Chapter 9

Turing Machines and Their Variants

This three-section chapter introduces Turing machines (TMs) and their variants. Section 9.1 gives their basic definition. Then, Section 9.2 covers several special variants of these machines while Section 9.3 presents their universal versions, which fulfill an important role in Chapter 10.

9.1 Turing Machines and Their Languages

Return to the notion of a finite automaton (FA) (see Section 3.1). TM generalizes the FA in three essential ways. First, it can read and write on its tape. Second, its read-write head can move both to the right and to the left on the tape. Finally, the tape can be limitlessly extended to the right.

Definition 9.1 A *Turing machine* (TM) is a rewriting system $M = (\Sigma, R)$, where

- Σ contains subalphabets Q, F, Γ, Δ, $\{\triangleright, \triangleleft\}$ such that $\Sigma = Q \cup \Gamma \cup \{\triangleright, \triangleleft\}$, $F \subseteq Q$, $\Delta \subset \Gamma$, $\Gamma - \Delta$ always contains $\square$—the *blank* symbol (see Convention 4.6), and $\{\triangleright, \triangleleft\}$, Q, Γ are pairwise disjoint
- R is a finite *set of rules* of the form $x \to y$ satisfying
  i. $\{x, y\} \subseteq \{\triangleright\}Q$, or
  ii. $\{x, y\} \subseteq \Gamma Q \cup Q\Gamma$, or
  iii. $x \in Q\{\triangleleft\}$ and $y \in Q\{\square\triangleleft, \triangleleft\}$

Q, F, Γ, and Δ are referred to as the *set of states*, the *set of final states*, the *alphabet of tape symbols*, and the *alphabet of input symbols*, respectively. Q contains the *start state*, denoted by $\blacktriangleright$. Relations $\Rightarrow$ and $\Rightarrow^*$ are defined like in any rewriting system (see Definition 2.2 and Convention 2.3). *M accepts* $w \in \Delta^*$ if $\triangleright\blacktriangleright w\triangleleft \Rightarrow^* \triangleright ufv\triangleleft$ in M, where $u, v \in \Gamma^*$, $f \in F$. The *language*

accepted by M or, briefly, the *language of M* is denoted by $L(M)$ and defined as the set of all strings that M accepts; formally,

$$L(M) = \{w \mid w \in \Delta^*, \triangleright\blacktriangleright w \triangleleft \Rightarrow^* \triangleright ufv \triangleleft, u, v \in \Gamma^*, f \in F\} \quad \blacksquare$$

A *configuration* of M is a string of the form $\triangleright uqv\triangleleft$, $u, v \in \Gamma^*$, $q \in Q$, and let ${}_M\mathrm{X}$ denote the set of all configurations of M. We say that uv is on the *tape* of M, which is always delimited by $\triangleright$ and $\triangleleft$, referred to as the *left* and *right bounders*, respectively. In essence, $\triangleright uqv\triangleleft$ captures the current situation M occurs in. That is, in $\triangleright uqv\triangleleft$, q is the current state of M, whose read-write *head* occurs over the current *input symbol* defined as the leftmost symbol of $v\triangleleft$. If $\beta \Rightarrow \chi$ in M, where $\beta, \chi \in {}_M\mathrm{X}$, we say that M makes a *move* or a *computational step* from β to χ. A move made by an *extending rule* of the form $q\triangleleft \rightarrow p\square\triangleleft \in R$, where $q, p \in Q$, deserves our special attention because it actually extends the current tape by inserting a new occurrence of $\square$ in front of $\triangleleft$; more formally and briefly, $\triangleright uq\triangleleft \Rightarrow \triangleright up\square\triangleleft$, where $u \in \Gamma^*$. As follows from Definition 9.1, $\triangleright\blacktriangleright w\triangleleft \Rightarrow^* \chi$ implies $\chi \in {}_M\mathrm{X}$. We say that M makes a *sequence of moves* or a *computation* from β to χ if $\beta \Rightarrow^* \chi$ in M, where $\beta, \chi \in {}_M\mathrm{X}$.

Convention 9.2 For any TM M, we automatically assume that Q, F, Γ, Δ, and R have the same meaning as in Definition 9.1. If there exists any danger of confusion, we mark Q, F, Γ, Δ, and R with M as ${}_MQ$, ${}_MF$, ${}_M\Gamma$, ${}_M\Delta$, and ${}_MR$, respectively, to emphasize that they represent the components of M (in particular, we use these marks when several TMs are simultaneously discussed). ■

Example 9.1 Consider $L = \{x \mid x \in \{a, b, c\}^*, occur(x, a) = occur(x, b) = occur(x, c)\}$. Less formally, x is in L iff x has an equal number of as, bs, and cs; for instance, $babcca \in L$, but $babcc \notin L$. In this example, we construct a TM M such that $L(M) = L$.

Gist. M records symbols it has read by using its states from $power(\{a, b, c\})$ (see Section 1.2). That is, M moves on the tape in any direction. Whenever it reads an input symbol that is not already recorded in the current state, M can add this symbol into its current state while simultaneously changing it to $\square$ on the tape. M can anytime change the state that records all three symbols to the state that records no symbol at all. By using a special state, $\hookleftarrow$, M can scan the entire tape so it starts from $\triangleleft$ and moves left toward $\triangleright$ to find out whether the tape is completely blank, and if this is the case, M accepts.

Definition. Define $M = (\Sigma, R)$, where $\Sigma = Q \cup \Gamma \cup \{\triangleright, \triangleleft\}$, $\Gamma = \Delta \cup \{\square\}$, $\Delta = \{a, b, c\}$, $Q = \{\blacktriangleright, \hookleftarrow, \blacksquare\} \cup W$ with $W = \{\langle O\rangle \mid O \subseteq \{a, b, c\}\}$ and $F = \{\blacksquare\}$. Construct the rules of R by performing (1) through (5). (As stated in Section 1.2, {} denotes the empty set just like $\varnothing$ does. In this example, we use {} for this purpose.)

1. Add $\triangleright\blacktriangleright \rightarrow \triangleright\langle\{\}\rangle$ to R.
2. For every $\langle O\rangle \in W$ and every $d \in \Delta \cup \{\square\}$, add $\langle O\rangle d \rightarrow d\langle O\rangle$ and $d\langle O\rangle \rightarrow \langle O\rangle d$ to R.
3. For every $\langle O\rangle \in W$ such that $O \subset \{a, b, c\}$ and every $d \in \Delta - O$, add $\langle O\rangle d \rightarrow \langle O \cup \{d\}\rangle\square$ to R.
4. Add $\langle\{a, b, c\}\rangle d \rightarrow \langle\{\}\rangle d$ to R, where $d \in \Delta \cup \{\square, \triangleleft\}$.
5. Add $\langle\{\}\rangle\triangleleft \rightarrow \hookleftarrow\triangleleft$, $\square\hookleftarrow \rightarrow \hookleftarrow\square$, and $\triangleright\hookleftarrow \rightarrow \triangleright\blacksquare$ to R.

