleleft \Rightarrow^* \triangleright\blacksquare yu \triangleleft \text{ in } M, u \in \{\square\}^*\}$. ■

Consider *M-f*, where $M \in {}_{TM}\Psi$, and an argument $x \in \Delta^*$. In a general case, *M-f* is partial, so $M\text{-}f(x)$ may or may not be defined. Clearly, if $M\text{-}f(x) = y$ is defined, M computes $\triangleright\blacktriangleright x \triangleleft \Rightarrow^* \triangleright\blacksquare yu \triangleleft$, where $u \in \{\square\}^*$. However, if $M\text{-}f(x)$ is undefined, M, starting from $\triangleright\blacktriangleright x \triangleleft$, never reaches a configuration of the form $\triangleright\blacksquare vu \triangleleft$, where $v \in \Delta^*$ and $u \in \{\square\}^*$, so it either rejects x or loops on x (see Convention 9.8).

Definition 10.2 A function f is a *computable function* if there exists $M \in {}_{TM}\Psi$ such that $f = M\text{-}f$; otherwise, f is an *uncomputable function*. ■

10.1.1 Integer Functions Computed by Turing Machines

By Definition 10.1, for every $M \in {}_{TM}\Psi$, *M-f* is defined over Δ^*, where Δ is an alphabet. However, in mathematics, we usually study numeric functions defined over sets of infinitely many numbers. To use TMs to compute functions like these, we first need to represent these numbers by strings over Δ. In this introductory book, we restrict our attention only to integer functions over ${}_0\mathbb{N}$, so we need to represent every nonnegative integer $i \in {}_0\mathbb{N}$ as a string over Δ. Traditionally, we represent i in unary as $unary(i)$ for all $i \in {}_0\mathbb{N}$, where *unary* is defined in Example 2.4. That is, $unary(j) = a^j$ for all $j \geq 0$; for instance, $unary(0)$, $unary(2)$, and $unary(999)$ are equal to ε, aa, and a^{999}, respectively. Under this representation, used in the sequel, we obviously automatically assume that $\Delta = \{a\}$ simply because a is the only input symbol needed. Next, we formalize the computation of integer functions by TMs based on *unary*.

Definition 10.3

I. Let g be a function over ${}_0\mathbb{N}$ and $M \in {}_{TM}\Psi$. *M computes g in unary* or, more briefly, *M computes g* iff $unary(g) = M\text{-}f$.
II. A function h over ${}_0\mathbb{N}$ is a *computable function* if there exists $M \in {}_{TM}\Psi$ such that M computes h; otherwise, h is an *uncomputable function*. ■

In greater detail, part I of Definition 10.3 says that M computes an integer function g over ${}_0\mathbb{N}$ if this equivalence holds:

$$g(x) = y \text{ iff } (unary(x), unary(y)) \in M\text{-}f, \text{ for all } x, y \in {}_0\mathbb{N}$$

Convention 10.4 Whenever $M \in {}_{TM}\Psi$ works on an integer $x \in {}_0\mathbb{N}$, x is expressed as *unary*(x). For brevity, whenever no confusion exists, instead of stating that *M works on x represented as unary*(x), we just state that *M works on x* in what follows. ■

Example 10.1 Let g be the *successor function* defined as $g(i) = i + 1$, for all $i \geq 0$. Construct a TM M that computes $\triangleright\blacktriangleright a^i \triangleleft \Rightarrow^* \triangleright\blacksquare a^{i+1}\triangleleft$ so it moves across a^i to the right bounder $\triangleleft$, replaces it with $a\triangleleft$, and returns to the left to finish its accepting computation in $\triangleright\blacksquare a^{i+1}\triangleleft$. As a result, M increases the number of as by one on the tape. Thus, by Definition 10.3, M computes g.

Example 10.2 Let g be the total function defined as $g(i) = j$, for all $i \geq 0$, where j is the smallest prime satisfying $i \leq j$. Construct a TM M that tests whether i, represented by a^i, is a prime in the way described in Example 9.2. If i is prime, M accepts in the configuration $\triangleright\blacksquare a^i \triangleleft$. If not, M continues its computation from $\triangleright\blacktriangleright a^{i+1}\triangleleft$ and tests whether $i + 1$ is prime; if it is, it accepts in $\triangleright\blacksquare a^{i+1}\triangleleft$. In this way, it continues increasing the number of as by one and testing whether the number is prime until it reaches a^j such that j is prime. As this prime j is obviously the smallest prime satisfying $i \leq j$, M accepts in $\triangleright\blacksquare a^j \triangleleft$. Thus, M computes g.

Both functions discussed in Examples 10.1 and 10.2 are total. However, there also exist partial integer functions, which may be undefined for some arguments. Suppose that g is a function over ${}_0\mathbb{N}$, which is undefined for some arguments. Let $M \in {}_{TM}\Psi$ compute g. According to Definition 10.3, for any $x \in {}_0\mathbb{N}$, $g(x)$ is undefined iff (*unary*(x), *unary*(y)) $\notin$ *M-f* for all $y \in {}_0\mathbb{N}$. Example 10.3 illustrates a partial integer function computed in this way.

Convention 10.5 As opposed to Examples 10.1 and 10.2, the next function as well as all other functions discussed throughout the rest of this section is defined over the set of positive integers, $\mathbb{N}$, which excludes 0. ■

Example 10.3 In this example, we consider a partial function g over $\mathbb{N}$ that is defined for 1, 2, 4, 8, 16, …, but it is undefined for the other positive integers. More precisely, $g(x) = 2x$ if $x = 2^n$, for some $n \in \mathbb{N}$; otherwise, $g(x)$ is undefined (see Figure 10.1).

We construct $M \in {}_{TM}\Psi$ that computes g as follows. Starting from $\triangleright\blacktriangleright a^i \triangleleft$, M computes $\triangleright\blacktriangleright a^i \triangleleft \Rightarrow^* \triangleright\blacktriangleright a^i A^j \triangleleft$ with j being the smallest natural number simultaneously satisfying $i \leq j$ and $j = 2^n$ with $n \in \mathbb{N}$. If $i = j$, then $i = 2^n$ and $g(i) = 2i = 2^{n+1}$, so M computes $\triangleright\blacktriangleright a^i A^j \triangleleft \Rightarrow^* \triangleright\blacksquare a^i a^j \triangleleft$ and, thereby, defines $g(i) = 2^{n+1}$. If $i < j$, then $2^{n-1} < i < 2^n$ and $g(i)$ is undefined, so M rejects a^i by $\triangleright\blacktriangleright a^i A^j \triangleleft \Rightarrow^* \triangleright\blacklozenge a^i \square^j \triangleleft$.

In somewhat greater detail, we describe M by the following Pascal-like algorithm that explains how M changes its configurations.

x	$g(x)$
1	2
2	4
3	undefined
4	8
5	undefined
6	undefined
7	undefined
8	16
⋮	⋮

Figure 10.1 Partial function *g* discussed in Example 10.3.

Let $\triangleright\blacktriangleright a^i\triangleleft$ be the input, for some $i \in \mathbb{N}$;
change $\triangleright\blacktriangleright a^i\triangleleft$ to $\triangleright\blacktriangleright a^iA\triangleleft$;
while the current configuration $\triangleright\blacktriangleright a^iA^j\triangleleft$ satisfies $j \leq i$ **do**
begin
  if $i = j$ **then**
    ACCEPT by computing $\triangleright\blacktriangleright a^iA^j\triangleleft \Rightarrow^* \triangleright\blacksquare a^ia^i\triangleleft$ {because $i = j = 2^m$ for some $m \in \mathbb{N}$}
  else
    compute $\triangleright\blacktriangleright a^iA^j\triangleleft \Rightarrow^* \triangleright\blacktriangleright a^iA^{2j}\triangleleft$ by changing each A to AA
end {of the **while** loop}
REJECT by computing $\triangleright\blacktriangleright a^iA^j\triangleleft \Rightarrow^* \triangleright\blacklozenge a^i\square^j\triangleleft$ {because $j > i$, so $i \neq 2^m$ for any $m \in \mathbb{N}$}.

As explained in the conclusion of Section 2.2.3, the set of all rewriting systems is countable because every definition of a rewriting system is finite, so this set can be put into a bijection with $\mathbb{N}$. For the same reason, the set of all TMs, which are defined as rewriting systems, is countable. However, the set of all functions is uncountable (see Example 1.3). From this observation, it straightforwardly follows the existence of uncomputable functions: there are just more functions than TMs. More surprisingly, however, even some simple total well-defined functions over $\mathbb{N}$ are uncomputable as Example 10.4 illustrates.

Example 10.4 For every $k \in \mathbb{N}$, set

$$ {}_kX = \{M \in {}_{TM}\Psi \mid card({}_MQ) = k + 1,\ {}_M\Delta = \{a\}\} $$

Informally, ${}_kX$ denotes the set of all TMs in ${}_{TM}\Psi$ with $k + 1$ states such that their languages are over $\{a\}$. Without any loss of generality, suppose that ${}_MQ = \{q_0, q_1, \ldots, q_k\}$ with $\blacktriangleright = q_0$ and $\blacksquare = q_k$. Let g be the total function over $\mathbb{N}$ defined for every $i \in \mathbb{N}$ so $g(i)$ equals the greatest integer $j \in \mathbb{N}$ satisfying $\triangleright q_0a\triangleleft \Rightarrow^* \triangleright q_ia^ju\triangleleft$ in M with $M \in {}_iX$, where $u \in \{\square\}^*$. In other words, $g(i) = j$ iff j is the greatest positive integer satisfying M-$f(a) = a^j$, where $M \in {}_iX$. Consequently, for every TM $K \in {}_iX$, either $|K\text{-}f(a)| \leq g(i)$ or K-$f(a)$ is undefined.

Observe that for every $i \in \mathbb{N}$, ${}_iX$ is finite. Furthermore, ${}_iX$ always contains $M \in {}_{TM}\Psi$ such that $\triangleright q_0a\triangleleft \Rightarrow^* \triangleright q_ia^ju\triangleleft$ in M with $j \in \mathbb{N}$, so g is total. Finally, $g(i)$ is defined quite rigorously because each TM in ${}_iX$ is deterministic (see Convention 9.8). At first glance, these favorable mathematical properties might suggest that g is computable, yet we next show that g is uncomputable by a proof based on diagonalization (see Example 1.3).

Gist. To show that g is uncomputable, we proceed, in essence, as follows. We assume that g is computable. Under this assumption, ${}_{TM}\Psi$ contains a TM M that computes g. We convert M to a TM N, which we subsequently transform to a TM O and show that O performs a computation that contradicts the definition of g, so our assumption that g is computable is incorrect. Thus, g is uncomputable.

In greater detail, let $M \in {}_{TM}\Psi$ be a TM that computes g. We can easily modify M to another TM $N \in {}_{TM}\Psi$ such that N computes $h(x) = g(2x) + 1$ for every $x \in \mathbb{N}$. Let $N \in {}_mX$, where $m \in \mathbb{N}$, so ${}_NQ = \{q_0, q_1, \ldots, q_m\}$ with $\blacktriangleright = q_0$ and $\blacksquare = q_m$. Modify N to the TM $O = ({}_O\Sigma, {}_OR)$, $O \in {}_{TM}\Psi$, in the following way. Define q_m as a nonfinal state. Set ${}_OQ = \{q_0, q_1, \ldots, q_m, q_{m+1}, \ldots, q_{2m}\}$ with $\blacktriangleright = q_0$ and $\blacksquare = q_{2m}$, so $O \in {}_{2m}X$. Initialize ${}_OR$ with the rules of ${}_NR$. Then, extend ${}_OR$ by the following new rules:

- $q_ma \rightarrow aq_m$ and $q_m\square \rightarrow aq_m$
- $q_h\triangleleft \rightarrow q_{h+1}\square\triangleleft$ and $q_{h+1}\square \rightarrow aq_{h+1}$, for all $m \leq h \leq 2m - 1$
- $aq_{2m} \rightarrow q_{2m}a$

Starting from $\triangleright q_0a\triangleleft$, O first computes $\triangleright q_0a\triangleleft \Rightarrow^* \triangleright q_ma^{h(m)}u\triangleleft$ with $u \in \{\square\}^*$ just like N does. Then, by the newly introduced rules, O computes $\triangleright q_ma^{h(m)}u\triangleleft \Rightarrow^* \triangleright q_{2m}a^{h(m)}a^{|u|}a^m\triangleleft$ with $q_{2m} = \blacksquare$. In brief, $\triangleright q_0a\triangleleft \Rightarrow^* \triangleright q_{2m}a^{h(m)}a^{|u|}a^m\triangleleft$ in O, which is impossible, however. Indeed, $|a^{h(m)}a^{|u|}a^m| = |a^{g(2m)+1}a^{|u|}a^m| > g(2m)$, so O-$f(1) > g(2m)$, which contradicts K-$f(1) \leq g(2m)$ for all $K \in {}_{2m}X$ because $O \in {}_{2m}X$. From this contradiction, we conclude that g is uncomputable.

In what follows, we often consider an enumeration of ${}_{TM}\Psi$. In essence, to enumerate ${}_{TM}\Psi$ means to list all TMs in ${}_{TM}\Psi$. We can easily obtain a list like this, for instance, by enumerating their codes according to length and alphabetic order (see Convention 10.6). If the code of $M \in {}_{TM}\Psi$ is the ith string in this lexicographic enumeration, we let M be the ith TM in the list.

Convention 10.6 In the sequel, ζ denotes some fixed enumeration of all possible TMs,

$$\zeta = {}_1M, {}_2M, \ldots$$

Regarding ζ, we just require the existence of two algorithms—(1) an algorithm that translates every $i \in \mathbb{N}$ to ${}_iM$ and (2) an algorithm that translates every $M \in {}_{TM}\Psi$ to i so $M = {}_iM$, where $i \in \mathbb{N}$. Let

$$\xi = {}_1M\text{-}f, {}_2M\text{-}f, \ldots$$

That is, ξ corresponds to ζ so ξ denotes the enumeration of the functions computed by the TMs listed in ζ. The positive integer i of ${}_iM\text{-}f$ is referred to as the *index of* ${}_iM\text{-}f$; in terms of ζ, i is referred to as the *index of* ${}_iM$. ■

Throughout the rest of this chapter, we frequently discuss TMs that construct other TMs, represented by their codes, and the TMs constructed in this way may subsequently create some other machines, and so on. Let us note that a construction like this commonly occurs in real-world computer science practice; for instance, a compiler produces a program that itself transforms the codes of some other programs, and so forth. Crucially, by means of universal TMs described in Section 9.3, we always know how to run any TM on any string, including a string that encodes another TM.

10.1.2 Recursion Theorem

Consider any total computable function γ over $\mathbb{N}$ and apply γ to the indices of TMs in ζ (see Convention 10.6). Theorem 10.7 says that there necessarily exists $n \in \mathbb{N}$, customarily referred to as a *fixed point* of γ, such that ${}_nM$ and ${}_{\gamma(n)}M$ compute the same function—that is, in terms of ξ, ${}_nM\text{-}f = {}_{\gamma(n)}M\text{-}f$. As a result, this important theorem rules out the existence of a total computable function that would map each index i to another index j so ${}_iM\text{-}f \neq {}_jM\text{-}f$.

Theorem 10.7 *Recursion Theorem*. For every total computable function γ over $\mathbb{N}$, there is $n \in \mathbb{N}$ such that ${}_nM\text{-}f = {}_{\gamma(n)}M\text{-}f$ in ξ.

