PART III

Computability

We now begin our exploration of the capabilities and limitations of algorithmic computation. The term *effective procedure* is used to describe processes that we intuitively understand as computable. An effective procedure consists of a finite set of instructions and a specification, based on the input, of the order of execution of the instructions. The execution of an instruction is mechanical; it requires no cleverness or ingenuity on the part of the machine or person doing the computation. A computation produced by an effective procedure executes a finite number of instructions and terminates. The preceding properties can be summarized as follows: An effective procedure is a deterministic discrete process that halts for all possible inputs.

In 1936 British mathematician Alan Turing designed a family of abstract machines for performing effective computation. The Turing machine represents the culmination of a series of increasingly powerful abstract computing devices that include finite and pushdown automata. As with a finite automaton, the applicable Turing machine instruction is determined by the state of the machine and the symbol being read. A Turing machine may read its input multiple times and an instruction may write information to memory. The ability to perform multiple reads and writes increases the computational power of the Turing machine and provides a theoretical prototype for the modern computer.

The Church-Turing Thesis, proposed by logician Alonzo Church in 1936, asserts that any effective computation in any algorithmic system can be accomplished using a Turing machine. The Church-Turing Thesis should not be considered as providing a definition of algorithmic computation—this would be an extremely limiting viewpoint. Many systems have been designed to perform effective computations. Moreover, who can predict the formalisms and techniques that will be developed in the future? The Church-Turing Thesis does not claim that these other systems do not perform algorithmic computation. It does, however, assert that a computation performed in any such system can be accomplished by a suitably designed Turing machine. Perhaps the strongest evidence supporting the Church-Turing Thesis is that after 70 years, no counterexamples have been discovered. The formulation of this thesis and its implications for computability are discussed in Chapter 11.

The correspondence between the generation of languages by grammars and their recognition by machines extends to the languages of Turing machines. If Turing machines represent the ultimate in string recognition machines, it seems reasonable to expect the associated family of grammars to be the most general string transformation systems. This is indeed the case; the grammars that correspond to Turing machines are called unrestricted grammars because there are no restrictions on the form or the applicability of their rules. To establish the correspondence between recognition by a Turing machine and generation by an unrestricted grammar, we show that a computation of a Turing machine can be simulated by a derivation in an unrestricted grammar.

With the acceptance of the Church-Turing Thesis, the extent of algorithmic problem solving can be identified with the capabilities of Turing machine computations. Consequently, to prove a problem to be unsolvable, it suffices to show that there is no Turing machine solution to the problem. Using this approach, we show that the Halting Problem for Turing machines is undecidable. That is, there is no algorithm that can determine, for an arbitrary Turing machine M and string w, whether M will halt when run with w. We will then use problem reduction to establish undecidability of additional questions about the results of Turing machine computations, of the existence of derivations using the rules of a grammar, and of properties of context-free languages.

CHAPTER 8

Turing Machines

The Turing machine, introduced by Alan Turing in 1936, represents another step in the development of finite-state computing machines. Turing machines were originally proposed for the study of effective computation and exhibit many of the features commonly associated with a modern computer. This is no accident; the Turing machine provided a model for the design and development of the stored-program computer. Utilizing a sequence of elementary operations, a Turing machine may access and alter any memory position. A Turing machine, unlike a computer, has no limitation on the amount of time or memory available for a computation.

The Church-Turing Thesis, which will be discussed in detail in Chapter 11, asserts that any effective procedure can be realized by a suitably designed Turing machine. The variations of Turing machine architectures and applications presented in the next two chapters indicate the robustness and the versatility of Turing machine computation.

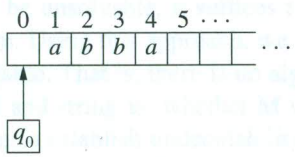
8.1 The Standard Turing Machine

A Turing machine is a finite-state machine in which a transition prints a symbol on the tape. The tape head may move in either direction, allowing the machine to read and manipulate the input as many times as desired. The structure of a Turing machine is similar to that of a finite automaton, with the transition function incorporating these additional features.

Definition 8.1.1

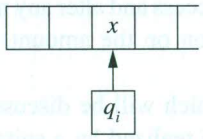
A **Turing machine** is a quintuple $M = (Q, \Sigma, \Gamma, \delta, q_0)$ where Q is a finite set of states, Γ is a finite set called the *tape alphabet*, Γ contains a special symbol B that represents a blank, Σ is a subset of $\Gamma - \{B\}$ called the *input alphabet*, δ is a partial function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$ called the *transition function*, and $q_0 \in Q$ is a distinguished state called the *start state*.

The tape of a Turing machine has a left boundary and extends indefinitely to the right. Tape positions are numbered by the natural numbers, with the leftmost position numbered zero. Each tape position contains one element from the tape alphabet.

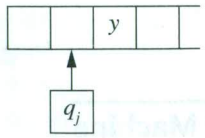


A computation begins with the machine in state q_0 and the tape head scanning the leftmost position. The input, a string from Σ^*, is written on the tape beginning at position one. Position zero and the remainder of the tape are blank. The diagram shows the initial configuration of a Turing machine with input *abba*. The tape alphabet provides additional symbols that may be used during a computation.

A transition consists of three actions: changing the state, writing a symbol on the square scanned by the tape head, and moving the tape head. The direction of the movement is specified by the final component of the transition. An L indicates a move of one tape position to the left and R one position to the right. The machine configuration



and transition $\delta(q_i, x) = [q_j, y, L]$ combine to produce the new configuration



The transition changed the state from q_i to q_j, replaced the tape symbol x with y, and moved the tape head one square to the left. The ability of the machine to move in both directions and process blanks introduces the possibility of a computation continuing indefinitely.

A computation halts when it encounters a state, symbol pair for which no transition is defined. A transition from tape position zero may specify a move to the left of the boundary of the tape. When this occurs, the computation is said to *terminate abnormally*. When we say that a computation halts, we mean that it terminates in a normal fashion.

The Turing machine presented in Definition 8.1.1 is deterministic, that is, at most one transition is specified for every combination of state and tape symbol. The one-tape deterministic Turing machine, with initial conditions as described above, is referred to as the **standard Turing machine**. The first two examples demonstrate the use of Turing machines to manipulate strings. After developing a facility with Turing machine computations, we will use Turing machines to accept languages and to compute functions.