Computation. Consider both the informal and formal description of M. Observe that by (1), M starts every computation. By (2), M moves on its tape. By (3), M adds the input symbol into its current state from $power(\Delta)$ and, simultaneously, changes the input symbol to $\square$ on the tape. By (4), M empties $\{a, b, c\}$ so it changes this state to the state equal to the empty set. By (5), M makes a final

scan of the tape, starting from ◁ and moving left toward ▷, to make sure that the tape is completely blank, and if it is, M accepts.

For instance, in this way, M accepts *babcca* as follows:

$$
\begin{aligned}
▷▶babcca◁ &\Rightarrow ▷⟨\{\}⟩babcca◁\\
&\Rightarrow^{*} ▷babc⟨\{\}⟩ca◁\\
&\Rightarrow ▷babc⟨\{c\}⟩□a◁\\
&\Rightarrow^{*} ▷ba⟨\{c\}⟩bc□a◁\\
&\Rightarrow ▷ba⟨\{b, c\}⟩□c□a◁\\
&\Rightarrow^{*} ▷ba□c□⟨\{b, c\}⟩a◁\\
&\Rightarrow ▷ba□c□⟨\{a, b, c\}⟩□◁\\
&\Rightarrow^{*} ▷b⟨\{a, b, c\}⟩a□c□□◁\\
&\Rightarrow ▷b⟨\{\}⟩a□c□□◁\\
&\Rightarrow^{*} ▷□□□□□□⟨\{\}⟩◁\\
&\Rightarrow ▷□□□□□□↵⌟◁\\
&\Rightarrow ▷□□□□□↵⌟□◁\\
&\Rightarrow^{*} ▷↵⌟□□□□□□◁\\
&\Rightarrow ▷■□□□□□□◁
\end{aligned}
$$

Notice, however, M accepts the same string in many other ways, including

$$
\begin{aligned}
▷▶babcca◁ &\Rightarrow ▷⟨\{\}⟩babcca◁\\
&\Rightarrow ▷⟨\{b\}⟩□abcca◁\\
&\Rightarrow ▷□⟨\{b\}⟩abcca◁\\
&\Rightarrow ▷□⟨\{a, b\}⟩□bcca◁\\
&\Rightarrow^{*} ▷□□b⟨\{a, b\}⟩cca◁\\
&\Rightarrow ▷□□b⟨\{a, b, c\}⟩□ca◁\\
&\Rightarrow ▷□□b⟨\{\}⟩□ca◁\\
&\Rightarrow ▷□□⟨\{\}⟩b□ca◁\\
&\Rightarrow ▷□□⟨\{b\}⟩□□ca◁\\
&\Rightarrow^{*} ▷□□□□□□↵⌟◁\\
&\Rightarrow^{*} ▷↵⌟□□□□□□◁\\
&\Rightarrow ▷■□□□□□□◁
\end{aligned}
$$

Working on the same string in several different ways, M represents a nondeterministic rewriting system (see Section 2.2.3). In Chapter 10, we construct another TM that accepts L in a deterministic way (see Example 9.2). From a more general viewpoint, we explain how to turn any TM to an equivalent TM that works deterministically later in this chapter (see Theorem 9.5).

As illustrated in Example 9.1, the strictly *formal description* of a TM spells out the states, symbols, and rules of the TM under discussion. It is the most detailed and, thereby, rigorous description. At the same time, this level of description tends to be tremendously lengthy and tedious. Thus, paradoxically, this fully detailed description frequently obscures what the TM actually is designed for. For instance, without any intuitive comments included in Example 9.1, we would found somewhat difficult to figure out the way the TM accepts its language. Therefore, in the sequel, we prefer an *informal description* of TMs. That is, we describe them as procedures, omitting various details concerning their components. Crucially, the Church–Turing thesis makes both ways of description perfectly legitimate because it assures us that every procedure is identifiable with a TM defined in a rigorously mathematical way. As a matter of fact, whenever describing TMs in an informal way, we always make sure that the translation from this informal description to the corresponding formal description represents a straightforward task; unfortunately, this task

is usually unbearably time-consuming, too. To illustrate an informal description of TMs, we give the following example that informally describes a TM as a Pascal-like procedure, which explains the changes of the tape but omits the specification of states or rules.

Convention 9.3 By analogy with Convention 4.1, when informally describing a TM as a Pascal-like procedure, we express that the machine accepts or rejects its input by **ACCEPT** or **REJECT**, respectively. ■

Example 9.2 Consider $L = \{a^i \mid i \text{ is a prime number}\}$. This example constructs a TM M satisfying $L(M) = L$. Therefore, from a more general viewpoint, TMs are able to recognize the primes as opposed to pushdown automata (PDAs) (see Theorem 6.65 and Exercise 2 in Chapter 8). M is defined as a Pascal-like procedure in the following way:

Let a^i be the input string on the tape, where $i \in \mathbb{N}$;

if $i \leq 1$ **then**
  REJECT

change a^i to AAa^{i-2} on the tape
while $A^k a^h$ occurs on the tape with $k \leq h$ and $i = k + h$ **do**
begin
  on the tape, change $A^k a^h$ to the unique string y satisfying $i = |y|$ and $y \in A^k\{a^k A^k\}^* z$ with $z \in prefix(a^k A^{k-1})$;
  if $|z| = 0$ **or** $|z| = k$ **then**
    REJECT
  else
    change y to $A^{k+1}a^{h-1}$ on the tape
end {of the **while** loop}
ACCEPT.

Observe that i is no prime iff an iteration of the **while** loop obtains $y = A^k a^k A^k \ldots a^k A^k$. Indeed, at this point, i is divisible by k, so M rejects a^i. On the other hand, if during every iteration, $y = A^k a^k A^k \ldots a^k A^k z$ such that $z \in prefix(a^k A^{k-1}) - \{\varepsilon, a^k\}$, then after exiting from this loop, M accepts the input string because i is a prime.

In the **while** loop, consider the entrance test whether $A^k a^h$ occurs on the tape with $k \leq h$ and $i = k + h$. By using several states, tape symbols, and rules, we can easily reformulate this test to its strictly formal description in a straightforward way. However, a warning is in order: this reformulation also represents a painfully tedious task. As obvious, a strictly mathematical definition of the other parts of M is lengthy as well.

Even more frequently and informally, we just use English prose to describe procedures representing TMs under consideration. As a matter of fact, this *highly informal description* of TMs is used in most proofs of theorems given in the sequel.

9.2 Restricted Turing Machines

In this section, we restrict TMs so that compared to their general versions (see Definition 9.1), the resulting restricted TMs are easier to deal with, and yet, they are equally powerful. In essence, we classify all the restrictions into (i) restrictions placed on the way TMs perform their computation and (ii) restrictions placed on the size of TMs.