Proof. Let γ be any total computable function over $\mathbb{N}$, and let $X \in {}_{TM}\Psi$ computes γ—that is, $X\text{-}f = \gamma$. First, for each $i \in \mathbb{N}$, introduce a TM $N_i \in {}_{TM}\Psi$ that works on every input $x \in \mathbb{N}$ as follows:

1. N_i saves x
2. N_i runs ${}_iM$ on i (according to Convention 10.6, ${}_iM$ denotes the TM of index i in ζ)
3. If ${}_iM\text{-}f(i)$ is defined and, therefore, ${}_iM$ actually computes ${}_iM\text{-}f(i)$, then N_i runs X on ${}_iM\text{-}f(i)$ to compute $X\text{-}f({}_iM\text{-}f(i))$
4. N_i runs ${}_{X\text{-}f({}_iM\text{-}f(i))}M$ on x to compute ${}_{X\text{-}f({}_iM\text{-}f(i))}M\text{-}f(x)$

Let O be a TM in ζ that computes the function $O\text{-}f$ over $\mathbb{N}$ such that for each $i \in \mathbb{N}$, $O\text{-}f(i)$ is equal to the index of N_i in ζ, constructed earlier. Note that although ${}_iM\text{-}f(i)$ may be undefined in (3), $O\text{-}f$ is total because N_i is defined for all $i \in \mathbb{N}$. Furthermore, ${}_{O\text{-}f(i)}M\text{-}f = {}_{X\text{-}f({}_iM\text{-}f(i))}M\text{-}f$ because

O-$f(i)$ is the index of N_i in ζ, and N_i computes ${}_{X\text{-}f({}_iM\text{-}f(i))}M\text{-}f$. As $X\text{-}f = \gamma$, we have ${}_{X\text{-}f({}_iM\text{-}f(i))}M\text{-}f = {}_{\gamma({}_iM\text{-}f(i))}M\text{-}f$. Let $O = {}_kM$ in ζ, where $k \in \mathbb{N}$; in other words, k is the index of O. Set $n = O\text{-}f(k)$ to obtain ${}_nM\text{-}f = {}_{O\text{-}f(k)}M\text{-}f = {}_{X\text{-}f({}_kM\text{-}f(k))}M\text{-}f = {}_{\gamma({}_kM\text{-}f(k))}M\text{-}f = {}_{\gamma(O\text{-}f(k))}M\text{-}f = {}_{\gamma(n)}M\text{-}f$. Thus, n is a fixed point of γ, and Theorem 10.7 holds true. ■

The recursion theorem is a powerful tool frequently applied in the theory of computation as illustrated next.

Example 10.5 Consider the enumeration $\zeta = {}_1M, {}_2M, \ldots$ (see Convention 10.6). Observe that Theorem 10.7 implies the existence of $n \in \mathbb{N}$ such that ${}_nM\text{-}f = {}_{n+1}M\text{-}f$, meaning that ${}_nM$ and ${}_{n+1}M$ compute the same function. Indeed, define the total computable function γ for each $i \in \mathbb{N}$ as $\gamma(i) = i + 1$. By Theorem 10.7, there is $n \in \mathbb{N}$ such that ${}_nM\text{-}f = {}_{\gamma(n)}M\text{-}f$ in ξ, and by the definition of γ, $\gamma(n) = n + 1$. Thus, ${}_nM\text{-}f = {}_{n+1}M\text{-}f$.

From a broader perspective, this result holds in terms of any enumeration of ${}_{TM}\Psi$, which may differ from ζ, provided that it satisfies the simple requirements stated in Convention 10.6. That is, the enumeration can be based on any representation whatsoever provided that there exists an algorithm that translates each representation to the corresponding machine in ${}_{TM}\Psi$ and vice versa. As an exercise, consider an alternative enumeration of this kind and prove that it necessarily contains two consecutive TMs that compute the same function. To rephrase this generalized result in terms of the Church–Turing thesis, any enumeration of procedures contains two consecutive procedures that compute the same function.

Before closing this section, we generalize functions so they map multiple arguments to a set, and we briefly discuss their computation by TMs. For k elements, $a_1, \ldots, a_k$, where $k \in \mathbb{N}$, $(a_1, \ldots, a_k)$ denotes the *ordered k-tuple* consisting of a_1 through a_k in this order. Let $A_1, \ldots, A_k$ be k sets. The *Cartesian product of* $A_1, \ldots, A_k$ is denoted by $A_1 \times \ldots \times A_k$ and defined as

$$A_1 \times \ldots \times A_k = \{(a_1, \ldots, a_k) | \; a_i \in A_i, \; 1 \leq i \leq k\}$$

Let $m \in \mathbb{N}$ and B be a set. Loosely speaking, an *m-argument function* from $A_1 \times \ldots \times A_m$ to B maps each $(a_1, \ldots, a_m) \in A_1 \times \ldots \times A_m$ to no more than one $b \in B$. To express that a function f represents an m-argument function, we write $f^{\underline{m}}$ (carefully distinguish $f^{\underline{m}}$ from f^m, which denotes the m-fold product of f, defined in the conclusion of Section 1.3). If $f^{\underline{m}}$ maps $(a_1, \ldots, a_m) \in A_1 \times \ldots \times A_m$ to $b \in B$, then $f^{\underline{m}}(a_1, \ldots, a_m)$ is defined as b, written as $f^{\underline{m}}(a_1, \ldots, a_m) = b$, where b is the *value* of $f^{\underline{m}}$ for arguments $a_1, \ldots, a_m$. If $f^{\underline{m}}$ maps $(a_1, \ldots, a_m)$ to no member of B, $f^{\underline{m}}(a_1, \ldots, a_m)$ is *undefined*. If $f^{\underline{m}}(a_1, \ldots, a_m)$ is defined for all $(a_1, \ldots, a_m) \in A_1 \times \ldots \times A_m$, $f^{\underline{m}}$ is *total*. If we want to emphasize that $f^{\underline{m}}$ may not be total, we say that $f^{\underline{m}}$ is *partial*.

Next, we generalize Definition 10.1 to the m-argument function $M\text{-}f^{\underline{m}}$ computed by $M \in {}_{TM}\Psi$. For the sake of this generalization, we assume that Δ contains #, used in the following definition to separate the m arguments of $M\text{-}f^{\underline{m}}$.

Definition 10.8 Let $M \in {}_{TM}\Psi$. The *m-argument function computed by M* is denoted by $M\text{-}f^{\underline{m}}$ and defined as

$$M\text{-}f^{\underline{m}} = \{(x, y) | \; x \in \Delta^*, \; occur(x, \#) = m - 1, \; y \in (\Delta - \{\#\})^*, \; \triangleright\blacktriangleright x \triangleleft \Rightarrow^* \triangleright\blacksquare yu \triangleleft \text{ in } M, \; u \in \{\square\}^*\}$$

■

That is, $f^{\underline{m}}(x_1, x_2, \ldots, x_m) = y$ iff $\triangleright\blacktriangleright x_1\#x_2\#\ldots\#x_m \triangleleft \Rightarrow^* \triangleright\blacksquare yu \triangleleft$ in M with $u \in \{\square\}^*$, and $f^{\underline{m}}(x_1, x_2, \ldots, x_m)$ is undefined iff M loops on $x_1\#x_2\#\ldots\#x_m$ or rejects $x_1\#x_2\#\ldots\#x_m$. Notice that $M\text{-}f^{\underline{1}}$ coincides with $M\text{-}f$ (see Definition 10.1).

According to Definition 10.8, for every $M \in {}_{TM}\Psi$ and every $m \in \mathbb{N}$, there exists $M\text{-}f^{m}$. At a glance, it is hardly credible that every $M \in {}_{TM}\Psi$ defines $M\text{-}f^{m}$ because ${}_{TM}\Psi$ obviously contains TMs that never perform a computation that define any member of $M\text{-}f^{m}$. However, if we realize that we might have $M\text{-}f^{m}$ completely undefined—that is, $M\text{-}f^{m} = \varnothing$, which is perfectly legal from a mathematical point of view, then the existence of $M\text{-}f^{m}$ corresponding to every $M \in {}_{TM}\Psi$ comes as no surprise.

Definition 10.9 Let $m \in \mathbb{N}$. A function f^{m} is a *computable function* if there exists $M \in {}_{TM}\Psi$ such that $f^{m} = M\text{-}f^{m}$; otherwise, f^{m} is *uncomputable*. ■

To use TMs as computers of m-argument integer functions, we automatically assume that TMs work with the *unary*-based representation of integers by analogy with one-argument integer functions computed by TMs (see Definition 10.3 and Convention 10.4).

Definition 10.10 Let $M \in {}_{TM}\Psi$, $m \in \mathbb{N}$, and f^{m} be an m-argument function from $A_1 \times \ldots \times A_m$ to $\mathbb{N}$, where $A_i = \mathbb{N}$, for all $1 \leq i \leq m$. *M computes f^{m}* iff this equivalence holds

$$f^{m}(x_1, \ldots, x_m) = y \text{ iff } (unary(x_1)\#\ldots\#unary(x_m), unary(y)) \in M\text{-}f^{m}$$ ■

10.1.3 Kleene's s-m-n Theorem

Theorem 10.12 says that for all $m, n \in \mathbb{N}$, there is a total computable function s of $m + 1$ arguments such that ${}_{i}M\text{-}f^{m+n}(x_1, \ldots, x_m, y_1, \ldots, y_n) = {}_{s(i, x_1, \ldots, x_m)}M\text{-}f^{n}(y_1, \ldots, y_n)$ for all $i, x_1, \ldots, x_m, y_1, \ldots, y_n$. In other words, considering the Church–Turing thesis, there is an algorithm such that from ${}_{i}M$ and $x_1, \ldots, x_m$, it determines another TM that computes ${}_{i}M\text{-}f^{m+n}(x_1, \ldots, x_m, y_1, \ldots, y_n)$ with only n arguments $y_1, \ldots, y_n$. In this way, the number of arguments is lowered, yet the same function is computed.

Convention 10.11 In this chapter, we often construct $M \in {}_{TM}\Psi$ from a finite sequence of strings, $z_1, \ldots, z_n$ (see, for instance, Theorem 10.12 and Example 10.6), and to express this clearly and explicitly, we denote M constructed in this way by $M_{[z_1, \ldots, z_n]}$. Specifically, in the proof of Theorem 10.12, $M_{[i, x_1, \ldots, x_m]}$ is constructed from $i, x_1, \ldots, x_m$, which are unary strings representing integers (see Convention 10.4). ■

Theorem 10.12 *Kleene's s-m-n Theorem*. For all $i, m, n \in \mathbb{N}$, there is a total computable $(m+1)$-argument function s^{m+1} such that ${}_{i}M\text{-}f^{m+n}(x_1, \ldots, x_m, y_1, \ldots, y_n) = {}_{s^{m+1}(i, x_1, \ldots, x_m)}M\text{-}f^{n}(y_1, \ldots, y_n)$.

Proof. We first construct a TM $S \in {}_{TM}\Psi$. Then, we show that $S\text{-}f^{m+1}$ satisfies the properties of s^{m+1} stated in Theorem 10.12, so we just take $s^{m+1} = S\text{-}f^{m+1}$ to complete the proof.

Construction of S. Let $m, n \in \mathbb{N}$. We construct a TM $S \in {}_{TM}\Psi$ so S itself constructs another machine in ${}_{TM}\Psi$ and produces its index in ζ as the resulting output value. More precisely, given input $i\#x_1\#\ldots\#x_m$, S constructs a TM, denoted by $M_{[i, x_1, \ldots, x_m]}$, for $i = 1, 2, \ldots$, and produces the index of $M_{[i, x_1, \ldots, x_m]}$—that is, j satisfying $M_{[i, x_1, \ldots, x_m]} = {}_{j}M$ in ζ—as the resulting output value. $M_{[i, x_1, \ldots, x_m]}$ constructed by S works as follows:

1. When given input $y_1\#\ldots\#y_n$, $M_{[i, x_1, \ldots, x_m]}$ shifts $y_1\#\ldots\#y_n$ to the right, writes $x_1\#\ldots\#x_m\#$ to its left, so it actually changes $y_1\#\ldots\#y_n$ to $x_1\#\ldots\#x_m\#y_1\#\ldots\#y_n$
2. $M_{[i, x_1, \ldots, x_m]}$ runs ${}_{i}M$ on $x_1\#\ldots\#x_m\#y_1\#\ldots\#y_n$

Properties of $S\text{-}f^{\underline{m+1}}$. Consider the $(m+1)$-argument function $S\text{-}f^{\underline{m+1}}$ computed by S constructed earlier. Recall that $S\text{-}f^{\underline{m+1}}$ maps $(i, x_1, \ldots, x_m)$ to the resulting output value equal to the index of $M_{[i, x_1, \ldots, x_m]}$ in ζ. More briefly, $S\text{-}f^{\underline{m+1}}(i, x_1, \ldots, x_m) = j$ with j satisfying $M_{[i, x_1, \ldots, x_m]} = {}_jM$ in ζ. Observe that $M_{[i, x_1, \ldots, x_m]}$ computes ${}_iM\text{-}f^{\underline{m+n}}(x_1, \ldots, x_m, y_1, \ldots, y_n)$ on every input $(y_1, \ldots, y_n)$, where ${}_iM\text{-}f^{\underline{m+n}}$ denotes the $(m+n)$-argument computable function. By these properties,

$${}_iM\text{-}f^{\underline{m+n}}(x_1, \ldots, x_m, y_1, \ldots, y_n) = {}_jM\text{-}f^{\underline{n}}(y_1, \ldots, y_n) = {}_{S\text{-}f^{\underline{m+1}}(i, x_1, \ldots, x_m)}M\text{-}f^{\underline{n}}(y_1, \ldots, y_n)$$

Therefore, to obtain the total computable $(m+1)$-argument function $s^{\underline{m+1}}$ satisfying Theorem 10.12, set $s^{\underline{m+1}} = S\text{-}f^{\underline{m+1}}$. ■

Theorem 10.12 represents a powerful tool for demonstrating closure properties concerning computable functions. To illustrate, by using this theorem, we prove that the set of computable one-argument functions is closed with respect to composition in Example 10.6.

Example 10.6 There is a total computable 2-argument function g^2 such that ${}_iM\text{-}f({}_jM\text{-}f(x)) = {}_{g^2(i, j)}M\text{-}f(x)$ for all $i, j, x \in \mathbb{N}$. We define the 3-argument function h^3 as $h^3(i, j, x) = {}_iM\text{-}f({}_jM\text{-}f(x))$ for all $i, j, x \in \mathbb{N}$. First, we show that h^3 is computable. Given $i, j, x \in \mathbb{N}$, we introduce a TM H that computes h^3 so it works on every input x as follows:

1. H runs ${}_jM$ on x
2. If ${}_jM\text{-}f(x)$ is defined and, therefore, produced by H in (1), H runs ${}_iM$ on ${}_jM\text{-}f(x)$
3. If ${}_iM\text{-}f({}_jM\text{-}f(x))$ is defined, H produces ${}_iM\text{-}f({}_jM\text{-}f(x))$, so H computes ${}_iM\text{-}f({}_jM\text{-}f(x))$

Thus, h^3 is computable. Let h^3 be computed by ${}_kM$ in ζ. That is, ${}_kM\text{-}f^2 = h^3$. By Theorem 10.12, there is a total computable function s such that ${}_{s^2(k, i, j)}M\text{-}f(x) = {}_kM\text{-}f^2(i, j, x)$ for all $i, j, x \in \mathbb{N}$. Set $g^2(i, j) = s^3(k, i, j)$ for all $i, j \in \mathbb{N}$. Thus, ${}_iM\text{-}f({}_jM\text{-}f(x)) = {}_{s^2(k, i, j)}M\text{-}f(x) = {}_{g^2(i, j)}M\text{-}f(x)$, for all i, j, $x \in \mathbb{N}$.

As already noted, from a broader perspective, we have actually proved that the composition of two computable functions is again computable, so the set of computable one-argument functions is closed with respect to composition. Establishing more closure properties concerning other common operations, such as addition and product, is left as an exercise.