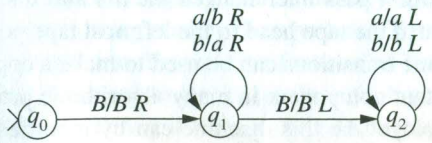
Example 8.1.1

The tabular representation of the transition function of a standard Turing machine with input alphabet $\{a, b\}$ is given in the table below.

δ	B	a	b
q_0	q_1, B, R		
q_1	q_2, B, L	q_1, b, R	q_1, a, R
q_2		q_2, a, L	q_2, b, L

The transition from state q_0 moves the tape head to position one to read the input. The transitions in state q_1 read the input string and interchange the symbols a and b. The transitions in q_2 return the machine to the initial position.

A Turing machine can be graphically represented by a state diagram. The transition $\delta(q_i, x) = [q_j, y, d]$, $d \in \{L, R\}$ is depicted by an arc from q_i to q_j labeled $x/y\ d$. The state diagram



represents the Turing machine defined in the preceding transition table. □

A machine configuration consists of the state, the tape, and the position of the tape head. At any step in a computation of a standard Turing machine, only a finite segment of the tape is nonblank. A configuration is denoted uq_ivB, where all tape positions to the right of the B are blank and uv is the string spelled by the symbols on the tape from the left-hand boundary to the B. Blanks may occur in the string uv; the only requirement is that the

ntire nonblank portion of the tape be included in uv. The notation uq_ivB indicates that the nachine is in state q_i scanning the first symbol of v and the entire tape to the right of uvB s blank.

This representation of machine configurations can be used to trace the computations f a Turing machine. The notation $uq_ivB \vdash_M xq_jyB$ indicates that the configuration xq_jyB s obtained from uq_ivB by a single transition of M. Following the standard conventions, $q_ivB \vdash_M^* xq_jyB$ signifies that xq_jyB can be obtained from uq_ivB by a finite number, ossibly zero, of transitions. The reference to the machine is omitted when there is no ossible ambiguity.

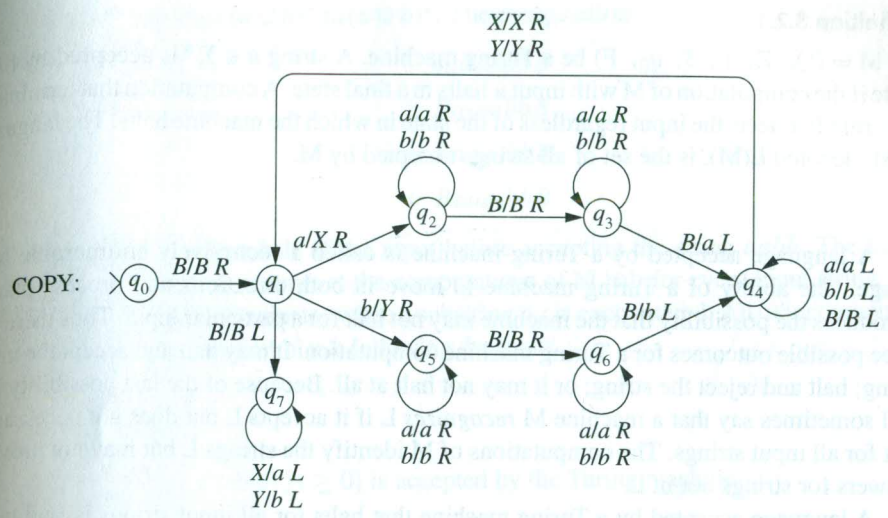
The Turing machine in Example 8.1.1 interchanges the a's and b's in the input string. racing the computation generated by the input string $abab$ yields

$$
\begin{aligned}
& q_0BababB \\
\vdash\ & Bq_1ababB \\
\vdash\ & Bbq_1babB \\
\vdash\ & Bbaq_1abB \\
\vdash\ & Bbabq_1bB \\
\vdash\ & Bbabaq_1B \\
\vdash\ & Bbabq_2aB \\
\vdash\ & Bbaq_2baB \\
\vdash\ & Bbq_2abaB \\
\vdash\ & Bq_2babaB \\
\vdash\ & q_2BbabaB.
\end{aligned}
$$

The Turing machine from Example 8.1.1 made two passes through the input string. Moving left to right, the first pass interchanged the a's and b's. The second pass, going ight to left, simply returned the tape head to the leftmost tape position. The next example hows how Turing machine transitions can be used to make a copy of a string. The ability o copy data is an important component in many algorithmic processes. When copies are eeded, the strategy employed be this machine can by modified to suit the type of data onsidered in the particular problem.

xample 8.1.2

he Turing machine COPY with input alphabet $\{a, b\}$ produces a copy of the input string. hat is, a computation that begins with the tape having the form BuB terminates with tape $BuBuB$.



The computation copies the input string one symbol at a time beginning with the leftmost symbol in the input. Tape symbols X and Y record the portion of the input that has been copied. The first unmarked symbol in the string specifies the arc to be taken from state q_1. The cycle q_1, q_2, q_3, q_4, q_1 replaces an a with X and adds an a to the string being constructed. Similarly, the lower branch copies a b using Y to mark the input string. After the entire string has been copied, the transitions in state q_7 change the X's and Y's to a's and b's and return the tape head to the initial position. □

8.2 Turing Machines as Language Acceptors

Turing machines have been introduced as a paradigm for effective computation. A Turing machine computation consists of a sequence of elementary operations determined from the machine state and the symbol being read by the tape head. The machines constructed in the previous section were designed to illustrate the features of Turing machine computations. The computations read and manipulated the symbols on the tape; no interpretation was given to the result of a computation. Turing machines can be designed to accept languages and to compute functions. The result of a computation can be defined in terms of the state in which the computation terminates or the configuration of the tape at the end of the computation.

In this section we consider the use of Turing machines as language acceptors; a computation accepts or rejects the input string. Initially, acceptance is defined by the final state of the computation. This is similar to the technique used by finite-state and pushdown automata to accept strings. Unlike finite-state and pushdown automata, a Turing machine need not read the entire input string to accept the string. A Turing machine augmented with final states is a sextuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where $F \subseteq Q$ is the set of final states.