9.2.1 Computational Restrictions

Perhaps most importantly, we want TMs to work deterministically—that is, from any configuration, they can make no more than one move. As TMs are defined based on rewriting systems (see Definition 9.1), we make use of these systems and simply define deterministic TMs in the following way.

Definition 9.4 A TM is *deterministic* if it represents a rewriting system that is deterministic over ${}_M\mathrm{X}$ (see Definition 2.6). ■

In the proof of Theorem 9.5, we demonstrate how to turn any TM to an equivalent deterministic TM (of course, just like any rewriting systems, TMs are *equivalent* if they define the same language). As an exercise, give this proof in greater detail.

Theorem 9.5 From every TM I, we can construct equivalent deterministic TM O.

Proof. Let I be any TM. From I, we obtain an equivalent deterministic TM O so on every input string $x \in {}_M\Delta^*$, O works as follows. First, O saves x somewhere on the tape so this string is available whenever needed. Then, O systematically produces the sequences of the rules from ${}_IR$ on its tape, for instance, in the lexicographical order. Always after producing a sequence of the rules from ${}_IR$ in this way, O simulates the moves that I performs on x according to this sequence. If the sequence causes I to accept, O accepts as well; otherwise, it proceeds to the simulation according to the next sequence of rules. If there exists a sequence of moves according to which I accepts x, O eventually produces this sequence and accepts x, too. ■

Next, without affecting the power of TMs, we place further reasonable restrictions on the way deterministic TMs work.

Definition 9.6 Let M be a TM. If from $\chi \in {}_M\mathrm{X}$, M can make no move, then χ is a *halting configuration* of M. ■

Theorem 9.7 From every deterministic TM I, we can construct an equivalent deterministic TM $O = ({}_O\Sigma, {}_OR)$ such that ${}_OQ$ contains two new states, ◆ and ■, which do not occur on the left-hand side of any rule in ${}_OR$, ${}_OF = \{■\}$, and

I. Every halting configuration $\chi \in {}_O\mathrm{X}$ has the form $\chi = \triangleright qu \triangleleft$ with $q \in \{◆, ■\}$ and $u \in {}_O\Gamma^*$, and every non-halting configuration $\nu \in {}_O\mathrm{X}$ satisfies $\{◆, ■\} \cap \mathit{symbols}(\nu) = \varnothing$.
II. On every input string $x \in {}_O\Delta^*$, O performs one of these three kinds of computation:
   i. $\triangleright ▶ x \triangleleft \Rightarrow^* \triangleright ■ u \triangleleft$, where $u \in {}_O\Gamma^*$.
   ii. $\triangleright ▶ x \triangleleft \Rightarrow^* \triangleright ◆ v \triangleleft$, where $v \in {}_O\Gamma^*$.
   iii. Never enters any halting configuration.

Proof. Let I be a deterministic TM. From I, construct O satisfying the properties of Theorem 9.7 as follows. In both I and O, ▶ is the start state. Introduce ◆ and ■ as two new states into ${}_{O}Q$. Define ■ as the only final state in O; formally, set ${}_{O}F = \{\blacksquare\}$. On every input string x, O works as

1. Runs I on x.
2. If I halts in $\triangleright yqv\triangleleft$, where y, $v \in {}_{I}\Gamma^{*}$ and $q \in {}_{I}Q$, O continues from $\triangleright yqv\triangleleft$ and computes $\triangleright yqv\triangleleft \Rightarrow^{*} \triangleright qyv\triangleleft$.
3. If $q \in {}_{I}F$, O computes $\triangleright qyv\triangleleft \Rightarrow \triangleright\blacksquare yv\triangleleft$ and halts, and if $q \in {}_{I}Q - {}_{I}F$, O computes $\triangleright qyv\triangleleft \Rightarrow \triangleright\blacklozenge yv\triangleleft$ and halts.

As obvious, O satisfies the properties stated in Theorem 9.7. ■

We use Convention 9.8 in almost all proofs throughout the discussion concerning TMs in this book, so pay special attention to it.

Convention 9.8 In what follows, we automatically assume that every TM has the properties satisfied by O stated in Theorems 9.5 and 9.7. We denote the set of all these TMs by ${}_{TM}\Psi$. We set ${}_{TM}\Phi = \{L(M) | M \in {}_{TM}\Psi\}$ and refer to ${}_{TM}\Phi$ as the *family of Turing languages*.

Consider the three ways of computation described in part II of Theorem 9.7—(i), (ii), and (iii). Let $M \in {}_{TM}\Psi$ and $x \in {}_{M}\Delta^{*}$. We say that M *accepts* x iff on x, M makes a computation of the form (i). M *rejects* x iff on x, M makes a computation of the form (ii). M *halts* on x iff it accepts or rejects x; otherwise, M *loops* on x—in other words, M loops on x iff it performs a computation of the form (iii). States ■ and ◆ are referred to as the *accepting* and *rejecting states*, respectively; accordingly, configurations of the form $\triangleright\blacksquare u\triangleleft$ and $\triangleright\blacklozenge u\triangleleft$, where $u \in {}_{M}\Gamma^{*}$, are referred to as *accepting* and *rejecting configurations*, respectively.

We assume that Δ denotes the input alphabet of all TMs in what follows. Under this assumption, for brevity, we usually simply state that $M \in {}_{TM}\Psi$ works on an input string x instead of stating that M works on an input string x, where $x \in \Delta^{*}$. ■

According to Convention 9.8, we restrict our attention strictly to ${}_{TM}\Psi$ and ${}_{TM}\Phi$ in the sequel (a single exception is made in Section 10.2.5). Observe that this restriction is without any loss of generality because ${}_{TM}\Phi$ is also characterized by the general versions of TMs (see Definition 9.1) as follows from Theorems 9.5 and 9.7.

Next, we prove that every $L \in {}_{TM}\Phi$ is accepted by $O \in {}_{TM}\Psi$ that never rejects any input x—that is, either O accepts x or O loops on x. It is worth noting that we cannot reformulate this result so O never loops on any input. In other words, ${}_{TM}\Phi$ contains languages accepted only by TMs that loop on some inputs; we prove this important result and explain its crucial consequences in computer science as a whole in Section 10.2.3 of Chapter 10 (see Theorem 10.42).

Theorem 9.9 From any $I \in {}_{TM}\Psi$, we can construct $O \in {}_{TM}\Psi$ such that $L(I) = L(O)$ and O never rejects any input.

Proof. Consider any $I \in {}_{TM}\Psi$. In I, replace every rule with ◆ on its right-hand side with a set of rules that cause the machine to keep looping in the same configuration. Let O be the TM resulting

from this simple modification. Clearly, $L(I) = L(O)$ and O never rejects any input. A fully rigorous proof of this theorem is left to the reader. ■

9.2.2 Size Restrictions

By Theorem 9.10 and Corollary 9.11, we can always place a limit on the number of tape symbols in TMs.