Most topics concerning the computability of multi-argument functions are far beyond this introductory text. Therefore, we narrow our attention to one-argument functions. In fact, we just consider total functions from Δ^* to $\{\varepsilon\}$, on which we base Section 10.2, which discusses another crucially important topic of the computation theory—decidability.

10.2 Decidability

In this section, we formally explore the power of algorithms that decide problems. Because problem-deciding algorithms are used in most computer science areas, it is unfeasible to examine them all. Therefore, we consider only the algorithmic decidability concerning problems related to the language models, such as automata and grammars, discussed in Chapters 3, 6, and 9.

10.2.1 Turing Deciders

In essence, we express every problem by a language in this book. More specifically, a problem P is associated with the set of all its instances $\prod$ and with a property π that each instance either satisfies or, in contrast, does not satisfy. Given a particular instance $i \in \prod$, P asks whether i satisfies π. To decide

P by means of Turing deciders, which work on strings like any TMs, we represent P by an encoding language as

$$_{P}L = \{\langle i\rangle |\ i \in \prod, i \text{ satisfies } \pi\}$$

where $\langle i\rangle$ is a string representing instance i (see Convention 9.13). A Turing decider M, which halts on all inputs, decides P if (1) M rejects every input that represents no instance from $\prod$ and (2) for every $\langle i\rangle$ with $i \in \prod$, M accepts $\langle i\rangle$ iff i satisfies π, so M rejects $\langle i\rangle$ iff i does not satisfy π. More formally, $L(M) = {}_{P}L$, and $\Delta^* - L(M) = (\Delta^* - \{\langle i\rangle |\ i \in \prod\}) \cup \{\langle i\rangle |\ i \in \prod, i \text{ does not satisfy } \pi\}$.

In brief, we state P as

Problem 10.13 *P.*
Question: a formulation of *P*.
Language: ${}_{P}L$.

To illustrate our approach to decidability, we consider the problem referred to as *FA-Emptiness*. For any FA M, *FA-Emptiness* asks whether the language accepted by M is empty. ${}_{FA}\Psi$ is thus the set of its instances. The language encoding *FA-Emptiness* is defined as

$$_{\textit{FA-Emptiness}}L = \{\langle M\rangle |\ M \in {}_{FA}\Psi, L(M) = \varnothing\}$$

Formally, *FA-Emptiness* is specified as

Problem 10.14 *FA-Emptiness.*
Question: Let $M \in {}_{FA}\Psi$. Is $L(M)$ empty?
Language: ${}_{\textit{FA-Emptiness}}L = \{\langle M\rangle |\ M \in {}_{FA}\Psi, L(M) = \varnothing\}$.

We can construct a Turing decider for ${}_{\textit{FA-Emptiness}}L$ in a trivial way as demonstrated shortly (see Theorem 10.20).

In general, a problem that can be decided by a Turing decider is referred to as a *decidable problem* while an *undecidable problem* cannot be decided by any Turing decider. For instance, Problem 10.14 of emptiness reformulated in terms of TMs, symbolically referred to as *TM-Emptiness*, is undecidable as we demonstrate in Section 10.2.3.2 (see Problem 10.56 and Theorem 10.57). That is, we encode this important undecidable problem by its encoding language

$$_{\textit{TM-Emptiness}}L = \{\langle M\rangle |\ M \in {}_{TM}\Psi, L(M) = \varnothing\}$$

and prove that no Turing decider accepts ${}_{\textit{TM-Emptiness}}L$ (see Theorem 10.57).

Next, we define Turing deciders rigorously. As already pointed out, any Turing decider M halts on every input string. In addition, we require that M always halts with its tape completely blank. The following definition makes use of *M-f* (see Definition 10.1) and the domain of a function (see Section 1.3), so recall both notions before reading it.

Definition 10.15

I. Let $M \in {}_{TM}\Psi$. If M always halts and *M-f* is a function from Δ^* to $\{\varepsilon\}$, then M is a *Turing decider* (a *TD* for short).
II. Let L be a language and M be a TD. M is a *TD for L* if *domain*(*M-f*) = L.
III. A language is *decidable* if there is a TD for it; otherwise, the language is *undecidable*. ■

By part I in Definition 10.15, $M \in {}_{TM}\Psi$ is a TD if it never loops, and for every $x \in \Delta^*$, $\triangleright\blacktriangleright x\triangleleft \Rightarrow^* \triangleright iu\triangleleft$ in M with $i \in \{\blacksquare, \blacklozenge\}$ and $u \in \{\square\}^*$. By part II, a TD M for a language L satisfies $\triangleright\blacktriangleright x\triangleleft \Rightarrow^* \triangleright\blacksquare u\triangleleft$ in M for every $x \in L$ and $\triangleright\blacktriangleright y\triangleleft \Rightarrow^* \triangleright\blacklozenge v\triangleleft$ in M for every $y \in {}_M\Delta^* - L$, where $u, v \in \{\square\}^*$.

Convention 10.16 ${}_{TD}\Psi$ denotes the set of all TDs, and ${}_{TD}\Phi = \{L(M) | M \in {}_{TD}\Psi\}$; in other words, ${}_{TD}\Phi$ denotes the family of all decidable languages. ■

We close this section strictly in terms of formal languages—the principal subject of this book. First, we give a specific decidable formal language in Example 10.7. Then, we state a relation between the language families ${}_{FA}\Phi$, ${}_{CF}\Phi$, and ${}_{TD}\Phi$.

Example 10.7 Return to Example 9.1, in which we have designed a TM that accepts $L = \{x | x \in \{a, b, c\}^*, occur(x, a) = occur(x, b) = occur(x, c)\}$. As obvious, the TM does not satisfy Convention 9.8; in fact, it is not even deterministic. As a result, it is out of ${}_{TM}\Psi$, so it is definitely out of ${}_{TD}\Psi$ as well.

In this example, we design another TM D such that $D \in {}_{TD}\Psi$ and D accepts L. D repeatedly scans across the tape in a left-to-right way, erasing the leftmost occurrence of a, b, and c during every single scan. When it reaches $\triangleleft$ after erasing all these three occurrences, it moves left to $\triangleright$ and makes another scan like this. However, when D reaches $\triangleleft$ while some of the three symbols are missing on the tape, it can decide whether the input string is accepted. Indeed, if all three symbols are missing, D accepts; otherwise, it rejects. Therefore, D performs its final return to $\triangleright$ in either of the following two ways.

1. If the tape is completely blank and, therefore, all as, bs, and cs have been erased during the previous scans, D moves its head left to $\triangleright$ and accepts in a configuration of the form $\triangleright\blacksquare\square\ldots\square\triangleleft$.
2. If the tape is not blank and, therefore, contains some occurrences of symbols from X, where $\varnothing \subset X \subset \{a, b, c\}$, then during its return to $\triangleright$, D changes all these occurrences to $\square$ and rejects in a configuration of the form $\triangleright\blacklozenge\square\ldots\square\triangleleft$.

Omitting the state specification, Figure 10.2 schematically describes the acceptance of *babcca* by D.

Clearly, D is a TD for L, so L is a decidable language. Symbolically and briefly, $D \in {}_{TD}\Psi$ and $L \in {}_{TD}\Phi$.

We close this section by establishing the relations between ${}_{FA}\Phi$, ${}_{CF}\Phi$, and ${}_{TD}\Phi$, which are used in Section 11.3.

Theorem 10.17 ${}_{FA}\Phi \subset {}_{CF}\Phi \subset {}_{TD}\Phi$.

Proof. By Corollary 6.74, ${}_{FA}\Phi \subset {}_{CF}\Phi$. As an exercise, prove ${}_{CF}\Phi \subseteq {}_{TD}\Phi$. Consider L in Example 10.7. By the pumping lemma for ${}_{CF}\Phi$ (see Lemma 8.3), prove that $L \notin {}_{CF}\Phi$. Since $L \in {}_{TD}\Phi$ (see Example 10.7), ${}_{CF}\Phi \subset {}_{TD}\Phi$. Thus, Theorem 10.17 holds. ■

Scan	Tape
0	*babcca*
1	□□*b*□*ca*
2	□□□□□□
ACCEPT	

Figure 10.2 Acceptance of *babcca*.

10.2.2 Decidable Problems

In this section, we present several decidable problems for FAs and CFGs. However, we also point out that there exist problems that are decidable for FAs but undecidable for CFGs.

10.2.2.1 Decidable Problems for Finite Automata

Let M be any FA (see Definition 3.1). We give algorithms for deciding the following three problems.

- Is the language accepted by M empty?
- Is the language accepted by M finite?
- Is the language accepted by M infinite?

In addition, for any input string w, we decide the next problem.

- Is w a member of the language accepted by M?

Strictly speaking, we decide all these four problems for ${}_{cs}$DFAs (as stated in Section 3.2.2, a ${}_{cs}$DFA stands for a *c*ompletely *s*pecified *d*eterministic *f*inite *a*utomaton) because deciding them in this way is somewhat simpler than deciding these problems for the general versions of FAs, contained in ${}_{FA}\Psi$ (see Definitions 3.1 and 3.17, and Convention 3.2). However, because any FA can be algorithmically converted to an equivalent ${}_{cs}$DFA, all four problems are decidable for the general versions of FAs in ${}_{FA}\Psi$, too.

Convention 10.18 ${}_{csDFA}\Psi$ denotes the set of all ${}_{cs}$DFA. We suppose there exist a fixed encoding and decoding of automata in ${}_{csDFA}\Psi$ by analogy with the encoding and decoding of TMs (see Convention 9.13). That is, $\langle M \rangle$ represents the code of $M \in {}_{csDFA}\Psi$. Similarly, we suppose there exist an analogical encoding and decoding of the members of ${}_{csDFA}\Psi \times \Delta^*$ and ${}_{csDFA}\Psi \times {}_{csDFA}\Psi$. For brevity, we denote the codes of $(M, w) \in {}_{csDFA}\Psi \times \Delta^*$ and $(M, N) \in {}_{csDFA}\Psi \times {}_{csDFA}\Psi$ by $\langle M, w \rangle$ and $\langle M, N \rangle$, respectively. ■

As already stated in Section 10.2.1, the *FA-Emptiness* problem asks whether the language accepted by an FA is empty. Next, we prove that *FA-Emptiness* is decidable by demonstrating that its encoding language ${}_{FA\text{-}Emptiness}L$ belongs to ${}_{TD}\Phi$.

Problem 10.19 *FA-Emptiness.*
Question: Let $M \in {}_{csDFA}\Psi$. Is $L(M)$ empty?
Language: ${}_{FA\text{-}Emptiness}L = \{\langle M \rangle | \ M \in {}_{csDFA}\Psi, L(M) = \emptyset\}$.

Theorem 10.20 ${}_{FA\text{-}Emptiness}L \in {}_{TD}\Phi$.

Proof. As M is a ${}_{cs}$DFA, each of its states is reachable (see Definition 3.17). Thus, $L(M) = \emptyset$ iff ${}_{M}F = \emptyset$, which says that M has no final state. Design a TD D that works on every $\langle M \rangle$, where $M \in {}_{csDFA}\Psi$, so D accepts $\langle M \rangle$ iff ${}_{M}F = \emptyset$, and D rejects $\langle M \rangle$ iff ${}_{M}F \neq \emptyset$. ■

The *FA-Membership* problem asks whether a string $w \in {}_{M}\Delta^*$ is a member of the language accepted by $M \in {}_{csDFA}\Psi$. Like *FA-Emptiness*, *FA-Membership* is decidable.

Problem 10.21 *FA-Membership.*
Question: Let $M \in {}_{\alpha DFA}\Psi$ and $w \in {}_{M}\Delta^*$. Is w a member of $L(M)$?
Language: ${}_{FA\text{-}Membership}L = \{\langle M, w\rangle |\ M \in {}_{\alpha DFA}\Psi, w \in {}_{M}\Delta^*, w \in L(M)\}$.

Theorem 10.22 ${}_{FA\text{-}Membership}L \in {}_{TD}\Phi$.

Proof. Recall that any $M \in {}_{\alpha DFA}\Psi$ reads an input symbol during every move. Thus, after making precisely $|w|$ moves on $w \in \Delta^*$, M either accepts or rejects w. Therefore, construct a TD D that works on every $\langle M, w\rangle$ as follows:

1. D runs M on w until M accepts or rejects w (after $|w|$ moves)
2. D accepts $\langle M, w\rangle$ iff M accepts w, and D rejects $\langle M, w\rangle$ iff M rejects w ■

FA-Infiniteness is a problem that asks whether the language accepted by $M \in {}_{\alpha DFA}\Psi$ is infinite. To show that the decidability of the same problem can be often proved in several different ways, we next give two alternative proofs that *FA-Infiniteness* is decidable. We only sketch the first proof while describing the other in detail.

Problem 10.23 *FA-Infiniteness.*
Question: Let $M \in {}_{\alpha DFA}\Psi$. Is $L(M)$ infinite?
Language: ${}_{FA\text{-}Infiniteness}L = \{\langle M\rangle |\ M \in {}_{\alpha DFA}\Psi, L(M) \text{ is infinite}\}$.

Under our assumption that M is from ${}_{\alpha DFA}\Psi$, we obviously see that $L(M)$ is infinite iff its state diagram contains a cycle, so we can easily reformulate and decide this problem in terms of the graph theory in this way. Alternatively, we can prove this by using the pumping lemma for regular languages (see Lemma 5.1) in the following way. For every $M \in {}_{\alpha DFA}\Psi$, let ${}_{\infty ?}L(M)$ denote this finite language

$${}_{\infty ?}L(M) = \{x |\ x \in L(M), card({}_{M}Q) \leq |x| < 2card({}_{M}Q)\} \subseteq L(M)$$

Lemma 10.24 For every $M \in {}_{\alpha DFA}\Psi$, $L(M)$ is infinite iff ${}_{\infty ?}L(M) \neq \emptyset$.

Proof. To prove the *if* part of the equivalence, suppose that ${}_{\infty ?}L(M) \neq \emptyset$. Take any $z \in {}_{\infty ?}L(M)$. Recall that the pumping lemma constant k equals $card({}_{M}Q)$ in the proof of Lemma 5.1. As $card({}_{M}Q) \leq |z|$ by the definition of ${}_{\infty ?}L(M)$, Lemma 5.1 implies that $z = uvw$, where $0 < |v| \leq |uv| \leq card({}_{M}Q)$, and most importantly, $uv^m w \in L$, for all $m \geq 0$. Hence, $L(M)$ is infinite.

To prove the *only-if* part, assume that L is infinite. Let z be the shortest string such that $z \in L(M)$ and $|z| \geq 2card({}_{M}Q)$. As $|z| \geq 2card({}_{M}Q) \geq card({}_{M}Q)$, Lemma 5.1 implies that $z = uvw$, where $0 < |v| \leq |uv| \leq card({}_{M}Q)$, and $uv^m w \in L$, for all $m \geq 0$. Take $uv^0 w = uw \in L(M)$. Observe that $uw \in L(M)$ and $0 < |v|$ imply $2card({}_{M}Q) > |uw|$; indeed, if $2card({}_{M}Q) \leq |uw| < |z|$, then z would not be the shortest string satisfying $z \in L(M)$ and $|z| \geq 2card({}_{M}Q)$—a contradiction. As $0 < |v| \leq card({}_{M}Q)$, $card({}_{M}Q) \leq |uw| < 2card({}_{M}Q) \leq |z|$, so $uw \in {}_{\infty ?}L(M)$ and, therefore, ${}_{\infty ?}L(M) \neq \emptyset$. ■

Theorem 10.25 ${}_{FA\text{-}Infiniteness}L \in {}_{TD}\Phi$.

Proof. Construct a TD D that works on every $\langle M\rangle \in {}_{FA\text{-}Infiniteness}L$ so it first constructs ${}_{\infty ?}L(M)$. After the construction of this finite language, D accepts $\langle M\rangle$ iff ${}_{\infty ?}L(M) \neq \emptyset$, and D rejects $\langle M\rangle$ iff ${}_{\infty ?}L(M) = \emptyset$. ■

Consequently, Problem 10.26 is decidable as well.