Definition 8.2.1

Let M = (Q, Σ, Γ, δ, q_0, F) be a Turing machine. A string $u \in \Sigma^*$ is **accepted by final state** if the computation of M with input u halts in a final state. A computation that terminates abnormally rejects the input regardless of the state in which the machine halts. The language of M, denoted L(M), is the set of all strings accepted by M.

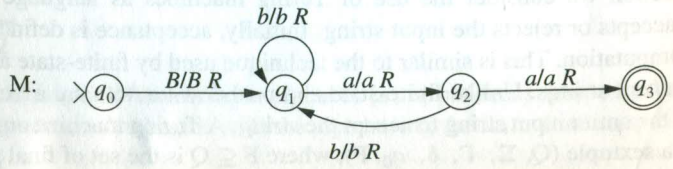
A language accepted by a Turing machine is called a **recursively enumerable language**. The ability of a Turing machine to move in both directions and process blanks introduces the possibility that the machine may not halt for a particular input. Thus there are three possible outcomes for a Turing machine computation: It may halt and accept the input string; halt and reject the string; or it may not halt at all. Because of the last possibility, we will sometimes say that a machine M *recognizes* L if it accepts L but does not necessarily halt for all input strings. The computations of M identify the strings L but may not provide answers for strings not in L.

A language accepted by a Turing machine that halts for all input strings is said to be **recursive**. Membership in a recursive language is decidable; the computations of a Turing machine that halts for all inputs provide a procedure for determining whether a string is in the language. A Turing machine of this type is sometimes said to *decide* the language. Being recursive is a property of a language, not of a Turing machine that accepts it. There are multiple Turing machines that accept a particular language; some may halt for all input, whereas others may not. The existence of one Turing machine that halts for all inputs is sufficient to show that the membership in the language is decidable and the language is recursive.

In Chapter 12 we will show that there are languages that are recognized by a Turing machine but are not decided by any Turing machine. It follows that the set of recursive languages is a proper subset of the recursively enumerable languages. The terms *recursive* and *recursively enumerable* have their origins in the functional interpretation of Turing computability that will be presented in Chapter 13.

Example 8.2.1

The Turing machine M



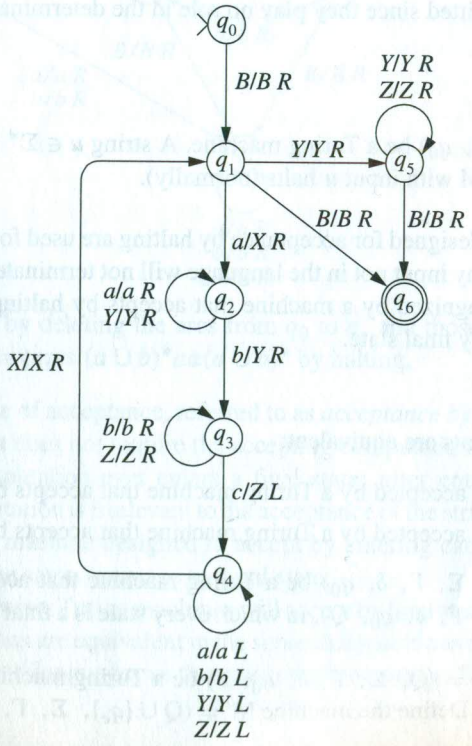
accepts the language $(a \cup b)^*aa(a \cup b)^*$. The computation

$$q_0BaabbB$$
$$\vdash Bq_1aabbB$$
$$\vdash Baq_2abbB$$
$$\vdash Baaq_3bbB$$

examines only the first half of the input before accepting the string *aabb*. The language $(a \cup b)^*aa(a \cup b)^*$ is recursive; the computations of M halt for every input string. A successful computation terminates when a substring *aa* is encountered. All other computations halt upon reading the first blank following the input. □

Example 8.2.2

The language L = $\{a^ib^ic^i \mid i \geq 0\}$ is accepted by the Turing machine



The tape symbols X, Y, and Z mark the a's, b's, and c's as they are matched. A computation successfully terminates when all the symbols in the input string have been transformed to the appropriate tape symbol. The transition from q_1 to q_6 accepts the null string.

The Turing machine M shows that L is recursive. The computations for strings in L halt in q_6. For strings not in L, the computations halt in a nonaccepting state as soon as it is discovered that the input string does not match the pattern $a^i b^i c^i$. For example, the computation with input *bca* halts in q_1 and with input *abb* in q_3. □

8.3 Alternative Acceptance Criteria

Using Definition 8.2.1, the acceptance of a string by a Turing machine is determined by the state of the machine when the computation halts. Alternative approaches to defining acceptance are presented in this section.

The first alternative is acceptance by halting. In a Turing machine that is designed to accept by halting, an input string is accepted if the computation initiated with the string halts. Computations for which the machine terminates abnormally reject the string. When acceptance is defined by halting, the machine is defined by the quintuple $(Q, \Sigma, \Gamma, \delta, q_0)$. The final states are omitted since they play no role in the determination of the language of the machine.

Definition 8.3.1

Let $M = (Q, \Sigma, \Gamma, \delta, q_0)$ be a Turing machine. A string $u \in \Sigma^*$ is **accepted by halting** if the computation of M with input u halts (normally).

Turing machines designed for acceptance by halting are used for language recognition. The computation for any input not in the language will not terminate. Theorem 8.3.2 shows that any language recognized by a machine that accepts by halting is also accepted by a machine that accepts by final state.

Theorem 8.3.2

The following statements are equivalent:

i) The language L is accepted by a Turing machine that accepts by final state.

ii) The language L is accepted by a Turing machine that accepts by halting.

Proof. Let $M = (Q, \Sigma, \Gamma, \delta, q_0)$ be a Turing machine that accepts L by halting. The machine $M' = (Q, \Sigma, \Gamma, \delta, q_0, Q)$, in which every state is a final state, accepts L by final state.

Conversely, let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a Turing machine that accepts the language L by final state. Define the machine $M' = (Q \cup \{q_e\}, \Sigma, \Gamma, \delta', q_0)$ that accepts by halting as follows:

i) If $\delta(q_i, x)$ is defined, then $\delta'(q_i, x) = \delta(q_i, x)$.