Theorem 9.10 From any $I \in {}_{TM}\Psi$ with $card({}_I\Delta) \geq 2$, we can construct $O \in {}_{TM}\Psi$ with ${}_O\Gamma = {}_I\Delta \cup \{\square\}$.

Proof. Let $I = ({}_I\Sigma, {}_IR)$ be a TM in ${}_{TM}\Psi$ such that $a, b \in {}_I\Delta$. Let $2^{k-1} \leq card({}_I\Gamma) \leq 2^k$, for some $k \in \mathbb{N}$. We encode every symbol in ${}_I\Gamma - \{\square\}$ as a unique string of length k over $\{a, b\}$ by a function f from ${}_I\Gamma - \{\square\}$ to $\{a, b\}^k$. Based on f, define the homomorphism g from ${}_I\Sigma^*$ to $({}_IQ \cup \{\triangleright, \triangleleft, \square, a, b\})^*$ so that for all $Z \in {}_I\Gamma - \{\square\}$, $g(Z) = f(Z)$, and for all $Z \in ({}_IQ \cup \{\triangleright, \triangleleft, \square\})$, $g(Z) = Z$ (see Section 2.1 of Chapter 2 for the definition of homomorphism). Next, we construct a TM O that simulates I over the configurations encoded by g in the following way.

1. *Initialization*. Let $w = c_1c_2\ldots c_n$ be an input string, where $c_1, \ldots, c_n$ are input symbols from ${}_I\Delta$. O starts its computation on w by changing w to $g(w)$; in greater detail, it changes $\triangleright\blacktriangleright c_1c_2\ldots c_n\triangleleft$ to $g(\triangleright\blacktriangleright c_1c_2\ldots c_n\triangleleft)$, which equals $\triangleright\blacktriangleright f(c_1)f(c_2)\ldots f(c_n)\triangleleft$.
2. *Simulation of a Move*. If $\triangleright d_1d_2\ldots d_{i-1}qd_i\ldots d_m\triangleleft \in {}_I\mathrm{X}$ is the current configuration of I, where $q \in {}_IQ$ and $d_i \in {}_I\Gamma$, $1 \leq i \leq m$, then the corresponding configuration in ${}_O\mathrm{X}$ encoded by g is $g(\triangleright d_1d_2\ldots d_{i-1}qd_i\ldots d_m\triangleleft) = \triangleright f(d_1d_2\ldots d_{i-1})qf(d_i\ldots d_m)\triangleleft$. Let $\chi, \kappa \in {}_I\mathrm{X}$, and let I compute $\chi \Rightarrow \kappa$ by using $r \in {}_IR$. Then, O simulates $\chi \Rightarrow \kappa$ so it computes $g(\chi) \Rightarrow g(\kappa)$, during which it changes $g(\boldsymbol{lhs}(r))$ to $g(\boldsymbol{rhs}(r))$ by performing several moves.
3. *Simulation of a Computation*. O continues the simulation of moves in I one by one. If I makes a move by which it accepts, O also accepts; otherwise, O continues the simulation. ■

Notice that we can apply the encoding technique used in the proof of Theorem 9.10 even if a or b are not in ${}_I\Delta$, which gives rise to Corollary 9.11.

Corollary 9.11 Let $I \in {}_{TM}\Psi$. Then, there exists $O \in {}_{TM}\Psi$ with ${}_O\Gamma = \{a, b, \square\} \cup {}_I\Delta$. ■

Theorem 9.12, whose proof is left as an exercise, says we can also place a limit on the number of states in TMs without affecting their power.

Theorem 9.12 Let $I \in {}_{TM}\Psi$. Then, there exists $O \in {}_{TM}\Psi$ with $card({}_OQ) \leq 3$. ■

The bottom line of all the restricted versions of TMs discussed in this section is that they are as powerful as the general versions of TMs according to Definition 9.1. Of course, there

also exist restrictions placed on TMs that decrease their power. As a matter of fact, whenever we simultaneously place a limit on both the number of noninput tape symbols and the number of states, we decrease the power of TMs (a proof of this result is omitted because it is beyond the scope of this introductory text). Furthermore, in Chapters 10 and 11, we introduce some more restricted versions of TMs, such as Turing deciders in Section 10.2.1 and linear-bounded automata in Section 11.2.2, which are less powerful than TMs.

9.3 Universal Turing Machines

A formally described TM in ${}_{TM}\Psi$ resembles the machine code of a program executed by a computer, which thus acts as a universal device that executes all possible programs of this kind. Considering the subject of this chapter, we obviously want to know whether there also exists a TM acting as such a universal device, which simulates all machines in ${}_{TM}\Psi$. The answer is yes, and in this section, we construct a *universal TM* $U \in {}_{TM}\Psi$ that does the job—that is, U simulates every $M \in {}_{TM}\Psi$ working on any input w. However, because the input of any TM, including U, is always a string, we first show how to encode every $M \in {}_{TM}\Psi$ as a string, symbolically denoted by $\langle M \rangle$, from which U interprets M before it simulates its computation. To be quite precise, as its input, U has the code of M followed by the code of w, denoted by $\langle M, w \rangle$, from which U decodes M and w to simulate M working on w so U accepts $\langle M, w \rangle$ iff M accepts w. As a result, before the construction of U, we explain how to obtain $\langle M \rangle$ and $\langle M, w \rangle$ for every $M \in {}_{TM}\Psi$ and every input w.

9.3.1 Turing Machine Codes

Any reasonable encoding for TMs over a fixed alphabet $\vartheta \subseteq \Delta$ is acceptable provided that for every $M \in {}_{TM}\Psi$, U can mechanically and uniquely interpret $\langle M \rangle$ as M. Mathematically speaking, this encoding should represent a total function *code* from ${}_{TM}\Psi$ to ϑ^* such that $code(M) = \langle M \rangle$ for all $M \in {}_{TM}\Psi$. In addition, we select an arbitrary but fixed $Z \in {}_{TM}\Psi$ and define the decoding of TMs, *decode*, so for every $x \in range(code)$, $decode(x) = inverse(code(M))$ and for every $y \in \vartheta^* - range(code)$, $decode(y) = Z$ so $range(decode) = {}_{TM}\Psi$. As a result, *decode* is a total surjection because it maps every string in ϑ^*, including the strings that *code* maps to no machine in ${}_{TM}\Psi$, to a machine in ${}_{TM}\Psi$. Notice, on the other hand, that several strings in ϑ^* may be decoded to the same machine in ${}_{TM}\Psi$; mathematically, *decode* may not be an injection. From a more practical viewpoint, we just require that the mechanical interpretation of both *code* and *decode* is relatively easily performable. Apart from encoding and decoding all machines in ${}_{TM}\Psi$, we also use *code* and *decode* to encode and decode the pairs consisting of TMs and input strings. Next, we illustrate *code* and *decode* in binary.