Problem 10.26 *FA-Finiteness.*
Question: Let $M \in {}_{\sigma DFA}\mathbf{\Psi}$. Is $L(M)$ finite?
Language: ${}_{FA\text{-}Finiteness}L = \{\langle M\rangle | \ M \in {}_{\sigma DFA}\mathbf{\Psi}, L(M) \text{ is finite}\}$.

Corollary 10.27 ${}_{FA\text{-}Finiteness}L \in {}_{TD}\mathbf{\Phi}$. ■

The *FA-Equivalence* problem asks whether two ${}_{cs}$DFAs are equivalent; in other words, it asks whether both automata accept the same language. We decide this problem by using some elementary results of the set theory.

Problem 10.28 *FA-Equivalence.*
Question: Let $M, N \in {}_{\sigma DFA}\mathbf{\Psi}$. Are M and N equivalent?
Language: ${}_{FA\text{-}Equivalence}L = \{\langle M, N\rangle | \ M, N \in {}_{\sigma DFA}\mathbf{\Psi}, L(M) = L(N)\}$.

Theorem 10.29 ${}_{FA\text{-}Equivalence}L \in {}_{TD}\mathbf{\Phi}$.

Proof. Let M and N be in ${}_{\sigma DFA}\mathbf{\Psi}$. As an exercise, prove that $L(M) = L(N)$ iff $\varnothing = (L(M) \cap \sim L(N)) \cup (L(N) \cap \sim L(M))$. Construct a TD D that works on every $\langle M, N\rangle \in {}_{FA\text{-}Equivalence}L$ as follows:

1. From M and N, D constructs an FA O such that $L(O) = (L(M) \cap \sim L(N)) \cup (L(N) \cap \sim L(M))$
2. From O, D constructs an equivalent ${}_{cs}$DFA P
3. D decides whether $L(P) = \varnothing$ (see Theorem 10.20 and its proof)
4. If $L(P) = \varnothing$, $L(M) = L(N)$ and D accepts $\langle M, N\rangle$ while if $L(P) \neq \varnothing$, D rejects $\langle M, N\rangle$

Consider the constructions in (1) and (2); as an exercise, describe them in detail. ■

10.2.2.2 Decidable Problems for Context-Free Grammars

Let G be any CFG (see Definition 6.1). We give algorithms for deciding the following three problems.

- Is the language generated by G finite?
- Is the language accepted by G infinite?

In addition, for any input string w, we decide the next problem.

- Is w a member of the language generated by G?

Rather than discuss these problems in general terms of CFGs in ${}_{CFG}\mathbf{\Psi}$, we decide them for CFGs in the Chomsky normal form, in which every rule has either a single terminal or two nonterminals on its right-hand side (see Definition 6.34). Making use of this form, we find easier to decide two of them—*CFG-Membership* and *CFG-Infiniteness*. As any grammar in ${}_{CFG}\mathbf{\Psi}$ can be turned to an equivalent grammar in the Chomsky normal form by algorithms given in Section 6.2, deciding these problems for grammars satisfying the Chomsky normal form obviously implies their decidability for grammars in ${}_{CFG}\mathbf{\Psi}$ as well.

Convention 10.30 ${}_{CNF\text{-}CFG}\Psi$ denotes the set of all CFGs in Chomsky normal form. We suppose there exist a fixed encoding and decoding of the grammars in ${}_{CNF\text{-}CFG}\Psi$. Similarly to TMs and FAs (see Conventions 9.13 and 10.18), $\langle G \rangle$ represents the code of $G \in {}_{CNF\text{-}CFG}\Psi$. Similarly, we suppose there exist an analogical encoding and decoding of the members of ${}_{CNF\text{-}CFG}\Psi \times \Delta^*$ and ${}_{CNF\text{-}CFG}\Psi \times {}_{CNF\text{-}CFG}\Psi$. Again, for brevity, we denote the codes of $(G, w) \in {}_{CNF\text{-}CFG}\Psi \times \Delta^*$ and $(G, H) \in {}_{CNF\text{-}CFG}\Psi \times {}_{CNF\text{-}CFG}\Psi$ by $\langle G, w \rangle$ and $\langle G, H \rangle$, respectively. ■

Problem 10.31 *CFG-Emptiness.*
Question: Let $G \in {}_{CNF\text{-}CFG}\Psi$. Is $L(G)$ empty?
Language: ${}_{CFG\text{-}Emptiness}L = \{\langle G \rangle | \ G \in {}_{CNF\text{-}CFG}\Psi, L(G) = \emptyset\}$.

Theorem 10.32 ${}_{CFG\text{-}Emptiness}L \in {}_{TD}\Phi$.

Proof. Let $G \in {}_{CNF\text{-}CFG}\Psi$. Recall that a symbol in G is terminating if it derives a string of terminals (see Definition 6.15). As a result, $L(G)$ is nonempty iff ${}_GS$ is terminating, where ${}_GS$ denotes the start symbol of G (see Convention 6.2). Therefore, construct a TD D that works on $\langle G \rangle$ as follows:

1. D decides whether ${}_GS$ is terminating by Algorithm 6.16
2. D rejects $\langle G \rangle$ if ${}_GS$ is terminating; otherwise, D accepts $\langle G \rangle$ ■

Notice that the decision of *CFG-Emptiness* described in the proof of Theorem 10.32 is straightforwardly applicable to any CFG in ${}_{CFG}\Psi$ because this decision does not actually make use of the Chomsky normal form. During the decision of the next two problems, however, we make use of this form significantly.

Given a string w in Δ^* and a grammar G in ${}_{CNF\text{-}CFG}\Psi$, the *CFG-Membership* problem asks whether w is a member of $L(G)$. Of course, we can easily decide this problem by any of the general parsing algorithms discussed in Chapter 7 (see Algorithms 7.3 and 7.5). Next, we add yet another algorithm that decides this problem based on the Chomsky normal form.

Problem 10.33 *CFG-Membership*.
Question: Let $G \in {}_{CNF\text{-}CFG}\Psi$ and $w \in \Delta^*$. Is w a member of $L(G)$?
Language: ${}_{CFG\text{-}Membership}L = \{\langle G, w \rangle | \ G \in {}_{CNF\text{-}CFG}\Psi, w \in \Delta^*, w \in L(G)\}$.

The proof of Lemma 10.34 is simple and left as an exercise.

Lemma 10.34 Let $G \in {}_{CNF\text{-}CFG}\Psi$. Then, G generates every $w \in L(G)$ by making no more than $2|w| - 1$ derivation steps. ■

Theorem 10.35 ${}_{CFG\text{-}Membership}L \in {}_{TD}\Phi$.

Proof. As follows from the Chomsky normal form, ${}_{CNF\text{-}CFG}\Psi$ contains no grammar that generates ε. Therefore, we construct the following TD D that works on every $\langle G, w \rangle$ in either of the following two ways (A) and (B), depending on whether $w = \varepsilon$ or not.

A. Let $w = \varepsilon$. Clearly, $\varepsilon \in L(G)$ iff ${}_GS$ is an ε-nonterminal—that is, ${}_GS$ derives ε (see Definition 6.23). Thus, D decides whether ${}_GS$ is an ε-nonterminal by Algorithm 6.24, and if so, D accepts $\langle G, w \rangle$; otherwise, D rejects $\langle G, w \rangle$.

B. Let $w \neq \varepsilon$. Then, D works on $\langle G, w \rangle$ as follows:
   1. D constructs the set of all sentences that G generates by making no more than $2|w| - 1$ derivation steps
   2. If this set contains w, D accepts $\langle G, w \rangle$; otherwise, it rejects $\langle G, w \rangle$ ■

The *CFG-Infiniteness* problem asks whether the language generated by a CFG is infinite.

Problem 10.36 *CFG-Infiniteness.*
Question: Let $G \in {}_{CNF\text{-}CFG}\Psi$. Is $L(G)$ infinite?
Language: ${}_{CFG\text{-}Infiniteness}L = \{\langle G \rangle | \ G \in {}_{CNF\text{-}CFG}\Psi, L(G) \text{ is infinite}\}$.

As an exercise, prove Lemma 10.37.

Lemma 10.37 Let $G \in {}_{CNF\text{-}CFG}\Psi$. $L(G)$ is infinite iff $L(G)$ contains a sentence x such that $k \leq |x| < 2k$ with $k = 2^{card({}_{G}N)}$. ■

Theorem 10.38 ${}_{CFG\text{-}Infiniteness}L \in {}_{TD}\Phi$.

Proof. Construct a TD D that works on every $G \in {}_{CNF\text{-}CFG}\Psi$ as follows:

1. D constructs the set of all sentences x in $L(G)$ satisfying $k \leq |x| < 2k$ with $k = 2^{card({}_{G}N)}$
2. If this set is empty, D rejects $\langle G \rangle$; otherwise, D accepts $\langle G \rangle$ ■

Theorem 10.38 implies that we can also decide the following problem.

Problem 10.39 *CFG-Finiteness.*
Question: Let $G \in {}_{CNF\text{-}CFG}\Psi$. Is $L(G)$ finite?
Language: ${}_{CFG\text{-}Finiteness}L = \{\langle G \rangle | \ G \in {}_{CNF\text{-}CFG}\Psi, L(G) \text{ is finite}\}$.

Corollary 10.40 ${}_{CFG\text{-}Finiteness}L \in {}_{TD}\Phi$. ■

Recall that for FAs, we have formulated the problem *FA-Equivalence* and proved that it is decidable for them (see Theorem 10.29). However, we have not reformulated this problem for CFGs in this section. The reason is that this problem is undecidable for these grammars, which brings us to the topic of Section 10.2.3.

10.2.3 Undecidable Problems

As the central topic of this section, we consider several problems concerning TMs and show that they are undecidable. In addition, without any rigorous proofs, we briefly describe some undecidable problems not concerning TMs in the conclusion of this section.

Let P be a problem concerning TMs, and let P be encoded by a language ${}_{P}L$. Demonstrating that P is undecidable consists in proving that ${}_{P}L$ is an undecidable language. Like every rigorous proof in mathematics, a proof like this requires some ingenuity. Nevertheless, it is usually achieved by contradiction based on either of these two proof techniques—*diagonalization* and *reduction*.

10.2.3.1 Diagonalization

As a rule, a diagonalization-based proof is schematically performed in the following way.

1. Assume that ${}_{P}L$ is decidable, and consider a TD D such that $L(D) = {}_{P}L$.
2. From D, construct another TD O; then, by using the diagonalization technique (see Example 1.3), apply O on its own description $\langle O \rangle$ so this application results into a contradiction.
3. The contradiction obtained in (2) implies that the assumption in (1) is incorrect, so ${}_{P}L$ is undecidable.

Following this proof scheme almost literally, we next show the undecidability of the famous *halting problem* that asks whether $M \in {}_{TM}\Psi$ halts on input x. Observe that the following formulation of the *TM-Halting* problem makes use of the encoding language ${}_{TM\text{-}Halting}L$, introduced in the conclusion of Section 9.3.

Problem 10.41 *TM-Halting.*
Question: Let $M \in {}_{TM}\Psi$ and $w \in \Delta^*$. Does M halt on w?
Language: ${}_{TM\text{-}Halting}L = \{\langle M, w \rangle | \ M \in {}_{TM}\Psi, w \in \Delta^*, M \text{ halts on } w\}$.

Theorem 10.42 ${}_{TM\text{-}Halting}L \notin {}_{TD}\Phi$.

Proof. Assume that ${}_{TM\text{-}Halting}L$ is decidable. Then, there exists a TD D such that $L(D) = {}_{TM\text{-}Halting}L$. That is, for any $\langle M, w \rangle \in {}_{TM\text{-}Halting}L$, D accepts $\langle M, w \rangle$ iff M halts on w, and D rejects $\langle M, w \rangle$ iff M loops on w. From D, construct another TD O that works on every input w, where $w = \langle M \rangle$ with $M \in {}_{TM}\Psi$ (recall that according to Convention 9.13, every input string encodes a TM in ${}_{TM}\Psi$, so the case when w encodes no TM is ruled out), as follows:

1. O replaces w with $\langle M, M \rangle$, where $w = \langle M \rangle$
2. O runs D on $\langle M, M \rangle$
3. O accepts iff D rejects, and O rejects iff D accepts

That is, for every $w = \langle M \rangle$, O accepts $\langle M \rangle$ iff D rejects $\langle M, M \rangle$, and since $L(D) = {}_{TM\text{-}Halting}L$, D rejects $\langle M, M \rangle$ iff M loops on $\langle M \rangle$. Thus, O accepts $\langle M \rangle$ iff M loops on $\langle M \rangle$. Now, we apply the diagonalization technique in this proof: as O works on every input w, it also works on $w = \langle O \rangle$. Consider this case. Since O accepts $\langle M \rangle$ iff M loops on $\langle M \rangle$ for every $w = \langle M \rangle$, this equivalence holds for $w = \langle O \rangle$ as well, so O accepts $\langle O \rangle$ iff O loops on $\langle O \rangle$. Thus, $\langle O \rangle \in L(O)$ iff $\langle O \rangle \notin L(O)$—a contradiction. Therefore, ${}_{TM\text{-}Halting}L$ is undecidable; in symbols, ${}_{TM\text{-}Halting}L \notin {}_{TD}\Phi$. ■

Observe that Theorem 10.42 has its crucial consequences in practice as well as in theory. Indeed, considering the Church–Turing thesis, it rules out the existence of a universal algorithm that would decide, for any procedure, whether the procedure is an algorithm, which halts on all inputs (see Section 9.1). As a result, although we would obviously appreciate an algorithm like this in practice very much, we have to give up its existence once and for all. In terms of the formal language theory—the subject of this book—this theorem straightforwardly implies the following relation between the language families ${}_{TD}\Phi$ and ${}_{TM}\Phi$.

Theorem 10.43 ${}_{TD}\Phi \subset {}_{TM}\Phi$.

Proof. Clearly, ${}_{TD}\Phi \subseteq {}_{TM}\Phi$. By Theorems 9.16 and 10.42, ${}_{TM\text{-}Halting}L \in {}_{TM}\Phi - {}_{TD}\Phi$, so ${}_{TD}\Phi \subset {}_{TM}\Phi$. ■

As *TM-Halting* is undecidable, it comes as no surprise that the problem whether $M \in {}_{TM}\Psi$ loops on $w \in \Delta^*$ is not decidable either.

Problem 10.44 *TM-Looping.*
Question: Let $M \in {}_{TM}\Psi$ and $w \in \Delta^*$. Does M loop on x?
Language: ${}_{TM\text{-}Looping}L = \{\langle M, w\rangle | \ M \in {}_{TM}\Psi, x \in \Delta^*, M \text{ loops on } w\}$.

To prove the undecidability of *TM-Looping*, we establish Theorems 10.45 and 10.46. The first of them is obvious.

Theorem 10.45 ${}_{TM\text{-}Looping}L$ is the complement of ${}_{TM\text{-}Halting}L$. ■

Next, we prove that a language L is decidable iff ${}_{TM}\Phi$ contains both L and its complement $\sim L$.

Theorem 10.46 Let $L \subseteq \Delta^*$. $L \in {}_{TD}\Phi$ iff both L and $\sim L$ are in ${}_{TM}\Phi$.

Proof. To prove the *only-if* part of the equivalence, suppose that L is any decidable language, symbolically written $L \in {}_{TD}\Phi$. By Theorem 10.43, $L \in {}_{TM}\Phi$. By Definition 10.15, there is $M \in {}_{TD}\Psi$ such that $L = L(M)$. Change M to a TM $N \in {}_{TM}\Psi$ so that N enters a nonfinal state in which it keeps looping exactly when M enters the final state ■ (see Convention 9.8). As a result, $L(N) = \sim L(M) = \sim L$, so $\sim L \in {}_{TM}\Phi$. Thus, L and $\sim L$ are in ${}_{TM}\Phi$.