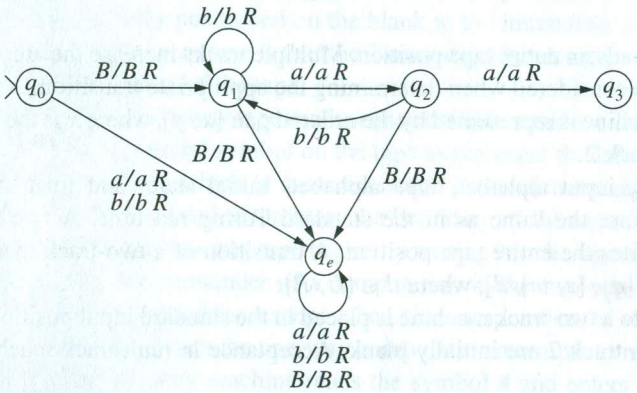
ii) For each state $q_i \in Q - F$, if $\delta(q_i, x)$ is undefined, then $\delta'(q_i, x) = [q_e, x, R]$.

iii) For each $x \in \Gamma$, $\delta'(q_e, x) = [q_e, x, R]$.

Computations that accept strings in M and M′ are identical. An unsuccessful computation in M may halt in a rejecting state, terminate abnormally, or fail to terminate. When an unsuccessful computation in M halts, the computation in M′ enters the state q_e. Upon entering q_e, the machine moves indefinitely to the right. The only computations that halt in M′ are those that are generated by computations of M that halt in an accepting state. Thus $L(M') = L(M)$. ■

Example 8.3.1

The Turing machine from Example 8.2.1 is altered to accept $(a \cup b)^*aa(a \cup b)^*$ by halting. The machine below is constructed as specified by Theorem 8.3.2. A computation enters q_e when the entire input string has been read and no *aa* has been encountered.



The machine obtained by deleting the arcs from q_0 to q_e and those from q_e to q_e labeled $a/a\ R$ and $b/b\ R$ also accepts $(a \cup b)^*aa(a \cup b)^*$ by halting. □

In Exercise 7 a type of acceptance, referred to as *acceptance by entering*, is introduced that uses final states but does not require the accepting computations to terminate. A string is accepted if the computation ever enters a final state; after entering a final state, the remainder of the computation is irrelevant to the acceptance of the string. As with acceptance by halting, any Turing machine designed to accept by entering can be transformed into a machine that accepts the same language by final state.

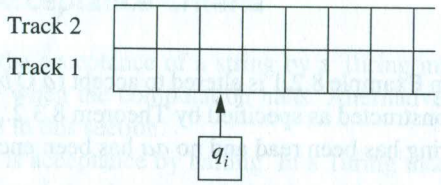
Unless noted otherwise, Turing machines will accept by final state as in Definition 8.2.1. The alternative definitions are equivalent in the sense that machines designed in this manner accept the same family of languages as those accepted by standard Turing machines.

8.4 Multitrack Machines

The remainder of the chapter is dedicated to examining variations of the standard Turing machine model. Each of the variations appears to increase the capability of the machine.

We prove that the languages accepted by these generalized machines are precisely those accepted by the standard Turing machines. Additional variations will be presented in the exercises.

A multitrack tape is one in which the tape is divided into tracks. A tape position in an n-track tape contains n symbols from the tape alphabet. The diagram depicts a two-track tape with the tape head scanning the second position.



The machine reads an entire tape position. Multiple tracks increase the amount of information that can be considered when determining the appropriate transition. A tape position in a two-track machine is represented by the ordered pair $[x, y]$, where x is the symbol in track 1 and y is in track 2.

The states, input alphabet, tape alphabet, initial state, and final states of a two-track machine are the same as in the standard Turing machine. A two-track transition reads and rewrites the entire tape position. A transition of a two-track machine is written $\delta(q_i, [x, y]) = [q_j, [z, w], d]$, where $d \in \{L, R\}$.

The input to a two-track machine is placed in the standard input position in track 1. All the positions in track 2 are initially blank. Acceptance in multitrack machines is by final state.

We will show that the languages accepted by two-track machines are precisely the recursively enumerable languages. The argument easily generalizes to n-track machines.

Theorem 8.4.1

A language L is accepted by a two-track Turing machine if, and only if, it is accepted by a standard Turing machine.

Proof. Clearly, if L is accepted by a standard Turing machine, it is accepted by a two-track machine. The equivalent two-track machine simply ignores the presence of the second track.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a two-track machine. A one-track machine will be constructed in which a single tape square contains the same information as a tape position in the two-track tape. The representation of a two-track tape position as an ordered pair indicates how this can be accomplished. The tape alphabet of the equivalent one-track machine M' consists of ordered pairs of tape elements of M. The input to the two-track machine consists of ordered pairs whose second component is blank. The input symbol a of M is identified with the ordered pair $[a, B]$ of M'. The one-track machine

$$M' = (Q, \Sigma \times \{B\}, \Gamma \times \Gamma, \delta', q_0, F)$$

with transition function

$$\delta'(q_i, [x, y]) = \delta(q_i, [x, y])$$

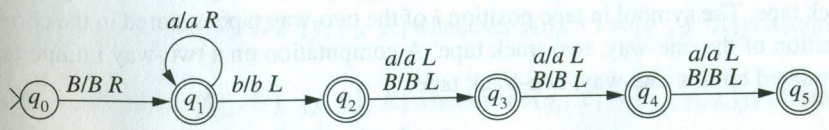
accepts L(M). ■

8.5 Two-Way Tape Machines

A Turing machine with a two-way tape is identical to the standard model except that the tape extends indefinitely in both directions. Since a two-way tape has no left boundary, the input can be placed anywhere on the tape. All other tape positions are assumed to be blank. The tape head is initially positioned on the blank to the immediate left of the input string. The advantage of a two-way tape is that the Turing machine designer need not worry about crossing the left boundary of the tape.