A Binary Code for TMs. Consider any $M \in {}_{TM}\Psi$. Recall that we automatically apply Convention 9.2 to M, including the meaning of Q, F, Γ, Δ, and R. Consider Q as the set of states of M. Rename these states to $q_1, q_2, q_3, q_4, \ldots, q_m$ so $q_1 = \blacktriangleright$, $q_2 = \blacksquare$, $q_3 = \blacklozenge$, where $m = card(Q)$. Rename the symbols of $\{\triangleright, \triangleleft\} \cup \Gamma$ to $a_1, a_2, \ldots, a_n$ so $a_1 = \triangleright$, $a_2 = \triangleleft$, $a_3 = \square$, where $n = card(\Gamma) + 2$. Introduce the homomorphism h from $Q \cup \{\triangleright, \triangleleft\} \cup \Gamma$ to $\{0, 1\}^*$ as $h(q_i) = 10^i$, $1 \leq i \leq m$, and $h(a_j) = 110^j$, $1 \leq j \leq n$ (homomorphism is defined in Section 2.1). Extend h so it is defined from $(\{\triangleright, \triangleleft\} \cup \Gamma \cup Q)^*$ to $\{0, 1\}^*$ in the standard way—that is, $h(\varepsilon) = \varepsilon$, and $h(X_1 \ldots X_k) = h(X_1) h(X_2) \ldots h(X_k)$, where $k \geq 1$, and $X_l \in \{\triangleright, \triangleleft\} \cup \Gamma \cup Q$, $1 \leq l \leq k$ (see Section 2.1). Based on h, we now define the function *code* from R to $\{0, 1\}^*$ so that for each rule $r: x \rightarrow y \in R$, $code(r) = h(xy)$. Then, write the rules of R one after the other in an order as $r_1, r_2, \ldots, r_o$ with $o = card(R)$; for

instance, order them lexicographically. Set $code(R) = code(r_1)111code(r_2)111\ldots code(r_o)111$. Finally, from $code(R)$, we obtain the desired $code(M)$ by setting $code(M) = 0^m10^n1code(R)1$. Taking a closer look at $code(M) = 0^m10^n1code(R)1$, 0^m1 and 0^n1 state that $m = card(Q)$ and $n = card(\Gamma) + 2$, respectively, and $code(R)$ encodes the rules of R. Seen as a function from ${}_{TM}\Psi$ to $\{0, 1\}^*$, *code* obviously represents a total function ${}_{TM}\Psi$ to ϑ^*. On the other hand, there are binary strings that represent no legal code of any machine in ${}_{TM}\Psi$; mathematically, *inverse*(*code*) is a partial function, not a total function. For example, ε, any string in $\{0\}^* \cup \{1\}^*$, or any string that starts with 1 are illegal codes, so their inverses are undefined. Select an arbitrary but fixed $Z \in {}_{TM}\Psi$; for instance, take Z as a TM that immediately rejects every input after it starts its computation. Extend *inverse*(*code*) to the total function *decode* from $\{0, 1\}^*$ to ${}_{TM}\Psi$ so that *decode* maps all binary strings that represent no code of any TM in ${}_{TM}\Psi$ to Z. More precisely, for every $x \in \{0, 1\}^*$, if x is a legal code of a TM K in ${}_{TM}\Psi$, *decode* maps x to K, but if it is not, *decode* maps x to Z; equivalently and briefly, if x encodes $K \in {}_{TM}\Psi$ and, therefore, $x \in range(code)$, $decode(x) = K$, and if $x \in \{0, 1\}^* - range(code)$, $decode(x) = Z$ (notice that *decode* represents a surjection).

To encode every $w \in \Delta^*$, we simply set $code(w) = h(w)$, where h is the homomorphism defined above. Select an arbitrary but fixed $y \in \Delta^*$; for instance, take $y = \varepsilon$. Define the total surjection *decode* from $\{0, 1\}^*$ to Δ^* so for every $x \in \{0, 1\}^*$, if $x \in range(code)$, $decode(x) = inverse(code(w))$; otherwise, $decode(x) = y$.

For every $(M, w) \in {}_{TM}\Psi \times \Delta^*$, define $code(M, w) = code(M)code(w)$. Viewed as a function from ${}_{TM}\Psi \times \Delta^*$ to $\{0, 1\}^*$, *code* obviously represents a total function from ${}_{TM}\Psi \times \Delta^*$ to ϑ^*. Define the total surjection *decode* from $\{0, 1\}^*$ to ${}_{TM}\Psi \times \Delta^*$ so $decode(xy) = decode(x)decode(y)$, where $decode(x) \in {}_{TM}\Psi$ and $decode(y) \in \Delta^*$.

Example 9.3 Consider this trivial TM $M \in {}_{TM}\Psi$, where $M = (\Sigma, R)$, $\Sigma = Q \cup \Gamma \cup \{\triangleright, \triangleleft\}$, $Q = \{\blacktriangleright, \blacksquare, \blacklozenge, A, B, C, D\}$, $\Gamma = \Delta \cup \{\square\}$, $\Delta = \{b\}$, and R contains these rules

$$\blacktriangleright\triangleleft \rightarrow \blacksquare\triangleleft,\ \blacktriangleright b \rightarrow bA,\ Ab \rightarrow bB,\ Bb \rightarrow bA,\ A\triangleleft \rightarrow C\triangleleft,\ B\triangleleft \rightarrow D\triangleleft,$$
$$bD \rightarrow D\square,\ bC \rightarrow C\square,\ \triangleright C \rightarrow \triangleright\blacklozenge,\ \triangleright D \rightarrow \triangleright\blacksquare$$

Leaving a simple proof that $L(M) = \{b^i | \ i \geq 0,\ i \text{ is even}\}$ as an exercise, we next obtain the binary code of M by applying the encoding method described above. Introduce the homomorphism h from $Q \cup \{\triangleright, \triangleleft\} \cup \Gamma$ to $\{0, 1\}^*$ as $h(q_i) = 10^i$, $1 \leq i \leq 7$, where q_1, q_2, q_3, q_4, q_5, q_6, and q_7 coincide with $\blacktriangleright$, $\blacksquare$, $\blacklozenge$, A, B, C, and D, respectively, and $h(a_j) = 110^j$, $1 \leq j \leq 4$, where a_1, a_2, a_3, and a_4 coincide with $\triangleright$, $\triangleleft$, $\square$, and b, respectively. Extend h so it is defined from $(Q \cup \{\triangleright, \triangleleft\} \cup \Gamma)^*$ to $\{0, 1\}^*$ in the standard way. Based on h, define the function *code* from R to $\{0, 1\}^*$ so for each rule $x \rightarrow y \in R$, $code(x \rightarrow y) = h(xy)$. For example, $code(\blacktriangleright b \rightarrow bA) = 1011000011000010000$. Take, for instance, the above order of the rules from R, and set

$$\begin{aligned} code(R) = {} & code(\blacktriangleright\triangleleft \rightarrow \blacksquare\triangleleft)\ 111\ code(\blacktriangleright b \rightarrow bA)\ 111 \\ & code(Ab \rightarrow bB)\ 111\ code(Bb \rightarrow bA)\ 111 \\ & code(A\triangleleft \rightarrow C\triangleleft)\ 111\ code(B\triangleleft \rightarrow D\triangleleft)\ 111 \\ & code(bD \rightarrow D\square)\ 111\ code(bC \rightarrow C\square)\ 111 \\ & code(\triangleright C \rightarrow \triangleright\blacklozenge)\ 111\ code(\triangleright D \rightarrow \triangleright\blacksquare)\ 111 \end{aligned}$$

$$\begin{aligned} = {} & 101100100110011110110000110000100001111 \\ & 100001100001100001000001111000001100001100001000001111 \\ & 100001100100000011001111000001100100000001100111 \\ & 110000100000001000000011000111110000100000010000001100011 1 \\ & 11010000001101000111110100000001101001111 \end{aligned}$$