To prove the *if* part of the equivalence, suppose that $L \in {}_{TM}\Phi$ and $\sim L \in {}_{TM}\Phi$. That is, there exist $N \in {}_{TM}\Psi$ and $O \in {}_{TM}\Psi$ such that $L(N) = L$ and $L(O) = \sim L$. Observe that N and O cannot accept the same string because $L \cap \sim L = \varnothing$. On the other hand, every input w is accepted by either N or O because $L \cup \sim L = \Delta^*$. These properties underlie the next construction of a TD M for L from N and O. M works on every input w in the following way.

1. M simultaneously runs N and O on w so M executes by turns one move in N and O—that is, step by step, M computes the first move in N, the first move in O, the second move in N, the second move in O, and so forth.
2. M continues the simulation described in (1) until a move that would take N or O to an accepting configuration, and in this way, M finds out whether $w \in L(N)$ or $w \in L(O)$.
3. Instead of entering the accepting configuration in N or O, M halts and either accepts if $w \in L(N)$ or rejects if $w \in L(O)$—in greater detail, M changes the current configuration to a halting configuration of the form $\triangleright iu \triangleleft$, where $u \in \{\square\}^*$, $i \in \{\blacksquare, \blacklozenge\}$, $i = \blacksquare$ iff $w \in L(N)$, and $i = \blacklozenge$ iff $w \in L(O)$.

Observe that $L(M) = L$. Furthermore, M always halts, so $M \in {}_{TD}\Psi$ and $L \in {}_{TD}\Phi$. ■

Making use of Theorems 9.16 and 10.46, we easily show *TM-Looping* as an undecidable problem. In fact, we prove a much stronger result stating that ${}_{TM\text{-}Looping}L$ is not even in ${}_{TM}\Phi$.

Theorem 10.47 ${}_{TM\text{-}Looping}L \notin {}_{TM}\Phi$.

Proof. Assume ${}_{TM\text{-}Looping}L \in {}_{TM}\Phi$. Recall that ${}_{TM\text{-}Looping}L$ is the complement of ${}_{TM\text{-}Halting}L$ (see Theorem 10.45). Furthermore, ${}_{TM\text{-}Halting}L \in {}_{TM}\Phi$ (see Theorem 9.16). Thus, by Theorem 10.46, ${}_{TM\text{-}Halting}L$ would be decidable, which contradicts Theorem 10.42. Thus, ${}_{TM\text{-}Looping}L \notin {}_{TM}\Phi$. ■

Theorems 10.45 and 10.47 imply Corollary 10.48, which says that *TM-Looping* is undecidable.

Corollary 10.48 ${}_{TM\text{-}Looping}L \notin {}_{TD}\Phi$. ■

10.2.3.2 Reduction

Apart from diagonalization, we often establish the undecidability of a problem P so the decidability of P would imply the decidability of a well-known undecidable problem U, and from this contradiction, we conclude that P is undecidable. In other words, from the well-known undecidability of U, we actually derive the undecidability of P; hence, we usually say that we *reduce* U to P when demonstrating that P is undecidable in this way. In terms of the problem-encoding languages, to prove that a language ${}_{P}L$, encoding P, is undecidable, we usually follow the proof scheme given next.

1. Assume that ${}_{P}L$ is decidable, and consider a TD D such that $L(D) = {}_{P}L$.
2. Modify D to another TD that would decide a well-known undecidable language ${}_{U}L$—a contradiction.
3. The contradiction obtained in (2) implies that the assumption in (1) is incorrect, so ${}_{P}L$ is undecidable.

On the basis of this reduction-proof scheme, we next show the undecidability of the *TM-Membership* problem that asks whether input w is a member of $L(M)$, where $M \in {}_{TM}\Psi$ and $w \in \Delta^*$. It is worth noting that the following formulation of this problem makes use of the encoding language ${}_{TM\text{-}Membership}L$ that coincides with ${}_{TM\text{-}Acceptance}L$ defined in Section 9.3.

Problem 10.49 *TM-Membership.*
Question: Let $M \in {}_{TM}\Psi$ and $w \in \Delta^*$. Is w a member of $L(M)$?
Language: ${}_{TM\text{-}Membership}L = \{\langle M, w\rangle | \ M \in {}_{TM}\Psi, w \in \Delta^*, w \in L(M)\}$.

We prove the undecidability of this problem by reducing Problem 10.41 *TM-Halting* to it. That is, we show that if there were a way of deciding the *TM-Membership* problem, we could decide Problem 10.41 *TM-Halting*, which contradicts Theorem 10.42.

Theorem 10.50 ${}_{TM\text{-}Membership}L \notin {}_{TD}\Phi$.

Proof. Given $\langle M, x\rangle$, construct a TM N that coincides with M except that N accepts x iff M halts on x (recall that M halts on x iff M either accepts or rejects x according to Convention 9.3). In other words, $x \in L(N)$ iff M halts on x. If there were a TD D for ${}_{TM\text{-}Membership}L$, we could use D and this equivalence to decide ${}_{TM\text{-}Halting}L$. Indeed, we could decide ${}_{TM\text{-}Halting}L$ by transforming M to N as described earlier and asking whether $x \in L(N)$; from $x \in L(N)$, we would conclude that M halts on x while from $x \notin L(N)$, we would conclude that M loops on x. However, Problem 10.41 *TM-Halting* is undecidable (see Theorem 10.42), which rules out the existence of D. Thus, there is no TD for ${}_{TM\text{-}Membership}L$, so ${}_{TM\text{-}Membership}L \notin {}_{TD}\Phi$. ■

Next, we formulate the *Non-TM-Membership* problem, and based on Theorems 9.15 and 10.50, we prove that it is not decidable either.

Problem 10.51 *Non-TM-Membership.*
Question: Let $M \in {}_{TM}\Psi$ and $w \in \Delta^*$. Is w out of $L(M)$?
Language: ${}_{Non\text{-}TM\text{-}Membership}L = \{\langle M, w\rangle |\ M \in {}_{TM}\Psi, w \in \Delta^*, w \notin L(M)\}$.

By analogy with the proof of Theorem 10.50, we prove that ${}_{Non\text{-}TM\text{-}Membership}L$ is even out of ${}_{TM}\Phi$.

Theorem 10.52 ${}_{Non\text{-}TM\text{-}Membership}L \notin {}_{TM}\Phi$.

Proof. For the sake of obtaining a contradiction, suppose that ${}_{Non\text{-}TM\text{-}Membership}L \in {}_{TM}\Phi$. As already pointed out, ${}_{TM\text{-}Membership}L = {}_{TM\text{-}Acceptance}L$, so ${}_{TM\text{-}Membership}L \in {}_{TM}\Phi$ (see Theorem 9.15). As obvious, ${}_{Non\text{-}TM\text{-}Membership}L$ is the complement of ${}_{TM\text{-}Membership}L$. Thus, by Theorem 10.46, ${}_{TM\text{-}Membership}L$ would belong to ${}_{TD}\Phi$, which contradicts Theorem 10.50. Thus, ${}_{Non\text{-}TM\text{-}Membership}L \notin {}_{TM}\Phi$. ■

From Theorems 10.43 and 10.52, we obtain Corollary 10.53, saying that *Non-TM-Membership* is an undecidable problem.

Corollary 10.53 ${}_{Non\text{-}TM\text{-}Membership}L \notin {}_{TD}\Phi$. ■

Problem 10.54 asks whether $L(M)$ is regular, where $M \in {}_{TM}\Psi$. By reducing *TM-Halting* to it, we prove its undecidability.

Problem 10.54 *TM-Regularness.*
Question: Let $M \in {}_{TM}\Psi$. Is $L(M)$ regular?
Language: ${}_{TM\text{-}Regularness}L = \{\langle M\rangle |\ M \in {}_{TM}\Psi, L(M) \in {}_{reg}\Phi\}$.

Theorem 10.55 ${}_{TM\text{-}Regularness}L \notin {}_{TD}\Phi$.

Proof. Consider ${}_{TM\text{-}Halting}L = \{\langle M, w\rangle |\ M \in {}_{TM}\Psi, w \in \Delta^*, M \text{ halts on } w\}$. Recall that ${}_{TM\text{-}Halting}L \in {}_{TM}\Phi - {}_{TD}\Phi$ (see Theorems 10.42 and 10.43). Take any TM O such that $L(O) = {}_{TM\text{-}Halting}L$; for instance, in the proof of Theorem 9.16, ${}_{TM\text{-}Halting}U$ satisfies this requirement because $L({}_{TM\text{-}Halting}U) = {}_{TM\text{-}Halting}L$. Next, we construct a TM $W \in {}_{TM}\Psi$ so that W converts every input $\langle M, w\rangle$, where $M \in {}_{TM}\Psi$ and $w \in \Delta^*$, to a new TM, denoted by $N_{[M, w]}$. Next, we describe this conversion in a greater detail. Given $\langle M, w\rangle$, W constructs a TM $N_{[M, w]}$ that works on every input $y \in \Delta^*$ as follows:

1. $N_{[M, w]}$ places w somewhere behind y on its tape
2. $N_{[M, w]}$ runs M on w
3. If M halts on w, $N_{[M, w]}$ runs O on y and accepts if and when O accepts

If M loops on w, $N_{[M, w]}$ never gets behind (2), so the language accepted by $N_{[M, w]}$ equals $\varnothing$ in this case. If M halts on w, $N_{[M, w]}$ accepts y in (3) if and when O accepts y, so the language accepted by $N_{[M, w]}$ coincides with $L(O)$ in this case. Thus, the language accepted by $N_{[M, w]}$ equals $L(O)$ iff M halts on w, and the language accepted by $N_{[M, w]}$ equals $\varnothing$ iff M loops on w. By Theorems 10.17 and 10.43, $L(O)$ is not regular because $L(O) \in {}_{TM}\Phi - {}_{TD}\Phi$ and ${}_{reg}\Phi \subset {}_{TD}\Phi$. By Definition 3.23, $\varnothing$ is regular. Thus, the language accepted by $N_{[M, w]}$ is regular iff M loops on w, and the language accepted by $N_{[M, w]}$ is nonregular iff M halts on w. Hence, if ${}_{TM\text{-}Regularness}L$ were

decidable by a TD $V \in {}_{TD}\Psi$, W could make use of V and these equivalences to decide ${}_{TM\text{-}Halting}L$. Simply put, W would represent a TD for ${}_{TM\text{-}Halting}L$, which contradicts Theorem 10.42. Thus, ${}_{TM\text{-}Regularness}L$ is undecidable. ■

As an exercise, show the undecidability of the following three problems by analogy with the proof of Theorem 10.55.

Problem 10.56 *TM-Emptiness*.
Question: Let $M \in {}_{TM}\Psi$. Is $L(M)$ empty?
Language: ${}_{TM\text{-}Emptiness}L = \{\langle M\rangle |\ M \in {}_{TM}\Psi, L(M) = \varnothing\}$.

Theorem 10.57 ${}_{TM\text{-}Emptiness}L \notin {}_{TD}\Phi$. ■

Problem 10.58 *TM-Finiteness.*
Question: Let $M \in {}_{TM}\Psi$. Is $L(M)$ finite?
Language: ${}_{TM\text{-}Finiteness}L = \{\langle M\rangle |\ M \in {}_{TM}\Psi, L(M)$ is finite$\}$.

Theorem 10.59 ${}_{TM\text{-}Finiteness}L \notin {}_{TD}\Phi$. ■

Problem 10.60 *TM-Context-freeness*.
Question: Let $M \in {}_{TM}\Psi$. Is $L(M)$ context-free?
Language: ${}_{TM\text{-}Context\text{-}freeness}L = \{\langle M\rangle |\ M \in {}_{TM}\Psi, L(M) \in {}_{CF}\Phi\}$.

Theorem 10.61 ${}_{TM\text{-}Context\text{-}freeness}L \notin {}_{TD}\Phi$. ■

Consider Problem 10.62 that asks whether $L(M) = \Delta^*$, where $M \in {}_{TM}\Psi$. We again prove its undecidability by reducing Problem 10.41 *TM-Halting* to it.

Problem 10.62 *TM-Universality.*
Question: Let $M \in {}_{TM}\Psi$. Is $L(M)$ equal to Δ^*?
Language: ${}_{TM\text{-}Universality}L = \{\langle M\rangle |\ M \in {}_{TM}\Psi, L(M) = \Delta^*\}$.

Theorem 10.63 ${}_{TM\text{-}Universality}L \notin {}_{TD}\Phi$.

Proof. We reduce Problem 10.41 *TM-Halting* to Problem 10.62 *TM-Universality*. Once again, recall that ${}_{TM\text{-}Halting}L = \{\langle M, w\rangle |\ M \in {}_{TM}\Psi, w \in \Delta^*, M$ halts on $w\}$. We introduce a TM $W \in {}_{TM}\Psi$ so that W constructs the following TM $N_{[M, w]}$ from every input $\langle M, w\rangle$, where $M \in {}_{TM}\Psi$ and $w \in \Delta^*$. That is, given $\langle M, w\rangle$, W makes $N_{[M, w]}$ that works on every input $y \in \Delta^*$ as follows:

1. $N_{[M, w]}$ replaces y with w
2. $N_{[M, w]}$ runs M on w and halts if and when M halts

As $N_{[M, w]}$ works on every y in this way, its language equals Δ^* iff M halts on w while its language is empty iff M loops on w. Assume that ${}_{TM\text{-}Universality}L \in {}_{TD}\Phi$, so there is a TD V for ${}_{TM\text{-}Universality}L$. Thus, W could use V and these equivalences to decide ${}_{TM\text{-}Halting}L$, which contradicts Theorem 10.42. Hence, ${}_{TM\text{-}Universality}L \notin {}_{TD}\Phi$. ■

10.2.3.3 Undecidable Problems Not Concerning Turing Machines

We have concentrated our attention on the undecidability concerning TMs and their languages so far. However, undecidable problems arise in a large variety of areas in the formal language theory as well as out of this theory. Therefore, before concluding this section, we present some of them, but we completely omit proofs that rigorously show their undecidability.

Take CFGs, which are central to Section III of this book. For these grammars, the following problems are undecidable.

Problem 10.64 *CFG-Equivalence*.
Question: Let $G, H \in {}_{CFG}\Psi$. Are G and H equivalent?
Language: ${}_{CFG\text{-}Equivalence}L = \{\langle G, H\rangle |\ G, H \in {}_{CFG}\Psi, L(G) = L(H)\}$.

Problem 10.65 *CFG-Containment*.
Question: Let $G, H \in {}_{CFG}\Psi$. Does $L(G)$ contain $L(H)$?
Language: ${}_{CFG\text{-}Containment}L = \{\langle G, H\rangle |\ G, H \in {}_{CFG}\Psi, L(H) \subseteq L(G)\}$.

Problem 10.66 *CFG-Intersection*.
Question: Let $G, H \in {}_{CFG}\Psi$. Is the intersection of $L(G)$ and $L(H)$ empty?
Language: ${}_{CFG\text{-}Intersection}L = \{\langle G, H\rangle |\ G, H \in {}_{CFG}\Psi, L(H) \cap L(G) = \emptyset\}$.

Problem 10.67 *CFG-Universality*.
Question: Let $G \in {}_{CFG}\Psi$. Is $L(G)$ equal to ${}_{G}\Delta^{*}$?
Language: ${}_{CFG\text{-}Universality}L = \{\langle G\rangle |\ G \in {}_{CFG}\Psi, L(G) = {}_{G}\Delta^{*}\}$.

Problem 10.68 *CFG-Ambiguity*.
Question: Let $G \in {}_{CFG}\Psi$. Is G ambiguous?
Language: ${}_{CFG\text{-}Ambiguity}L = \{\langle G\rangle |\ G \in {}_{CFG}\Psi, G \text{ is ambiguous}\}$.