A machine with a two-way tape can be constructed to simulate the actions of a standard machine by placing a special symbol on the tape to represent the left boundary of the one-way tape. The symbol #, which is assumed not to be an element of the tape alphabet of the standard machine, is used to simulate the boundary of the tape. A computation in the equivalent machine with two-way tape begins by writing # to the immediate left of the initial tape head position. The remainder of a computation in the two-way machine is identical to that of the one-way machine except when the computation of the one-way machine terminates abnormally. When the one-way computation attempts to move to the left of the tape boundary, the two-way machine reads the symbol # and enters a nonaccepting state that terminates the computation.

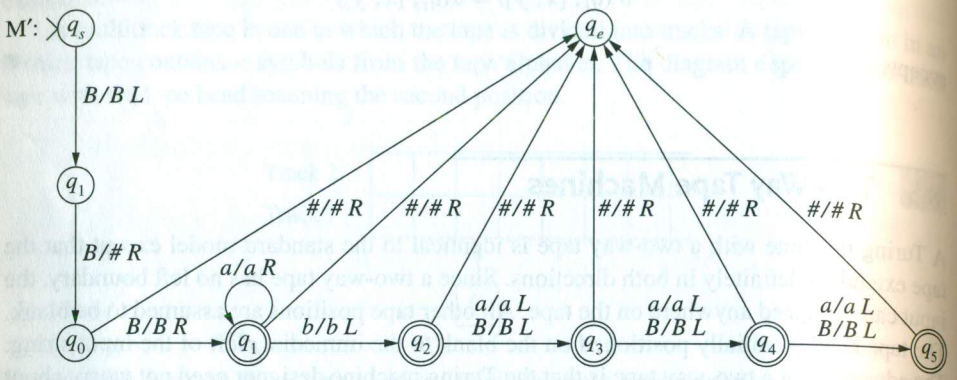
The standard Turing machine M



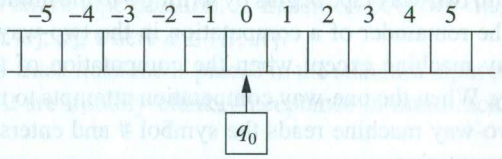
will be used to demonstrate the conversion of a machine with a one-way tape to an equivalent two-way machine. All the states of M other than q_0 are accepting. When the first b is encountered, the tape head moves four positions to the left, if possible. Acceptance is completely determined by the boundary of the tape. A string is rejected by M whenever the tape head attempts to cross the left-hand boundary. All computations that remain within the bounds of the tape accept the input. Thus the language of M consists of all strings over $\{a, b\}$ in which the first b, if present, is preceded by at least three a's.

A machine M' with a two-way tape can be obtained from M by the addition of three states q_s, q_t, and q_e. The transitions from states q_s and q_t insert the simulated endmarker to the left of the initial position of the tape head of M', the two-way machine that accepts L(M). After writing the simulated boundary, the computation enters a copy of the one-way

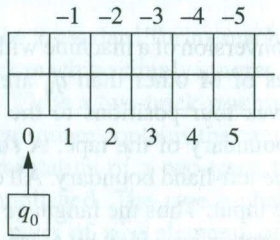
machine M. The error state q_e is entered in M′ when a computation in M attempts to move to the left of the tape boundary.



We will now show that a language accepted by a machine with a two-way tape is accepted by a standard Turing machine. The argument utilizes Theorem 8.4.1, which establishes the interdefinability of two-track and standard machines. The tape positions of the two-way tape can be numbered by the complete set of integers. The initial position of the tape head is numbered zero, and the input begins at position one.



Imagine taking the two-way infinite tape and folding it so that position $-i$ sits directly above position i. Adding an unnumbered tape square over position zero produces a two-track tape. The symbol in tape position i of the two-way tape is stored in the corresponding position of the one-way, two-track tape. A computation on a two-way infinite tape can be simulated on this one-way, two-track tape.



Let M = (Q, Σ, Γ, δ, q_0, F) be a Turing machine with a two-way tape. Using the correspondence between a two-way tape and a two-track tape, we construct a Turing machine M′ with a two-track, one-way tape to accept L(M). A transition of M is specified by the state and the symbol scanned. M′, scanning a two-track tape, reads two symbols at each

tape position. Symbols U (up) and D (down) are included in the states of M′ to designate which of the two tracks should be used to determine the transition.

The components of M′ are constructed from those of M and the symbols U and D:

$$Q' = (Q \cup \{q_s, q_t\}) \times \{U, D\}$$

$$\Sigma' = \Sigma$$

$$\Gamma' = \Gamma \cup \{\#\}$$

$$F' = \{[q_i, U],\ [q_i, D] \mid q_i \in F\}.$$

The initial state of M′ is a pair $[q_s, D]$. The transition from this state writes the marker # on the upper track in the leftmost tape position.

A transition from $[q_t, D]$ returns the tape head to its original position to begin the simulation of a computation of M. During the remainder of a computation, the # on track 2 is used to indicate when the tape head is reading position zero and to trigger changes from U to D in the state. The transitions of M′ are defined as follows:

1. $\delta'([q_s, D],\ [B, B]) = [[q_t, D],\ [B, \#],\ R]$.
2. For every $x \in \Gamma$, $\delta'([q_t, D],\ [x, B]) = [[q_0, D],\ [x, B],\ L]$.
3. For every $z \in \Gamma - \{\#\}$ and $d \in \{L, R\}$, $\delta'([q_i, D],\ [x, z]) = [[q_j, D],\ [y, z],\ d]$ whenever $\delta(q_i, x) = [q_j, y, d]$ is a transition of M.
4. For every $x \in \Gamma - \{\#\}$ and $d \in \{L, R\}$, $\delta'([q_i, U],\ [z, x]) = [[q_j, U],\ [z, y],\ d']$ whenever $\delta(q_i, x) = [q_j, y, d]$ is a transition of M, where d' is the opposite direction of d.
5. $\delta'([q_i, D],\ [x, \#]) = [[q_j, U],\ [y, \#],\ R]$ whenever $\delta(q_i, x) = [q_j, y, L]$ is a transition of M.
6. $\delta'([q_i, D],\ [x, \#]) = [[q_j, D],\ [y, \#],\ R]$ whenever $\delta(q_i, x) = [q_j, y, R]$ is a transition of M.
7. $\delta'([q_i, U],\ [x, \#]) = [[q_j, D],\ [y, \#],\ R]$ whenever $\delta(q_i, x) = [q_j, y, R]$ is a transition of M.
8. $\delta'([q_i, U],\ [x, \#]) = [[q_j, U],\ [y, \#],\ R]$ whenever $\delta(q_i, x) = [q_j, y, L]$ is a transition of M.