To encode M as a whole, set

$$\begin{aligned} code(M) &= 0^7 1 0^4 1 code(R) 1 \\ &= 0000000100001 \\ &\quad 1011001001100111101100001100001000011 \\ &\quad 1000011000011000010000011110000011000011000010000111 \\ &\quad 1000011001000000110011110000011001000000011100111 \\ &\quad 1100001000000010000000110001111100001000000100000011000111 \\ &\quad 1101000000110100011111010000000110100111 \\ &\quad 1 \end{aligned}$$

Take $w = bb$, whose $code(bb) = 110000110000$. As a result, the binary string denoted by $code(M, bb)$ is

$$\begin{aligned} &000000010000110110010011001111011000011000010000111100001100001100001000001111 \\ &100000110000110000100001111000011001000000110011110000011001000000011001111100 \\ &000100000001000000011000111110000100000010000001100011111010000001101000111111 \\ &010000000110100111111000011000 \end{aligned}$$

Convention 9.13 In what follows, we suppose there exist a fixed encoding and a fixed decoding of all TMs in ${}_{TM}\Psi$. We just require that both are uniquely and mechanically interpretable; otherwise, they may differ from *code* and *decode* (in fact, they may not even be in binary). As already stated in the beginning of this section, we denote the code of $M \in {}_{TM}\Psi$ by $\langle M \rangle$. Similarly, we suppose there exist an analogical encoding and decoding of the members of Δ^*, ${}_{TM}\Psi \times \Delta^*$, ${}_{TM}\Psi \times {}_{TM}\Psi$, and ${}_{TM}\Psi \times {}_0\mathbb{N}$. Again, for brevity, we denote the codes of $w \in \Delta^*$, $(M, w) \in {}_{TM}\Psi \times \Delta^*$, $(M, N) \in {}_{TM}\Psi \times {}_{TM}\Psi$, and $(M, i) \in {}_{TM}\Psi \times {}_0\mathbb{N}$ by $\langle w \rangle$, $\langle M, w \rangle$, $\langle M, N \rangle$, and $\langle M, i \rangle$, respectively (as an exercise, encode and decode the members of ${}_0\mathbb{N}$ similarly to encoding the machine in ${}_{TM}\Psi$). Even more generally, for any automaton or grammar, X, discussed in Chapters 2 through 8, $\langle X \rangle$ represents its code analogical to $\langle M \rangle$. ■

Out of all the terminology introduced in Convention 9.13, we just need $\langle M \rangle$ and $\langle M, w \rangle$ in the rest of Chapter 9.

9.3.2 Construction of Universal Turing Machines

We are now ready to construct U—that is, a universal TM (see Section 9.3). As a matter of fact, we construct two versions of U. The first version, denoted by ${}_{TM\text{-}Acceptance}U$, simulates every $M \in {}_{TM}\Psi$ on $w \in \Delta^*$ so ${}_{TM\text{-}Acceptance}U$ accepts $\langle M, w \rangle$ iff M accepts w. In other words, $L({}_{TM\text{-}Acceptance}U) = {}_{TM\text{-}Acceptance}L$ with

$${}_{TM\text{-}Acceptance}L = \{\langle M, w \rangle | \ M \in {}_{TM}\Psi, w \in \Delta^*, M \text{ accepts } w\}$$

The other version, denoted by ${}_{TM\text{-}Halting}U$, simulates every $M \in {}_{TM}\Psi$ on $w \in \Delta^*$ in such a way that ${}_{TM\text{-}Halting}U$ accepts $\langle M, w \rangle$ iff M halts on w (see Convention 9.8). To rephrase this in terms of formal languages, $L({}_{TM\text{-}Halting}U) = {}_{TM\text{-}Halting}L$ with

$${}_{TM\text{-}Halting}L = \{\langle M, w \rangle | \ M \in {}_{TM}\Psi, w \in \Delta^*, M \text{ halts on } w\}$$

Convention 9.14 Strictly speaking, in the proof of Theorem 9.15, we should state that ${}_{TM\text{-}Acceptance}U$ works on $\langle M, w\rangle$ so it first interprets $\langle M, w\rangle$ as M and w; then, it simulates the moves of M on w. However, instead of a long and obvious statement like this, we just state that ${}_{TM\text{-}Acceptance}U$ runs M on w. In a similar manner, we shorten the other proofs of results concerning TMs in the sequel whenever no confusion exists. ■

Theorem 9.15 There exists ${}_{TM\text{-}Acceptance}U \in {}_{TM}\Psi$ such that $L({}_{TM\text{-}Acceptance}U) = {}_{TM\text{-}Acceptance}L$.

Proof. On every input $\langle M, w\rangle$, ${}_{TM\text{-}Acceptance}U$ works so it runs M on w. ${}_{TM\text{-}Acceptance}U$ accepts $\langle M, w\rangle$ if and when it finds out that M accepts w; otherwise, ${}_{TM\text{-}Acceptance}U$ keeps simulating the moves of M in this way. ■

Observe that ${}_{TM\text{-}Acceptance}U$ represents a procedure, not an algorithm because if M loops on w, so does ${}_{TM\text{-}Acceptance}U$ on $\langle M, w\rangle$. As a matter of fact, in Section 10.2.3 of Chapter 10, we demonstrate that no TM can halt on every input and, simultaneously, act as a universal TM (see Theorem 10.43). To reformulate this in terms of formal languages, no TM accepts ${}_{TM\text{-}Acceptance}L$ in such a way that it halts on all strings. Indeed, for all $X \in {}_{TM}\Psi$ satisfying ${}_{TM\text{-}Acceptance}L = L(X)$, $\Delta^* - {}_{TM\text{-}Acceptance}L$ necessarily contains a string on which X loops.

By analogy with the proof of Theorem 9.15, we next obtain ${}_{TM\text{-}Halting}U$ that accepts ${}_{TM\text{-}Halting}L$, defined earlier.

Theorem 9.16 There exists ${}_{TM\text{-}Halting}U \in {}_{TM}\Psi$ such that $L({}_{TM\text{-}Halting}U) = {}_{TM\text{-}Halting}L$.