Within the formal language theory, however, there exist many undecidable problems concerning languages without involving their models, and some of them were introduced long time ago. To illustrate, in 1946, Post introduced a famous problem, which we here formulate in terms of ε-free homomorphisms, defined in Section 2.1. Let X, Y be two alphabets and g, h be two ε-free homomorphisms from X^{*} to Y^{*}; *Post's Correspondence Problem* is to determine whether there is $w \in X^{+}$ such that $g(w) = h(w)$. For example, consider $X = \{1, 2, 3\}$, $Y = \{a, b, c\}$, $g(1) = abbb$, $g(2) = a$, $g(3) = ba$, $h(1) = b$, $h(2) = aab$, and $h(3) = b$ and observe that $2231 \in X^{+}$ satisfies $g(2231) = h(2231)$. Consider a procedure that systematically produces all possible $w \in X^{+}$, makes $g(w)$ and $h(w)$, and tests whether $g(w) = h(w)$. If and when the procedure finds out that $g(w) = h(w)$, it halts and answers yes; otherwise, it continues to operate endlessly. Although there is a procedure like this, there is no algorithm, which halts on every input, to decide this problem. Simply put, Post's Correspondence Problem is undecidable.

Of course, out of the formal language theory, there exist many undecidable problems as well. To illustrate, mathematics will never have a general algorithm that decides whether statements in number theory with the plus and times are true or false. Although these results are obviously more than significant from a purely mathematical point of view, they are somewhat out of the scope of this book, which primarily concentrates its attention on formal languages and their models; therefore, we leave their discussion as an exercise.

10.2.4 General Approach to Undecidability

As showed in Section 10.2.3.2, many reduction-based proofs of undecidability are very similar. This similarity has inspired the theory of computation to undertake a more general approach to reduction, based on Definition 10.69, which makes use of the notion of a computable function (see Definition 10.2).

Definition 10.69 Let $K, L \subseteq \Delta^*$ be two languages. A total computable function f over Δ^* is a *reduction of K to L*, symbolically written as $K \,{}_f\!\angle\, L$, if for all $w \in \Delta^*$, $w \in K$ iff $f(w) \in L$. ■

Convention 10.70 Let $K, L \subseteq \Delta^*$. We write $K \angle L$ to express that there exists a reduction of K to L. Let us note that instead of $\angle$, $\leq$ is also used in the literature. ■

First, we establish a general theorem concerning $\angle$ in terms of ${}_{TM}\Phi$.

Theorem 10.71 Let $K, L \subseteq \Delta^*$. If $K \angle L$ and $L \in {}_{TM}\Phi$, then $K \in {}_{TM}\Phi$.

Proof. Let $K, L \subseteq \Delta^*$, $K \angle L$, and $L \in {}_{TM}\Phi$. Recall that $K \angle L$ means that there exists a reduction f of K to L, written as $K \,{}_f\!\angle\, L$ (see Definition 10.69 and Convention 10.70). As $L \in {}_{TM}\Phi$, there is a TM M satisfying $L = L(M)$. Construct a new TM N that works on every input $w \in \Delta^*$ as follows:

1. N computes $f(w)$ (according to Definition 10.2, f is computable)
2. N runs M on $f(w)$
3. If M accepts, then N accepts, and if M rejects, then N rejects

Notice that N accepts w iff M accepts $f(w)$. As $L = L(M)$, M accepts $f(w)$ iff $f(w) \in L$. As $K \angle L$ (see Definition 10.69), $w \in K$ iff $f(w) \in L$. Thus, $K = L(N)$, so $K \in {}_{TM}\Phi$. ■

Corollary 10.72 Let $K, L \subseteq \Delta^*$. If $K \angle L$ and $K \notin {}_{TM}\Phi$, then $L \notin {}_{TM}\Phi$. ■

By Theorem 10.71, we can easily prove that a language K belongs to ${}_{TM}\Phi$. Indeed, we take a language $L \in {}_{TM}\Phi$ and construct a TM M that computes a reduction of K to L, so $K \angle L$. Then, by Theorem 10.71, $K \in {}_{TM}\Phi$. For instance, from Theorem 9.15 (recall that ${}_{TM\text{-}Membership}L = {}_{TM\text{-}Acceptance}L$), it follows that ${}_{TM\text{-}Membership}L \in {}_{TM}\Phi$. Take this language. Demonstrate that ${}_{TM\text{-}Halting}L \angle {}_{TM\text{-}Membership}L$ to prove ${}_{TM\text{-}Halting}L \in {}_{TM}\Phi$. As a result, we have obtained an alternative proof that ${}_{TM\text{-}Halting}L \in {}_{TM}\Phi$, which also follows from Theorem 9.16.

Perhaps even more importantly, Corollary 10.72 saves us much work to prove that a language L is out of ${}_{TM}\Phi$. Typically, a proof like this is made in one of the following two ways.

I. Take a well-known language $K \notin {}_{TM}\Phi$ and construct a TM M that computes a reduction of K to L, $K \angle L$. As a result, Corollary 10.72 implies $L \notin {}_{TM}\Phi$.
II. By Definition 10.69, if f is a reduction of K to L, then f is a reduction of $\sim K$ to $\sim L$ as well. Therefore, to prove that $L \notin {}_{TM}\Phi$, take a language K with its complement $\sim K \notin {}_{TM}\Phi$ and construct a TM that computes a reduction of K to $\sim L$. As $K \angle \sim L$, we have $\sim K \angle \sim\sim L$. That is, $\sim K \angle L$, and by Corollary 10.72, $L \notin {}_{TM}\Phi$.

In fact, by a clever use of Corollary 10.72, we can sometimes show that both $L \notin {}_{TM}\Phi$ and $\sim L \notin {}_{TM}\Phi$, and both proofs, frequently resemble each other very much. To illustrate, in this way, we next prove that ${}_{TM\text{-}Equivalence}L \notin {}_{TM}\Phi$ and its complement ${}_{Non\text{-}TM\text{-}Equivalence}L \notin {}_{TM}\Phi$, where

$$_{TM\text{-}Equivalence}L = \{\langle M, N\rangle |\ M, N \in {}_{TM}\Psi, L(M) = L(N)\}, \text{ and}$$

$$_{Non\text{-}TM\text{-}Equivalence}L = \{\langle M, N\rangle |\ M, N \in {}_{TM}\Psi, L(M) \neq L(N)\}$$

Theorem 10.73 $_{TM\text{-}Equivalence}L \notin {}_{TM}\Phi$.

Proof. To show $_{TM\text{-}Equivalence}L \notin {}_{TM}\Phi$, we follow proof method II listed earlier. More specifically, we prove that $_{TM\text{-}Membership}L \angle {}_{Non\text{-}TM\text{-}Equivalence}L$ (see Problem 10.49 *TM-Membership* for $_{TM\text{-}Membership}L$); therefore, $_{TM\text{-}Equivalence}L \notin {}_{TM}\Phi$ because $_{Non\text{-}TM\text{-}Membership}L \notin {}_{TM}\Phi$ (see method II listed earlier and Theorem 10.52). To establish $_{TM\text{-}Membership}L \angle {}_{Non\text{-}TM\text{-}Equivalence}L$, we construct a TM X that computes a reduction of $_{TM\text{-}Membership}L$ to $_{Non\text{-}TM\text{-}Equivalence}L$. Specifically, X transforms every $\langle O, w\rangle$, where $O \in {}_{TM}\Psi$ and $w \in \Delta^*$, to the following two TMs, M and $N_{[O, w]}$, and produces $\langle M, N_{[O, w]}\rangle$ as output (we denote M without any information concerning $\langle O, w\rangle$ because its construction is completely independent of it—that is, X produces the same M for every $\langle O, w\rangle$). M and $N_{[O, w]}$ work as follows:

1. M rejects every input
2. On every input $x \in \Delta^*$, $N_{[O, w]}$ works so it runs O on w and accepts x if and when O accepts w

As obvious, $L(M) = \varnothing$. Because $N_{[O, w]}$ works on every input $x \in \Delta^*$ in the way described earlier, these two implications hold:

- If $w \in L(O)$, then $L(N_{[O, w]}) = \Delta^*$, which implies $L(M) \neq L(N_{[O, w]})$
- If $w \notin L(O)$, then $L(N_{[O, w]}) = \varnothing$, which means $L(M) = L(N_{[O, w]})$

Thus, X computes a reduction of $_{TM\text{-}Membership}L$ to $_{Non\text{-}TM\text{-}Equivalence}L$, so $_{TM\text{-}Equivalence}L \notin {}_{TM}\Phi$. ■

Observe that the proof of Theorem 10.74, which says that the complement of $_{TM\text{-}Equivalence}L$ is out of $_{TM}\Phi$ as well, parallels the proof of Theorem 10.73 significantly. As a matter of fact, while in the proof of Theorem 10.74, M always rejects, in the following proof, M always accepts; otherwise, both proofs coincide with each other.

Theorem 10.74 $_{Non\text{-}TM\text{-}Equivalence}L \notin {}_{TM}\Phi$.

Proof. To show that $_{Non\text{-}TM\text{-}Equivalence}L \notin {}_{TM}\Phi$, we prove that $_{TM\text{-}Membership}L \angle {}_{TM\text{-}Equivalence}L$. We define a reduction of $_{TM\text{-}Membership}L$ to $_{TM\text{-}Equivalence}L$ by a TM X that transforms every $\langle O, w\rangle$, where $O \in {}_{TM}\Psi$ and $w \in \Delta^*$, to the following two TMs M, $N_{[O, w]} \in {}_{TM}\Psi$ and produces $\langle M, N_{[O, w]}\rangle$ as output. M and $N_{[O, w]}$ are defined as follows:

1. M accepts every input string
2. On every input string x, $N_{[O, w]}$ runs O on w and accepts x if and when O accepts w

As obvious, $L(M) = \Delta^*$. If $w \in L(O)$, $L(N_{[O, w]}) = \Delta^*$ and $L(M) = L(N_{[O, w]})$; otherwise, $L(M) \neq L(N_{[O, w]})$. Hence, $_{TM\text{-}Membership}L \angle {}_{TM\text{-}Equivalence}L$. Therefore, by using proof method II, we obtain $_{Non\text{-}TM\text{-}Equivalence}L \notin {}_{TM}\Phi$. ■

Returning to the key topic of this section, we see that such results as Theorem 10.71 and Corollary 10.72 have often significant consequences in terms of undecidability. Indeed, if $L \notin {}_{TM}\Phi$, then a problem encoded by L is undecidable because ${}_{TD}\Phi \subset {}_{TM}\Phi$ (see Theorem 10.43).

Specifically, in this way, Theorems 10.73 and 10.74 imply the undecidability of the next two problems encoded by languages ${}_{TM\text{-}Equivalence}L$ and ${}_{Non\text{-}TM\text{-}Equivalence}L$, introduced earlier; for convenience, we repeat the definition of ${}_{TM\text{-}Equivalence}L$ and ${}_{Non\text{-}TM\text{-}Equivalence}L$ in the following problems again.

Problem 10.75 *TM-Equivalence*.
Question: Are M and N equivalent, where $M, N \in {}_{TM}\Psi$?
Language: ${}_{TM\text{-}Equivalence}L = \{\langle M, N\rangle |\ M, N \in {}_{TM}\Psi, L(M) = L(N)\}$.

Problem 10.76 *Non-TM-Equivalence*.
Question: Are M and N nonequivalent, where $M, N \in {}_{TM}\Psi$?
Language: ${}_{Non\text{-}TM\text{-}Equivalence}L = \{\langle M, N\rangle |\ M, N \in {}_{TM}\Psi, L(M) \neq L(N)\}$.

Corollary 10.77 ${}_{TM\text{-}Equivalence}L \notin {}_{TD}\Phi$ and ${}_{Non\text{-}TM\text{-}Equivalence}L \notin {}_{TD}\Phi$. ■

Next, we state results analogical to Theorem 10.71 and Corollary 10.72 in terms of ${}_{TD}\Phi$.

Theorem 10.78 Let $K, L \subseteq \Delta^*$. If $K \angle L$ and $L \in {}_{TD}\Phi$, then $K \in {}_{TD}\Phi$.

Proof. Let $K, L \subseteq \Delta^*$, $K \angle L$, and $L \in {}_{TD}\Phi$. Let f be a reduction of K to L. As already pointed out, by Definition 10.69, f is a reduction of $\sim K$ to $\sim L$, too. By Theorem 10.46, $L \in {}_{TD}\Phi$ iff $L \in {}_{TM}\Phi$ and $\sim L \in {}_{TM}\Phi$. By Theorem 10.71, $K \in {}_{TM}\Phi$ and $\sim K \in {}_{TM}\Phi$. Thus, $K \in {}_{TD}\Phi$ by Theorem 10.46. ■

Corollary 10.79 Let $K, L \subseteq \Delta^*$. If $K \angle L$ and $K \notin {}_{TD}\Phi$, then $L \notin {}_{TD}\Phi$. ■

Theorem 10.78 and Corollary 10.79 often save us much work when we show undecidability. In Examples 10.8 and 10.9, we revisit some of our earlier results concerning undecidability to see how they follow from Corollary 10.79.

Example 10.8 Reconsider Problem 10.56 *TM-Emptiness* and Theorem 10.57, stating that this problem is undecidable. In essence, this undecidability is established so that from any TM M and any string x, we algorithmically construct a TM N such that $L(N) = \emptyset$ iff M halts on x. To rephrase this in terms of languages, we define a reduction of ${}_{TM\text{-}Halting}L$ to ${}_{TM\text{-}Emptiness}L$, so ${}_{TM\text{-}Halting}L \angle {}_{TM\text{-}Emptiness}L$. As ${}_{TM\text{-}Halting}L \notin {}_{TD}\Phi$ (see Theorem 10.41), ${}_{TM\text{-}Emptiness}L \notin {}_{TD}\Phi$ by Corollary 10.79, so Problem 10.56 *TM-Emptiness* is undecidable.

Example 10.9 Earlier in this section, by using diagonalization, we proved that Problem 10.41 *TM-Halting* is undecidable, after which we showed that Problem 10.49 *TM-Membership* is undecidable so we reduced *TM-Halting* to *TM-Membership*. As this example shows, we could proceed the other way around. That is, first, by using diagonalization, we could prove that Problem 10.49 *TM-Membership* is undecidable; a proof like this is similar to the proof of Theorem 10.42 and, therefore, left as an exercise. Next, we show ${}_{TM\text{-}Membership}L \angle {}_{TM\text{-}Halting}L$, so the undecidability of *TM-Membership* implies the undecidability of *TM-Halting* by Corollary 10.79.

We construct a TM O that computes a total function f over Δ^* that maps every $\langle M, w\rangle$ to $\langle N, v\rangle$ so

$$\langle M, w\rangle \in {}_{TM\text{-}Membership}L \text{ iff } \langle N, v\rangle \in {}_{TM\text{-}Halting}L$$

Therefore, ${}_{TM\text{-}Membership}L \, {}_{f}\!\angle \, {}_{TM\text{-}Halting}L$. O is defined as follows:

1. On every input $\langle M, w\rangle$, O constructs a TM $W_{[M]}$ that works on every input in this way:
   a. $W_{[M]}$ runs M on the input
   b. $W_{[M]}$ accepts if M accepts, and $W_{[M]}$ loops if M rejects
2. Write $\langle W_{[M]}, w\rangle$ as output

Observe that $W_{[M]}$ loops on w iff M rejects w or loops on w; thus, in terms of the above equivalence, $W_{[M]}$ fulfills the role of N with v equal to w. Clearly, ${}_{TM\text{-}Membership}L \angle {}_{TM\text{-}Halting}L$. As ${}_{TM\text{-}Membership}L \notin {}_{TD}\Phi$, ${}_{TM\text{-}Halting}L \notin {}_{TD}\Phi$ by Corollary 10.79.