A transition generated by schema 3 simulates a transition of M in which the tape head begins and ends in positions labeled with nonnegative values. In the simulation, this is represented by writing on the lower track of the tape. Transitions defined in schema 4 use only the upper track of the two-track tape. These correspond to transitions of M that occur to the left of position zero on the two-way infinite tape.

The remaining transitions simulate the transitions of M from position zero on the two-way tape. Regardless of the U or D in the state, transitions from position zero are determined by the tape symbol on track 1. When the track is specified by D, the transition is defined by schema 5 or 6. Transitions defined in 7 and 8 are applied when the state is $[q_i, U]$.

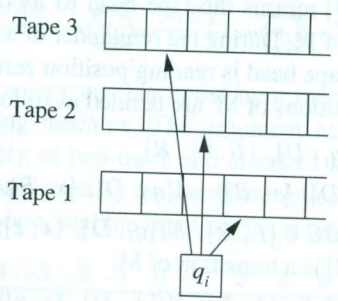
The preceding informal arguments outline the proof of the equivalence of one-way and two-way Turing machines.

Theorem 8.5.1

A language L is accepted by a Turing machine with a two-way tape if, and only if, it is accepted by a standard Turing machine.

8.6 Multitape Machines

A k-tape machine has k tapes and k independent tape heads. The states and alphabets of a multitape machine are the same as in a standard Turing machine. The machine reads the tapes simultaneously but has only one state. This is depicted by connecting each of the independent tape heads to a single control indicating the current state.



A transition is determined by the state and the symbols scanned by each of the tape heads. A transition in a multitape machine may

i) change the state,

ii) write a symbol on each of the tapes,

iii) independently reposition each of the tape heads.

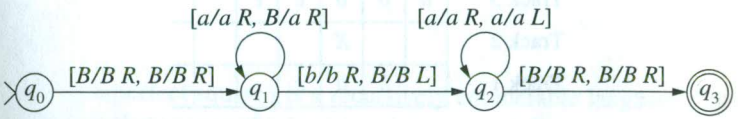
The repositioning consists of moving the tape head one square to the left or one square to the right or leaving it at its current position. A transition of a two-tape machine scanning x_1 on tape 1 and x_2 on tape 2 is written $\delta(q_i, x_1, x_2) = [q_j; y_1, d_1; y_2, d_2]$, where $x_i, y_i \in \Gamma$ and $d_i \in \{L, R, S\}$. This transition causes the machine to write y_i on tape i. The symbol d_i specifies the direction of the movement of tape head i: L signifies a move to the left, R a move to the right, and S means the head remains stationary. Any tape head attempting to move to the left of the boundary of its tape terminates the computation abnormally.

The input to a multitape machine is placed in the standard position on tape 1. All the other tapes are assumed to be blank. The tape heads originally scan the leftmost position of each tape. A multitape machine can be represented by a state diagram in which the label on an arc specifies the action for each tape. For example, the transition $\delta(q_i, x_1, x_2) = [q_j; y_1, d_1; y_2, d_2]$ will be represented by an arc from q_i to q_j labeled $[x_1/y_1\ d_1, x_2/y_2\ d_2]$.

Two advantages of multitape machines are the ability to copy data between tapes and to compare strings on different tapes. Both of these features will be demonstrated in the following example.

Example 8.6.1

The machine



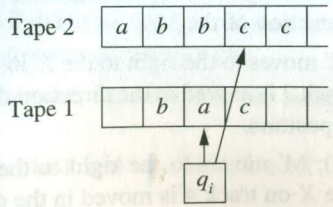
accepts the language $\{a^i b a^i \mid i \geq 0\}$. A computation with input string $a^i b a^i$ copies the leading a's to tape 2 in state q_1. When the b is read on tape 1, the computation enters state q_2 to compare the a's on tape 2 with the a's after the b on tape 1. If the same number of a's precede and follow the b, the computation halts in q_3 and accepts the input. The computation for strings without a b halt in q_1 and strings with more than one b in q_2. The computations for strings with with one b and an unequal number of leading and trailing a's also halt in q_2. Since every computation halts, M provides a decision procedure for membership in $\{a^i b a^i \mid i \geq 0\}$ and consequently the language is recursive. □

A standard Turing machine is a multitape Turing machine with a single tape. Consequently, every recursively enumerable language is accepted by a multitape machine. We will show that the computations of a two-tape machine can be simulated by computations of a five-track machine. The argument can be generalized to show that any language accepted by a k-tape machine is accepted by a $2k + 1$-track machine. The equivalence of acceptance by multitrack and standard machines then allows us to conclude the following.

Theorem 8.6.1

A language L is accepted by a multitape Turing machine if, and only if, it is accepted by a standard Turing machine.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a two-tape machine. During a computation, the tape heads of a multitape machine are independently positioned on the two tapes.



The single tape head of a multitrack machine reads all the tracks of a fixed position. The five-track machine M′ is constructed to simulate the computations of M. Tracks 1 and 3 maintain the information stored on tapes 1 and 2 of the two-tape machine. Tracks 2 and 4 have a single nonblank square indicating the position of the tape heads of the multitape machine.

Track 5	#						
Track 4				X			
Track 3	a	b	b	c	c		
Track 2			X				
Track 1		b	a	c			

The initial action of the simulation in the multitrack machine is to write # in the leftmost position of track 5 and X in the leftmost positions of tracks 2 and 4. The remainder of the computation of the multitrack machine consists of a sequence of actions that simulate the transitions of the two-tape machine.

A transition of the two-tape machine is determined by the two symbols being scanned and the machine state. The simulation in the five-track machine records the symbols marked by each of the X's. The states are 8-tuples of the form $[s, q_i, x_1, x_2, y_1, y_2, d_1, d_2]$, where $q_i \in Q$; $x_i, y_i \in \Sigma \cup \{U\}$; and $d_i \in \{L, R, S, U\}$. The element s represents the status of the simulation of the transition of M. The symbol U, added to the tape alphabet and the set of directions, indicates that this item is unknown.