Proof. On every $\langle M, w\rangle$, ${}_{TM\text{-}Halting}U$ works so it runs M on w. ${}_{TM\text{-}Halting}U$ accepts $\langle M, w\rangle$ iff M halts w, which means that M either accepts w or rejects w (see Convention 9.8). Thus, ${}_{TM\text{-}Halting}U$ loops on $\langle M, w\rangle$ iff M loops on w, which means that $\langle M, w\rangle \notin {}_{TM\text{-}Halting}L$. Observe that $L({}_{TM\text{-}Halting}U) = {}_{TM\text{-}Halting}L$. ■

Exercises

1. This chapter contains results whose proofs are only sketched or even completely omitted. These results include Theorems 9.5 and 9.12, Example 9.2, and Convention 9.13. Complete them.
2. Construct three equivalent TMs; A, B, and C; so A accepts every input string by infinitely many sequences of moves, B accepts every input string by exactly two sequences of moves, and C is deterministic. Give a rigorous proof that A, B, and C are equivalent.
3. Consider the TM M from Example 9.1. Construct an equivalent TM that has fewer rules than M has. Give a proof that both TMs are equivalent.
4. Consider each of languages (i) through (xxiii). Construct a TM that accepts it. Define the constructed TM strictly formally. Give a rigorous proof that verifies the construction.
   i. $\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i < j < k\}$
   ii. $\{a^i b^j a^j \mid i, j \geq 0 \text{ and } j = i^2\}$
   iii. $\{a^i \mid i \text{ is not a prime}\}$
   iv. $\{w \mid w \in \{a, b, c\}^* \text{ and } (occur(w, a) \neq occur(w, b) \text{ or } occur(w, b) \neq occur(w, c))\}$
   v. $\{a^i b^j a^j \mid i, j \geq 0 \text{ and } j \neq i\}$
   vi. $\{a^i b^j c^{ij} \mid i, j \geq 0 \text{ and } i \neq j \neq 2i\}$
   vii. $\{a^i b^j c^k \mid i, j, k \geq 0, i \neq j, k \neq i, \text{ and } j \neq k\}$
   viii. $\{a^i b^j c^i d^j \mid i, j \geq 0 \text{ and } j \neq i\}$

ix. $\{a^i b^{2i} c^i \mid i \geq 0\}$
x. $\{ww \mid w \in \{a, b\}^*\}$
xi. $\{wvw \mid v, w \in \{a, b\}^*, reversal(v) = w\}$
xii. $\{0^i 1 0^i 1 0^i 1 0^i \mid i \geq 1\}$
xiii. $\{wcv \mid v, w \in \{a, b\}^*$ and $w = vv\}$
xiv. $\{a^i b^i a^j \mid i, j \geq 1\}$
xv. $\{a^i b^j \mid i \geq 0$ and $j \geq 1\}$
xvi. $\{a^i b^j c^k \mid i, j, k \geq 0$ and $i = j$ or $j = k\}$
xvii. $\{a^i b^j c^i b^j \mid i \geq 0$ and $j \geq 1\}$
xviii. $\{a^j c a b^i a^j \mid i, j \geq 0\}$
xix. $\{x \mid x \in \{a, b\}^*, aa \in substring(x), occur(x, a) = occur(x, b)\}$
xx. $\{x \mid x \in \{a, b, c\}^*, occur(x, a) \geq occur(x, b) \geq occur(x, c)\}$
xxi. $\{x \mid x \in \{a, b\}^*, occur(x, a) = 2^{occur(x, b)}\}$
xxii. $\{x \mid x \in \{a, b\}^*, occur(x, b) \geq occur(x, a), |x|$ is divisible by 3$\}$
xxiii. $\{x \mid x \in \{a, b, c\}^*, occur(x, a) \geq occur(x, b)$ iff $occur(x, c) < occur(x, a)\}$

5 S. Introduce a graph-based representation for TMs. Then, by using this representation, describe the TMs constructed in Exercise 4.

6. Consider the TM binary code in Section 9.3. Introduce an alternative binary code for TMs and rephrase all the discussion given in Section 9.3 in terms of this alternative code. Then, introduce a ternary code for this purpose and reformulate Section 9.3 in terms of this ternary code.

7. Consider the basic definition of a TM, M (see Definition 9.1). Restrict this definition so M changes the position of its tape head to the left or to the right during every single move; in other words, it never keeps its head stationary during any move. Define this restriction rigorously. Construct an algorithm that turns any TM to an equivalent TM restricted in this way. Verify this algorithm formally.

8. Generalize the definition of a TM by allowing a set of start states. Formalize this generalization. Construct an algorithm that turns any TM generalized in this way to an equivalent one-start-state TM, which satisfies Definition 9.1.

9. During every move, a *simple TM* cannot simultaneously change its state, the tape symbol, and the position of its head; otherwise, it works just like any TM. Formalize the notion of a simple TM. Construct an algorithm that converts any TM to an equivalent simple TM. Verify this algorithm by a rigorous proof.

10. During a single move, a *long-reading TM* can read a string, consisting of several tape symbols. Formalize this generalization. Construct an algorithm that turns any TM generalized in this way to an equivalent TM, defined according to Definition 9.1. Verify this algorithm by a rigorous proof.

11 S. A *two-way TM* M has its tape infinite both to the right and to the left. Initially, M occurs in its start state with an input string w placed on the tape. The tape head occurs over the leftmost symbol of w. Starting from this initial configuration, M works by analogy with the basic model of a TM (see Definition 9.1). As opposed to this basic model, however, M can always make a left move because its tape is infinite to both directions.

Formalize the notion of a two-way TM. Construct an algorithm that turns any two-way TM M to an equivalent TM, defined according to Definition 9.1. Verify this algorithm by a rigorous proof.

12 S. Let $k \in \mathbb{N}$. A *k-head TM* M is a TM with k tape heads over a single tape. A move made by M depends on the current state and the k symbols scanned by the tape heads. During a move, M changes its state and rewrites the k scanned symbols; in addition, M can change the position of any of its heads to the left or to the right. Consider the special case when n tape heads occur at the same tape position, for some $n \in \{2, \ldots, k\}$; at this point, M makes the next move only if all these n tape heads rewrite the scanned tape symbol to the same symbol. Initially, the tape contains the input string, each of the k heads scans its leftmost symbol, and M is in its start state. If from this initial configuration, M can make a sequence of moves that ends in a final state, then M accepts the input string. The language accepted by M consists of all strings that M accepts in this way.

Formalize the notion of a k-head TM. Construct an algorithm that turns any k-head TM M to an equivalent one-head TM (see Definition 9.1). Verify this algorithm by a rigorous proof.