10.2.4.1 Rice's Theorem

Next, we discuss the undecidability concerning properties of Turing languages in ${}_{TM}\Phi$ rather than TMs in ${}_{TM}\Psi$. More specifically, we identify a property of Turing languages, π, with the subfamily of ${}_{TM}\Phi$ defined by this property—that is, this subfamily contains precisely the Turing languages that satisfy π. For instance, the property of being finite equals $\{L \in {}_{TM}\Phi | \ L \text{ is finite}\}$. In this way, we consider π as a decidable property if there exists a TD $D \in {}_{TD}\Psi$ such that $L(D)$ consists of all descriptions of TMs whose languages are in the subfamily defined by π.

Definition 10.80 Let $\pi \subseteq {}_{TM}\Phi$. Then, π is said to be a *property of Turing languages*.

I. A language $L \in {}_{TM}\Phi$ *satisfies* π if $L \in \pi$.
II. Set ${}_{\pi}L = \{\langle M\rangle | \ M \in {}_{TM}\Psi, L(M) \in \pi\}$. We say that π is *decidable* if ${}_{\pi}L \in {}_{TD}\Phi$; otherwise, π is *undecidable*.
III. We say that π is *trivial* if $\pi = {}_{TM}\Phi$ or $\pi = \varnothing$; otherwise, π is *nontrivial*. ■

For instance, the property of being finite is nontrivial because $\{L \in {}_{TM}\Phi | \ L \text{ is finite}\}$ is a nonempty proper subfamily of ${}_{TM}\Phi$. As a matter of fact, there are only two trivial properties—${}_{TM}\Phi$ and $\varnothing$—and both are trivially decidable because they are true either for all members of ${}_{TM}\Phi$ or for no member of ${}_{TM}\Phi$. As a result, we concentrate our attention on the nontrivial properties in what follows. Surprisingly, Rice's theorem, Theorem 10.81, states that all nontrivial properties are undecidable.

Theorem 10.81 *Rice's theorem*. Every nontrivial property is undecidable.

Proof. Let π be a nontrivial property. Without any loss of generality, suppose that $\varnothing \notin \pi$ (as an exercise, reformulate this proof in terms of $\sim\pi$ if $\varnothing \in \pi$). As π is nontrivial, π is nonempty, so there exists a Turing language $K \in \pi$. Let $N \in {}_{TM}\Psi$ be a TM such that $K = L(N)$.

For the sake of obtaining a contradiction, assume that π is decidable. In other words, there exists a TD $D \in {}_{TD}\Psi$ that decides ${}_{\pi}L$. Next, we demonstrate that under this assumption, ${}_{TM\text{-}Halting}L$ would belong to ${}_{TD}\Phi$, which contradicts Theorem 10.42. Indeed, we construct an algorithm that takes any $\langle M, x\rangle$, where $M \in {}_{TM}\Psi$ and $x \in \Delta^*$, and produces $\langle O\rangle$ as output, where $O \in {}_{TM}\Psi$, so $\langle M, x\rangle \in {}_{TM\text{-}Halting}L$ iff $\langle O\rangle \in {}_{\pi}L$, and by using this equivalence and D, we would decide ${}_{TM\text{-}Halting}L$. O is designed so that it works on every input string y as follows:

1. Saves y and runs M on x
2. If M halts on x, O runs N on y and accepts iff N accepts y

If M loops on x, so does O. As O works on every y in this way, $L(O) = \varnothing$ iff M loops on x. If M halts on x, O runs N on y, and O accepts y iff N accepts y, so $L(O) = L(N) = K$ in this case (recall that the case when $K = \varnothing$ is ruled out because $\varnothing \notin \pi$). Thus, $\langle M, x\rangle \in {}_{TM\text{-}Halting}L$ iff $\langle O\rangle \in {}_{\pi}L$. Apply D to decide whether $\langle O\rangle \in {}_{\pi}L$. If so, $\langle M, x\rangle \in {}_{TM\text{-}Halting}L$, and if not, $\langle M, x\rangle \notin {}_{TM\text{-}Halting}L$, so ${}_{TM\text{-}Halting}L$ would be in ${}_{TD}\Phi$, which contradicts ${}_{TM\text{-}Halting}L \notin {}_{TD}\Phi$ (see Theorem 10.42). Therefore, ${}_{\pi}L$ is undecidable. ■

Rice's theorem is a powerful result that has a great variety of consequences. For instance, consider the properties of being finite, regular, and context-free as properties of Turing languages. Rice's theorem straightforwardly implies that all these properties are undecidable.

10.2.5 Computational Complexity

This section takes a finer look at TDs by discussing their *computational complexity*. This complexity is measured according to their time and space computational requirements. The *time complexity* equals the number of moves they need to make a decision while the *space complexity* is defined as the number of visited tape symbols. Perhaps most importantly, this section points out that some problems are *tractable* for their reasonable computational requirements while others are *intractable* for their unmanageably high computational requirements to decide them. Simply put, there exist problems that are decidable in theory, but their decision is intractable in practice.

As most topics concerning complexity are too complicated to be discussed in this introductory text, this section differs from Sections 10.1.2 through 10.2.4, which have discussed their material in the form of mathematical formulas and proofs. Rather than give a fully rigorous presentation of computational complexity, this section explains only the basic ideas underlying it. Indeed, it restricts its attention to the very fundamental concepts and results, which are usually described informally. This section omits mathematically precise proofs. On the other hand, it points out some important open problems concerning the computational complexity.

We begin with the explanation of time complexity, after which we briefly conceptualize space complexity.

10.2.5.1 Time Complexity

Observe that the following definition that formalizes the time complexity of a TD considers the worst-case scenario concerning this complexity.

Definition 10.82 Let $M = ({}_{M}\Sigma, {}_{M}R)$ be a TD. The *time-complexity function of M*, denoted by ${}_{M}time$, is defined over ${}_{0}\mathbb{N}$ so for all $n \in {}_{0}\mathbb{N}$, ${}_{M}time(n)$ is the maximal number of moves M makes on an input string of length n before halting. ■

Example 10.10 Return to the TD D in Example 10.7 such that $L(D) = \{x \mid x \in \{a, b, c\}^{*}, occur(x, a) = occur(x, b) = occur(x, c)\}$. Recall that D scans across the tape in a left-to-right way while erasing the leftmost occurrence of a, b, and c. When it reaches $\triangleleft$ after erasing all three occurrences, it moves left to $\triangleright$ and makes another scan of this kind. However, when D reaches $\triangleleft$ while some of the three symbols are missing on the tape, D makes its final return to $\triangleright$ and halts by making one more move during which it accepts or rejects as described in Example 10.7. Let g be the integer function over ${}_{0}\mathbb{N}$ defined for all ${}_{0}\mathbb{N}$, so that if $n \in {}_{0}\mathbb{N}$ is divisible by 3, $g(n) = n(2(n/3)) + 1$, and if $n \in \mathbb{N}$ is indivisible by 3, $g(n) = g(m) + 2n$, where m is the smallest $m \in {}_{0}\mathbb{N}$ such that $m \leq n$ and m is divisible by 3. Observe that ${}_{D}time(n) = g(n)$. As an exercise, design another TD E such that $L(D) = L(E)$ and ${}_{E}time(n) < {}_{D}time(n)$, for all $n \in \mathbb{N}$.

As a general rule, for $M \in {}_{TD}\Phi$, ${}_{M}time$ is a complicated polynomial, whose determination represents a tedious and difficult task. Besides this difficulty, we are usually interested in the time complexity of M only when it is run on large inputs. As a result, rather than determine ${}_{M}time$ rigorously, we often consider the highest order term of ${}_{M}time$; on the other hand, we disregard the coefficient of this term as well as any lower terms. The elegant *big*-O *notation*, defined next, is customarily used for this purpose.

Definition 10.83

I. Let f and g be two functions over ${}_{0}\mathbb{N}$. If there exist $c, d \in \mathbb{N}$ such that for every $n \geq d$, $f(n)$ and $g(n)$ are defined and $f(n) \leq cg(n)$, then g is an *upper bound* for f, written as $f = O(g)$.
II. If $f = O(g)$ and g is of the form n^m, where $m \in \mathbb{N}$, then g is a *polynomial bound* for f.
III. Let $M \in {}_{TD}\Psi$. M is *polynomially bounded* if there is a polynomial bound for ${}_{M}time$. ■

Let f and g be two polynomials. In essence, according to points I and II of Definition 10.83, $f = O(g)$ says that f is less than or equal to g if we disregard differences regarding multiplicative constants and lower-order terms. Indeed, $f = O(g)$ implies $kf = O(g)$ for any $k \in \mathbb{N}$, so the multiplication constants are ignored. As $f(n) = cg(n)$ holds for all $n \geq d$, the values of any $n \leq d$ are also completely ignored as well. In practice, to obtain $g = n^m$ as described in point II, we simply take n^m as the highest-order term of f without its coefficient; for instance, if $f(n) = 918273645n^5 + 999n^4 + 1111n^3 + 71178n^2 + 98765431n + 1298726$, then $f = O(n^5)$. On the other hand, if $f \neq O(g)$, then there exist infinitely many values of n satisfying $f(n) > cg(n)$.

Based on point III of Definition 10.83, from a more practical point of view, we next distinguish the decidable problems that are possible to compute from those that are not.

Definition 10.84 Let P be a decidable problem. If P is decided by a polynomially bounded TD, P is *tractable*; otherwise, P is *intractable*. ■

Informally, this definition says that although intractable problems are decidable in principle, they can hardly be decided in reality as no decision maker can decide them in polynomial time. On the other hand, tractable problems can be decided in polynomial time, so they are central to practically oriented computer science. Besides their practical significance, however, tractable problems lead to some crucial topics of theoretical computer science as demonstrated next.

According to Convention 9.8, up until now, we have automatically assumed that the TMs work deterministically. We also know that deterministic TMs are as powerful as their nondeterministic versions (see Definition 9.4 and Theorem 9.5). In terms of their time complexity, however, their relationship remains open as pointed out shortly. Before this, we reformulate some of the previous notions in terms of nondeterministic TMs.

Definition 10.85

I. Let M be a TM according to Definition 9.1 (thus, M may not be deterministic). M is a *nondeterministic TD* if M always halts on every input string.
II. Let M be a nondeterministic TD. The *time complexity of M*, ${}_{M}time$, is defined by analogy with Definition 10.82—that is, for all $n \in {}_{0}\mathbb{N}$, ${}_{M}time(n)$ is the maximal number of moves M makes on an input string of length n before halting.
III. Like in Definition 10.83, a nondeterministic TD M is *polynomially bounded* if there is a polynomial bound for ${}_{M}time$. ■

Convention 10.86 ${}_{P}\Phi$ denotes the family of languages accepted by polynomially bounded (deterministic) TDs, and ${}_{NP}\Phi$ denotes the family of languages accepted by polynomially bounded nondeterministic TDs. ■

Notice that any TD represents a special case of a nondeterministic TD, so ${}_{P}\Phi \subseteq {}_{NP}\Phi$. However, it is a long-standing open problem whether ${}_{P}\Phi = {}_{NP}\Phi$, referred to as the *P = NP problem*. By using various methods, theoretical computer science has intensively attempted to decide this problem. One of the most important approaches to this problem is based on ordering the languages in ${}_{NP}\Phi$. The equivalence classes defined by this ordering consist of languages coding equally difficult problems. Considering the class corresponding to the most difficult problems, any problem coded by a language from this family is as difficult as any other problem coded by a language from ${}_{NP}\Phi$. Consequently, if we prove that this class contains a language that also belongs to ${}_{P}\Phi$, then ${}_{P}\Phi = {}_{NP}\Phi$; on the other hand, if we demonstrate that this class contains a language that does not belong to ${}_{P}\Phi$, then ${}_{P}\Phi \subset {}_{NP}\Phi$. Next, we describe this approach to the *P = NP* problem in somewhat greater detail.

Let $M \in {}_{TM}\Psi$. *M* is a *nonerasing TD* if it halts on every input string; consequently, as opposed to the basic definition of a TD (see Definition 10.15), *M* may halt with a non-blank tape. Definition 10.87, given next, makes use of the notion of a nonerasing TD.

Definition 10.87 Let Δ and ς be two alphabets, $J \subseteq \Delta^*$, and $K \subseteq \varsigma^*$. Then, *J* is *polynomially transformable* into *K*, symbolically written as $J \propto K$, if there is a polynomially bounded nonerasing TD *M* such that *M-f* (see Definition 10.1) is a total function from Δ^* to ς^* satisfying $x \in J$ iff $M\text{-}f(x) \in K$. ■

In other words, $J \propto K$ means that the difficulty of deciding *J* is no greater than the difficulty of deciding *K*, so the problem encoded by *J* is no more difficult than the problem encoded by *K*.

Definition 10.88 Let $L \in {}_{NP}\Phi$. If $J \propto L$ for every $J \in {}_{NP}\Phi$, then *L* is *NP-complete*. ■

A decision problem coded by an *NP*-complete language is an *NP-complete problem*. There exist a number of well-known *NP*-complete problems, such as Problem 10.89.

Problem 10.89 *Time-Bounded Acceptance.*
Question: Let *M* be a nondeterministic TM, $w \in {}_{M}\Delta^*$, and $i \in \mathbb{N}$. Does *M* accept *w* by computing no more than *i* moves?
Language: ${}_{TBA}L = \{\langle M, w, i\rangle |\ M$ is a nondeterministic TM, $w \in {}_{M}\Delta^*$, $i \in \mathbb{N}$, *M* accepts *w* by computing *i* or fewer moves$\}$.

Once again, by finding an *NP*-complete language *L* and proving either $L \in {}_{P}\Phi$ or $L \notin {}_{P}\Phi$, we would decide the *P = NP* problem. Indeed, if $L \in {}_{P}\Phi$, then ${}_{P}\Phi = {}_{NP}\Phi$, and if $L \notin {}_{P}\Phi$, then ${}_{P}\Phi \subset {}_{NP}\Phi$. So far, however, a proof like this has not been achieved yet, and the *P = NP* problem remains open.

10.2.5.2 Space Complexity

We close this section by a remark about the space complexity of TDs.

Definition 10.90 Let $M = ({}_{M}\Sigma, {}_{M}R)$ be a TD. A function over ${}_{0}\mathbb{N}$ represents the *space complexity of M*, denoted by ${}_{M}space$, if ${}_{M}space(i)$ equals the minimal number $j \in {}_{0}\mathbb{N}$ such that for all $x \in {}_{M}\Delta^i$, $y, v \in \Gamma^*$, $\triangleright {}_{M}sx \triangleleft \Rightarrow^* \triangleright yqv \triangleleft$ in *M* implies $|yv| \leq j$. ■

Thus, starting with an input string of length i, M always occurs in a configuration with no more than ${}_M space(i)$ symbols, including blanks, on the tape. As an exercise, define *polynomially space-bounded* (*deterministic*) *TDs* and *polynomially space-bounded nondeterministic TDs* by analogy with the corresponding deciders in terms of time complexity (see Definitions 10.83 and 10.85).

Convention 10.91 ${}_{PS}\Phi$ denotes the family of languages accepted by polynomially space-bounded (deterministic) TDs, and ${}_{NPS}\Phi$ denotes the family of languages accepted polynomially space-bounded nondeterministic TDs. ■

As opposed to the unknown relationship between ${}_P\Phi$ and ${}_{NP}\Phi$, we know more about ${}_{PS}\Phi$ and ${}_{NPS}\Phi$. Indeed, it holds that ${}_{PS}\Phi = {}_{NPS}\Phi \subset {}_{TD}\Phi$. It is also well-known that ${}_{NP}\Phi \subseteq {}_{PS}\Phi$, but it is not known whether this inclusion is proper—another important long-standing open problem in the theory of computation.