Let $\delta(q_i, x_1, x_2) = [q_j; y_1, d_1; y_2, d_2]$ be the applicable two-tape transition of M. M′ begins the simulation of the transition in the state $[f1, q_i, U, U, U, U, U, U]$. The following five actions simulate the transition of M in the multitrack machine.

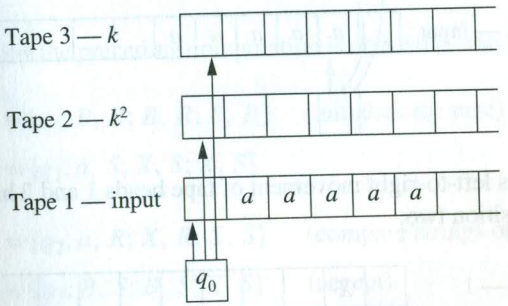
1. $f1$ (find first symbol): M′ moves to the right until it reads the X on track 2. State $[f1, q_i, x_1, U, U, U, U, U]$ is entered, where x_1 is the symbol in track 1 under the X. After recording the symbol on track 1 in the state, M′ returns to the initial position. The # on track 5 is used to reposition the tape head.
2. $f2$ (find second symbol): The same sequence of actions records the symbol beneath the X on track 4. M′ enters state $[f2, q_i, x_1, x_2, U, U, U, U]$, where x_2 is the symbol in track 3 under the X. The tape head is then returned to the initial position.
3. M′ enters the state $[p1, q_j, x_1, x_2, y_1, y_2, d_1, d_2]$, where the values q_j, y_1, y_2, d_1, and d_2 are obtained from the transition $\delta(q_i, x_1, x_2)$. This state contains the information needed to simulate the transition of the M.
4. $p1$ (print first symbol): M′ moves to the right to the X in track 2 and writes the symbol y_1 on track 1. The X on track 2 is moved in the direction designated by d_1. The machine then returns to the initial position.
5. $p2$ (print second symbol): M′ moves to the right to the X in track 4 and writes the symbol y_2 on track 3. The X on track 4 is moved in the direction designated by d_2.
6. The simulation of the transition $\delta(q_i, x_1, x_2) = [q_j; y_1, d_1; y_2, d_2]$ terminates by returning the tape head to the initial position to process the subsequent transition.

If $\delta(q_i, x_1, x_2)$ is undefined in the two-tape machine, the simulation halts after returning to the initial position following step 2. A state $[f2, q_i, x_1, y_1, U, U, U, U]$ is an accepting state of the multitrack machine M′ whenever q_i is an accepting state of M.

The next two examples illustrate the use of the additional tapes to store and manipulate data in a computation.

Example 8.6.2

The set $\{a^k \mid k \text{ is a perfect square}\}$ is a recursively enumerable language. The design of a three-tape machine that accepts this language is presented. Tape 1 contains the input string. The input is compared with a string of X's on tape 2 whose length is a perfect square. Tape 3 holds a string whose length is the square root of the string on tape 2. The initial configuration for a computation with input $aaaaa$ is



The values of k and k^2 are incremented until the length of the string on tape 2 is greater than or equal to the length of the input. A machine to perform these comparisons consists of the following actions.

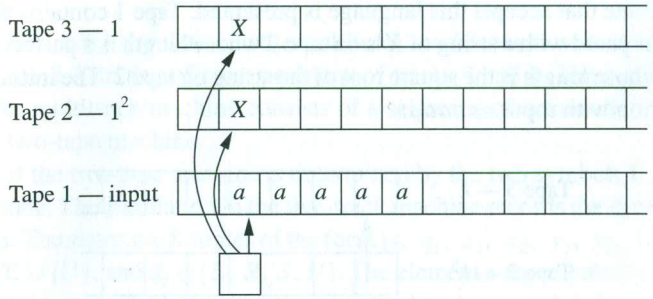
1. If the input is the null string, the computation halts in an accepting state. If not, tapes 2 and 3 are initialized by writing X in position one. The three tape heads are then moved to position one.
2. Tape 3 now contains a sequence of k X's and tape 2 contains k^2 X's. Simultaneously, the heads on tapes 1 and 2 move to the right while both heads scan nonblank squares. The head reading tape 3 remains at position one.
   a) If both heads simultaneously read a blank, the computation halts and the string is accepted.
   b) If tape head 1 reads a blank and tape head 2 an X, the computation halts and the string is rejected.
3. If neither of the halting conditions occur, the tapes are reconfigured for comparison with the next perfect square.
   a) An X is added to the right end of the string of X's on tape 2.
   b) Two copies of the string on tape 3 are added to the right end of the string on tape 2. This constructs a sequence of $(k + 1)^2$ X's on tape 2.

c) An X is added to the right end of the string of X's on tape 3. This constructs a sequence of $k + 1$ X's on tape 3.

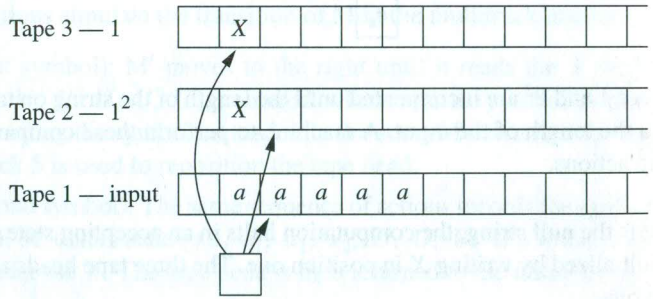
d) The tape heads are then repositioned at position one of their respective tapes.