13. Let $k \in \mathbb{N}$. A *k-tape TM M* represents a TM with k tapes, each of which has its read-write tape head. A move made by M depends on the current state and on the k symbols scanned by the k tape heads. During a move, M can change the current state and rewrite any of the k scanned symbols; in addition, it can change the position of any of its k heads. Initially, the first tape contains the input string, the other tapes are blank, each read-write head scans the leftmost symbol, and M occurs in its start state. If from this initial configuration, M can make a sequence of moves that ends in a final state, then M accepts the input string. The set of all strings that M accepts represents the language accepted by M.

    Formalize the notion of a k-tape TM. Construct an algorithm that turns any k-tape TM M to an equivalent one-tape TM, defined according to Definition 9.1.

14. We often accept a language whose strings satisfy a certain condition by a k-head TM because its multiple heads usually simplify verifying that a given input string w satisfies the condition in question. Typically, a verification of this kind is carried out so some of the k heads keep a finger on particular symbols on the tape while the other heads synchronously read some other symbols and, thereby, verify the condition. Consider each of the one-tape TMs constructed in Exercise 4. Denote the TM by I. Construct an equivalent k-head TM O (see Exercise 12) so O has fewer states, tape symbols, or rules than I has.
15. Reformulate and solve Exercise 14 in terms of two-way and k-tape TMs, introduced in Exercises 11 and 13, respectively.
16. Write a program that decides whether a given TM is deterministic.
17. Write a program that simulates any deterministic TM, M. Observe that in a general case, a program like this may enter an infinite loop because M may loop endlessly.
18. Write a program that simulates a universal TM (see Section 9.3). Just like in Exercise 17, a program like this may loop endlessly on some inputs.
19. Consider PDAs (see Section 6.3). Extend them to *two-PDAs* by adding another pushdown to them. Formalize this extension. Introduce table- and graph-based representations for them.

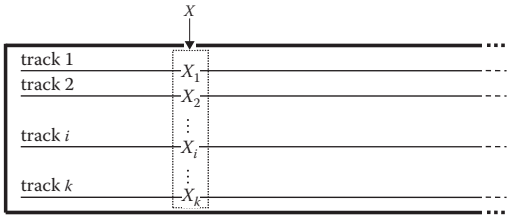
20 S. Construct an algorithm that turns any TM to an equivalent two-PDA (see Exercise 19).

21 S. As any two-PDA can be seen as a procedure, by the Church–Turing thesis, there exists an equivalent TM to it. This exercise, however, asks to demonstrate this result effectively by constructing an algorithm that converts any two-PDA to an equivalent TM.

Solutions to Selected Exercises

5. Let $M = (\Sigma, R)$ be any TM. In a pictorial way, M can be specified by its *state diagram*, which represents a labeled directed graph such that each node is labeled with a state $q \in Q$. To symbolically state that a state s is the start state, we point to it with an arrow. Final states are doubly circled. For two nodes $q, p \in Q$, there is an edge (q, p) if there is a rule $r \in R$ with q and p on its left-hand side and its right-hand side, respectively. If r is of the form $qX \rightarrow Yp \in R$, where $X, Y \in \Gamma$, then it is labeled by $\langle X/Y, right \rangle$, which says that M moves from q to p, rewrites X as Y on the tape, and moves its head to the right. Represent all other possible forms of rules analogically. Complete this solution by yourself.
11. Before giving the solution to this exercise, we intuitively sketch what we mean by the notion of a *k-tract tape* of a TM, where $k \in \mathbb{N}$, because the rest of Section IV, including this solution, makes use of it in what follows. Loosely speaking, at each position of a k-tract tape, there occurs a symbol X represented by a k-tuple $(X_1, \ldots, X_k)$. On a k-tract tape organized like this, the ith track contains the string consisting of the ith components of these k-tuples, where $1 \leq i \leq k$. We can realize X recorded so its k elements, X_1 through X_k, are vertically written above each other at this tape position; pictorially, a k-tract tape, organized in this way, can be sketched as in the figure below.

    Regarding Exercise 11, we only sketch an algorithm that converts any two-way TM I to an equivalent TM O, which satisfies Definition 9.1. Let $a_1 \ldots a_n$ be the input string of M with each a_i being an input symbol. By S, we denote the tape position at which a_1 initially occurs. O uses a two-track tape, so at each tape position, there occur two symbols above each



other—an upper symbol and a lower symbol. First, O places $a_1 \ldots a_n$ as the n upper symbols on the first tape track while setting all the n corresponding lower symbols to blanks on the second track. If I makes a move at S or to the right of S, O straightforwardly simulates this move within the first track. If M makes a move with its tape head to the left of S, O simulates this move by using the lower symbols placed on the second track (in the direction opposite to the direction in which I moves). Moves I and II deserve our special attention:

I. When I makes a move from S to the left, O first leaves the leftmost upper symbol on the first tape track for the leftmost lower symbol on the second tape track, after which it simulates the move. Analogically, when I makes a move during which it enters S from the left, O first leaves the first lower symbol for the first upper symbol, after which it performs the move simulation.

II. If I extends its tape at either end of the tape, O extends its tape by two blanks placed above each other on both tracks.

Naturally, O accepts $a_1 \ldots a_n$ when and if I accepts it.

12. We only sketch how to convert any k-head TM I to an equivalent one-head TM O, satisfying Definition 9.1. O uses a $(k+1)$-tract tape (see the solution to Exercise 11). Denote the tracks $t_0, t_1, \ldots, t_k$. Track t_0 corresponds to the original tape of I. The other tracks correspond to the k heads. That is, t_j is completely blank except for a single occurrence of a state placed at the tape position corresponding to the symbol scanned by the jth head of I, where $j = 1, \ldots, k$. Consequently, the entire tape holds k state occurrences, all of which specify the same state, which coincides with the current state of the simulated TM I. To simulate a move in I, O sweeps right t_1 through t_k on its tape to find out whether I has a rule applicable in the current configuration. If not, O rejects. If it has a rule r, O simulates the move according to r on t_0 and updates $t_1, \ldots, t_k$ accordingly. O accepts its input when and if I does.

20. We give only a gist of an algorithm that turns any TM to an equivalent two-PDA (see Exercise 19). Let I be a TM. From I, construct a two-PDA O that simulates I by using its two pushdowns as follows. O stores the symbols to the left of the tape head onto one pushdown so that symbols closer to the tape head appear closer to the pushdown top than symbols further from the tape head. Analogously, O stores the symbols to the right of the tape head onto the other pushdown. By using these two pushdowns, O simulates moves made by I. O accepts its input if and when I accepts it.

21. We only sketch an algorithm that turns any two-PDA to an equivalent TM. Let I be a two-PDA. From I, construct a three-tape TM O (see Exercise 13), which uses its tapes as follows. The first tape contains the input string of I. The second tape simulates one pushdown of I while the third tape simulates the other pushdown of I. By using its tapes in this way, O simulates I move by move. O accepts its input if and when I does. Convert O to an equivalent one-tape TM (see Exercise 13). The resulting TM is equivalent to I.