Exercises

1. This chapter contains results whose proofs are only sketched or even omitted. These results include Examples 10.5, 10.6, 10.9, and 10.10; Lemmas 10.34 and 10.37; and Theorems 10.17, 10.29, 10.57, 10.59, 10.61, and 10.81. Prove them rigorously.
2. Define the *constant function* f as $f(x) = 0$, for all $x \in {}_0\mathbb{N}$. Construct a TM that computes f (see Definition 10.3). Verify the construction by a rigorous proof.
3. Consider each of the following functions defined for all $x \in {}_0\mathbb{N}$. Construct a TM that computes it (see Definition 10.3). Verify the construction by a rigorous proof.
   i. $f(x) = 3x$
   ii. $f(x) = x + 3$
   iii. $f(x) = 3x + 3$
   iv. $f(x) = x^2$
   v. $f(x) = 2^x$

4 S. Consider each of the following total two-argument functions defined for all $x, y \in {}_0\mathbb{N}$. Construct a TM that computes it (Definition 10.10). Verify the construction by a rigorous proof.
   i. $f(x, y) = x + y$
   ii. $f(x, y) = xy$
   iii. $f(x, y) = x^y$
   iv. $f(x, y) = 1$ if $x = 0$; otherwise, $f(x, y) = x^y$
   v. $f(x, y) = 0$ if $x < y$; otherwise, $f(x, y) = x - y$
   vi. $f(x, y) = x^y$ if $x \geq y$; otherwise, $f(x, y) = y^x$

5. Let $\xi = {}_1M\text{-}f, {}_2M\text{-}f, \ldots$ have the same meaning as in Convention 10.6. Let $c \in \mathbb{N}$ be a constant. By analogy with Example 10.5, prove that there necessarily exists $i \in \mathbb{N}$ satisfying ${}_iM\text{-}f = {}_{i+c}M\text{-}f$.
6. For all $i, m \in \mathbb{N}$ satisfying $1 \leq i \leq m$, the *i-m-projection function*, ${}_{i\text{-}m}f$, is defined by

$$ {}_{i\text{-}m}f(n_1, \ldots, n_i, \ldots, n_m) = n_i $$

   for all $n_1, \ldots, n_i, \ldots, n_m \in {}_0\mathbb{N}$. Construct a TM that computes it (see Definition 10.10). Verify the construction by a rigorous proof.
7. Let g be a function of i variables, and let h_1 through h_i be i functions of j variables, where $i, j \in \mathbb{N}$. Then, the function f defined by $f = g(h_1(n_1, \ldots, n_j), \ldots, h_i(n_1, \ldots, n_j))$, where $n_1, \ldots, n_j \in {}_0\mathbb{N}$, represents the *composition of g and* h_1 *through* h_i. Construct a TM that computes it (see Definition 10.10). Verify the construction by a rigorous proof.

8. Let $m \in \mathbb{N}$. Let h be an m-argument function, g be an $(m+2)$-argument function of $m + 2$ variables, and f be the $(m+1)$-argument function defined by

$$f(n_1, \ldots, n_{m+1}) = h(n_1, \ldots, n_m) \text{ if } n_{m+1} = 0; \text{ otherwise,}$$

$$f(n_1, \ldots, n_{m+1}) = g(n_{m+1} - 1, f(n_1, \ldots, n_{m+1} - 1), n_1, \ldots, n_m) \text{ if } n_{m+1} \neq 0$$

for all $n_1, \ldots, n_{m+1} \in {}_0\mathbb{N}$. Then, f is the *recursion of h and g*. Design a TM that computes f. Verify the construction by a rigorous proof.

9. Let i be the constant function (see Exercise 2), a projection function (see Exercise 6), or the successor function (see Example 10.1); then, i is called an *initial function*.

   Any initial function is a *primitive recursive function*. Furthermore, the composition of primitive recursive functions is a primitive recursive function, and the recursion of primitive recursive functions is a primitive recursive function (see Exercises 7 and 8).

   Formalize the notion of a primitive recursive function strictly rigorously by a recursive definition. Prove that all primitive recursive functions are computable, but some computable functions are not primitive recursive functions.

10. Consider the function f over ${}_0\mathbb{N}$ defined for $n = 0$, $f(n) = 1$, and for $n > 0$, $f(n) = g(h(n - 1, f(n - 1)))$, where g is the constant function and h is a 1-2-projection function. Explain why f is a primitive recursive function. Simplify the definition of f. Design a TM that computes f. Verify the construction by a rigorous proof.

11. Let $j \in \mathbb{N}$. Let f be a total function of $j + 1$ variables and g be a partial function of j variables such that $g(n_1, \ldots, n_j)$ is the smallest k satisfying $f(n_1, \ldots, n_j, k) = 0$, and $g(n_1, \ldots, n_j)$ is undefined if for any k, $f(n_1, \ldots, n_j, k) \neq 0$, where $n_1, \ldots, n_j, k \in {}_0\mathbb{N}$. Then, g is the *minimization of f*.

    Design a TM that computes g. Verify the construction by a rigorous proof.

12. Any primitive recursive function is a *recursive function* (see Exercise 9). Furthermore, the composition of recursive functions is a recursive function, the recursion of recursive functions is a recursive function, and the minimization of a recursive function is a recursive function, too (see Exercises 7, 8, and 11).

    Formalize the notion of a recursive function strictly rigorously by a recursive definition. Prove statements (a) through (c).

    a. Every primitive recursive function is recursive.
    b. Some recursive functions are not primitive recursive functions.
    c. A function is recursive iff it is computable.

13. *Ackermann's function f* is the two-argument function over ${}_0\mathbb{N}$ defined by (i) through (iii) as follows:

    i. For $m \geq 0$, $f(0, m) = m + 1$
    ii. For $n \geq 1$, $f(n, 0) = f(n - 1, 1)$
    iii. For $n, m \geq 1$, $f(n, m) = f(n - 1, f(n, m - 1))$

    Demonstrate that f is a recursive function (see Exercise 12). Then, prove or disprove that f is a primitive recursive function (see Exercise 9).

14 S. Formalize the following problems by analogy with the formalization used throughout Section 10.2. Prove that all the following problems are decidable.

    i. Let $M \in {}_{FA}\mathbf{\Psi}$. Is $L(M) = {}_M\Delta^*$?
    ii. Let $M, N \in {}_{FA}\mathbf{\Psi}$. Is $L(M)$ contained in $L(N)$?
    iii. Let $M, N \in {}_{FA}\mathbf{\Psi}$. Is $L(M) \cap L(N)$ empty?

15. Consider each of problems (i) through (v). Formalize it by analogy with the formalization used throughout Section 10.2. Prove or disprove that it is decidable.

    i. Let $G \in {}_{LG}\mathbf{\Psi}$ (see Example 2.7). Is $L(G)$ regular?
    ii. Let $G \in {}_{LG}\mathbf{\Psi}$. Is $\sim L(G)$ linear?
    iii. Let $G \in {}_{LG}\mathbf{\Psi}$. Is $\sim L(G)$ context-free?
    iv. Let $G \in {}_{CFG}\mathbf{\Psi}$. Is $L(G)$ linear?
    v. Let $G \in {}_{CFG}\mathbf{\Psi}$. Is $\sim L(G)$ context-free?

16 S. Formalize the following problems by analogy with the formalization used throughout Section 10.2. Prove that they all are undecidable.
   i. Let $G, H \in {}_{CFG}\Psi$. Is $L(G)$ contained in $L(H)$?
   ii. Let $G, H \in {}_{CFG}\Psi$. Is $L(G) \cap L(H)$ empty?
   iii. Let $M \in {}_{TM}\Psi$. Does $L(M)$ contain ε?
   iv. Let $M, N \in {}_{TM}\Psi$. Is $L(M)$ contained in $L(N)$?
   v. Let $M, N \in {}_{TM}\Psi$. Is $L(M) \cap L(N)$ empty?

17. A *Post Tag System* G is a triple $G = (\Delta, R, S)$, where Σ is an alphabet, R is a finite *set of rules* of the form $x \rightarrow y$, where $x, y \in \Delta^*$, and $S \in \Delta^+$ is the start string. Define relation $\Rightarrow$ over Δ^* so $xz \Rightarrow zy$ for all $x \rightarrow y \in R$ and $z \in \Delta^*$. As usual, $\Rightarrow^*$ is the transitive and reflexive closure of $\Rightarrow$. The *language of* G is denoted by $L(G)$ and defined as $L(G) = \{w | \ w \in \Delta^*, S \Rightarrow^* w\}$.

    Consider the following problem. Formalize it by analogy with the formalization used throughout Section 10.2. Prove that it is undecidable.

    Let $G = (\Delta, R, S)$ be any Post Tag system and $w \in \Delta^*$. Is w a member of $L(G)$?

18. Reconsider Post's correspondence problem described in the conclusion of Section 10.2.3.3; pay a special attention to X therein. Assume that X contains a single symbol. Prove that under this assumption, this problem is decidable.
19. To demonstrate some very practical consequences implied by the results of undecidability covered in Section 10.2.3, consider an ordinary programming language, such as Java. Let L be the set of all well-written programs in this language. Demonstrate that the following problems are undecidable. In other words, computer science will never have software tools to answer these natural yes-no questions.
   i. Let $p \in L$. Can p enter an infinite loop?
   ii. Let $p \in L$. Can p produce any output?
   iii. Let $p, q \in L$. Do p and q produce the same output on all inputs?
20. Formalize the notion of a nondeterministic TM-based decision maker.
21. Let γ, η, θ be three functions over $\mathbb{N}$ such that $\gamma = O(\theta)$ and $\eta = O(\theta)$. Prove that $\gamma + \eta = O(\theta)$.
22. Let γ and θ be two functions such that θ is polynomial, and $\gamma = O(\theta)$. Prove that there exists a polynomial function η such that $\gamma(n) \leq \eta(n)$ for all $n \geq 1$.
23. Prove the following three results.
   a. The family of polynomial-space-bounded languages contains the family of nondeterministic polynomial-time-bounded languages.
   b. ${}_{TD}\Phi$ properly contains the family of nondeterministic polynomial-space-bounded languages.
   c. The family of polynomial-space-bounded languages coincides with the family of nondeterministic polynomial-space-bounded languages.
24. Prove that Problem 10.89 *Time-Bounded Acceptance* is NP-complete.

Solutions to Selected Exercises

4. Only (i) and (ii) are considered here.

   Consider (i). That is, $f(x, y) = x + y$ for all $x, y \in {}_0\mathbb{N}$. Construct a TM M so $(unary(x)\#unary(y), unary(x + y)) \in M\text{-}f^2$ (see Definition 10.10). Starting from $\triangleright\blacktriangleright a^x\#a^y\triangleleft$, M works as follows:
   a. Replaces # with a
   b. Changes the rightmost occurrence of a to □

   Thus, M computes $\triangleright\blacktriangleright a^x\#a^y\triangleleft \Rightarrow^* \triangleright\blacksquare a^{xy}\square\triangleleft$ and, therefore, $f(x, y) = x + y$. It is noteworthy that this construction takes care of the case when $x = 0$ or $y = 0$, too.

   Consider (ii). That is, $f(x, y) = xy$ for all $x, y \in {}_0\mathbb{N}$. Construct a TM M that computes this function as follows. Starting from $\triangleright\blacktriangleright a^x\#a^y\triangleleft$, M first finds out whether $x = 0$ or $y = 0$; if so, it computes $\triangleright\blacktriangleright a^x\#a^y\triangleleft \Rightarrow^* \triangleright\blacksquare\square^{x+y+1}\triangleleft$ and halts because $f(x, y) = 0$ in this case. Suppose that $x \geq 1$ and $y \geq 1$. M changes the current ordinary tape to a three-track tape, which has an analogical meaning to a two-track tape in Exercise 11 in Chapter 9. It places a^x and a^y on the

second and third tracks, respectively, while having the first track completely blank. Then, it performs this loop

i. Makes one new copy of a^x on the first track from the second track
ii. Changes one a to ▫ on the third track
iii. If an a occurs on the third track, continues from (1), and if no a occurs on the third track, M leaves this loop

When the third track contains no a, the first track contains xy occurrences of as. M changes the current three-track tape back to an ordinary one-track tape that contains a^{xy}. As a result, M computes $\triangleright\blacktriangleright a^x \# a^y \triangleleft \Rightarrow^* \triangleright\blacksquare a^{xy} \triangleleft$, so it computes $f(x, y) = xy$.

14. Consider (i). Formalize this problem in the following way.

Problem *FA-Universality*.
Question: Let $M \in {}_{\sigma}{}_{DFA}\Psi$. Is $L(M) = {}_M\Delta^*$?
Language: ${}_{FA\text{-}Universality}L = \{\langle M\rangle |\ M \in {}_{\sigma}{}_{DFA}\Psi, L(M) = {}_M\Delta^*\}$.

Theorem ${}_{FA\text{-}Universality}L \in {}_{TD}\Phi$.

Proof. From M, construct another $N \in {}_{\sigma}{}_{DFA}\Psi$ so N coincides with M except that ${}_NF = {}_MQ - {}_MF$. As M and N are completely specified, all their states are reachable (see Definition 3.13). Consequently, $L(M) = {}_M\Delta^*$ iff $L(N) = \emptyset$. Recall that ${}_{FA\text{-}Emptiness}L \in {}_{TD}\Phi$ (see Theorem 10.20); therefore, ${}_{FA\text{-}Universality}L \in {}_{TD}\Phi$. ■

This proof makes use of Theorem 10.20 to demonstrate ${}_{FA\text{-}Universality}L \in {}_{TD}\Phi$. However, in a simpler way, this result can be proved without any involvement of Theorem 10.20. An alternative proof like this is left as an exercise.

16. Consider (iii). Formalize the problem in the following way.

Problem ε-*TM-Membership*.
Question: Let $M \in {}_{TM}\Psi$. Does M accepts ε?
Language: ${}_{\varepsilon\text{-}TM\text{-}Membership}L = \{\langle M\rangle |\ M \in {}_{TM}\Psi, \varepsilon \in L(M)\}$.

To prove that this problem is undecidable, recall Problem 10.41 *TM-Halting* and Theorem 10.42.

Next, prove that the ε-*TM-Membership* problem is undecidable by contradiction. That is, assume that this problem is decidable. Under this assumption, show that Problem 10.41 *TM-Halting* would be decidable, which contradicts Theorem 10.42. Thus, the ε-*TM-Membership* problem is undecidable.

Theorem ${}_{\varepsilon\text{-}TM\text{-}Membership}L \notin {}_{TD}\Phi$.

Proof. Let M be any TM, and let $x \in \Delta^*$ be any input string. Without any loss of generality, suppose that either M accepts x or M loops on it (see Theorem 9.9). From M and x, construct a new TM $N_{[M, x]}$ that works on every input string $y \in \Delta^*$ as follows:

1. Replaces y with x on the tape
2. Runs M on x
3. Accepts if M accepts x

Observe that for all $y \in \Delta^*$, M accepts x iff $N_{[M, x]}$ accepts y, and M loops on x iff $N_{[M, x]}$ loops on y. Thus, $L(N_{[M, x]}) = \Delta^*$ iff $x \in L(M)$, and $L(N_{[M, x]}) = \emptyset$ iff $x \notin L(M)$.

Suppose that ${}_{\varepsilon\text{-}TM\text{-}Membership}L \in {}_{TD}\Phi$, so there exists a TD D such that $L(D) = {}_{\varepsilon\text{-}TM\text{-}Membership}L$. Of course, $\varepsilon \in \Delta^*$. Thus, $N_{[M, x]}$ accepts ε iff M halts on x while accepting x, and $N_{[M, x]}$ rejects ε iff M loops on $x \in L(M)$. Consequently, Problem 10.41 *TM-Halting* would be decidable, which contradicts Theorem 10.42. Thus, ${}_{\varepsilon\text{-}TM\text{-}Membership}L \notin {}_{TD}\Phi$, so the ε-*TM-Membership* problem is undecidable. ■