4. The computation continues with step 2.

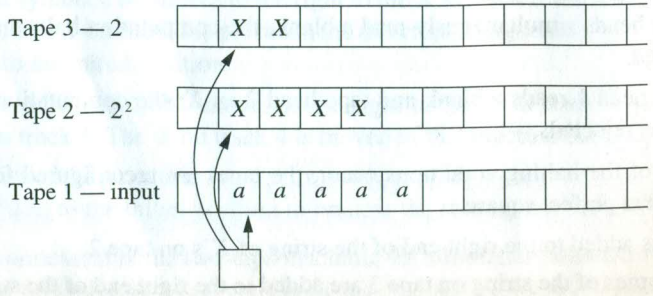
Tracing the computation for the input string $aaaaa$, step 1 produces the configuration



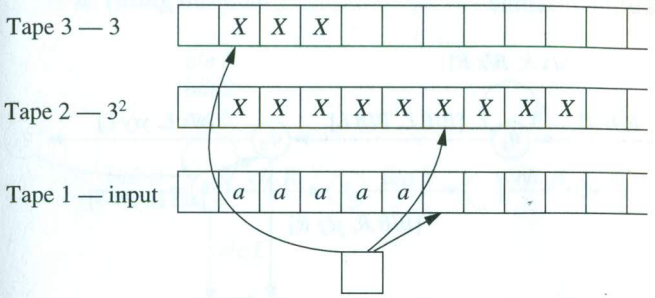
The simultaneous left-to-right movement of tape heads 1 and 2 halts when tape head 2 scans the blank in position two.



Part (c) of step 3 reformats tapes 2 and 3 so that the input string can be compared with the next perfect square.



Another iteration of step 2 halts and rejects the input.



A machine that performs the preceding computation is defined by the following transitions:

$$\delta(q_0, B, B, B) = [q_1; B, R; B, R; B, R] \quad \text{(initialize the tape)}$$
$$\delta(q_1, a, B, B) = [q_2; a, S; X, S; X, S]$$

$$\delta(q_2, a, X, X) = [q_2; a, R; X, R; X, S] \quad \text{(compare strings on tapes 1 and 2)}$$
$$\delta(q_2, B, B, X) = [q_3; B, S; B, S; X, S] \quad \text{(accept)}$$
$$\delta(q_2, a, B, X) = [q_4; a, S; X, R; X, S]$$

$$\delta(q_4, a, B, X) = [q_5; a, S; X, R; X, S] \quad \text{(rewrite tapes 2 and 3)}$$
$$\delta(q_4, a, B, B) = [q_6; a, L; B, L; X, L]$$
$$\delta(q_5, a, B, X) = [q_4; a, S; X, R; X, R]$$

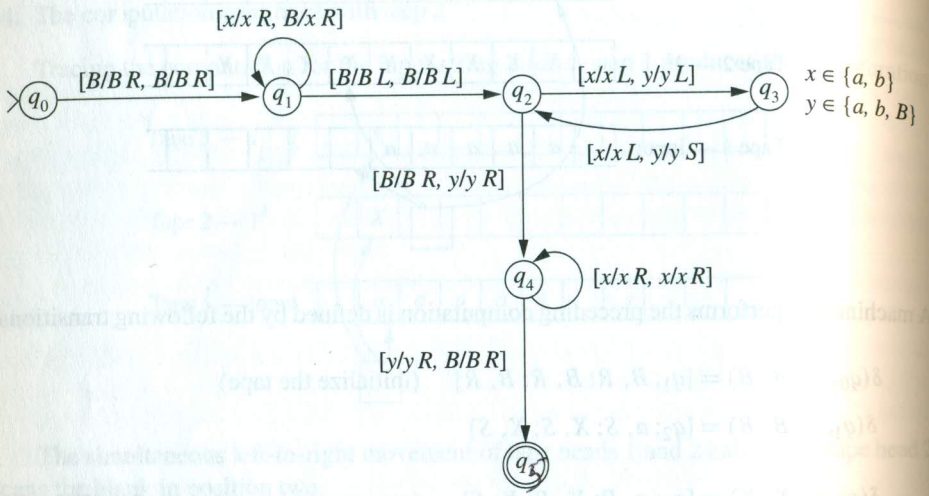
$$\delta(q_6, a, X, X) = [q_6; a, L; X, L; X, L] \quad \text{(reposition tape heads)}$$
$$\delta(q_6, a, X, B) = [q_6; a, L; X, L; B, S]$$
$$\delta(q_6, a, B, B) = [q_6; a, L; B, S; B, S]$$
$$\delta(q_6, B, X, B) = [q_6; B, S; X, L; B, S]$$
$$\delta(q_6, B, B, B) = [q_2; B, R; B, R; B, R]. \quad \text{(repeat comparison cycle)}$$

The accepting states are q_1 and q_3. The null string is accepted in q_1, and strings a^k, where k is a perfect square greater than zero, are accepted in q_3.

Since the machine designed above halts for all input strings, we have shown that the language $\{a^k \mid k \text{ is a perfect square}\}$ is not only recursively enumerable but also recursive. □

Example 8.6.3

The two-tape Turing machine



accepts the language $\{uu \mid u \in \{a, b\}^*\}$. The symbols x and y on the labels of the arcs represent an arbitrary input symbol.

The computation begins by making a copy of the input on tape 2. When this is complete, both tape heads are to the immediate right of the input. The tape heads now move back to the left, with tape head 1 moving two squares for every one square that tape head 2 moves. If the computation halts in q_3, the input string has odd length and is rejected. The loop in q_4 compares the first half of the input with the second; if they match, the string is accepted in state q_5. □

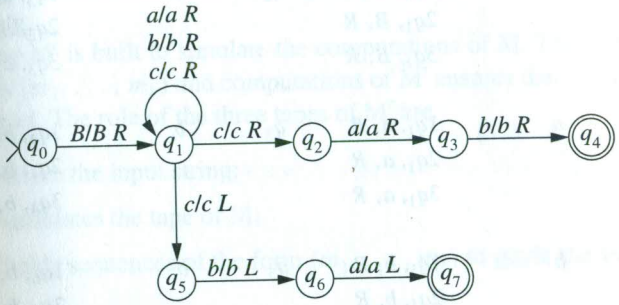
8.7 Nondeterministic Turing Machines

A nondeterministic Turing machine may specify any finite number of transitions for a given configuration. The components of a nondeterministic machine, with the exception of the transition function, are identical to those of the standard Turing machine. Transitions in a nondeterministic machine are defined by a function from $Q \times \Gamma$ to subsets of $Q \times \Gamma \times \{L, R\}$.

Whenever the transition function indicates that more than one action is possible, a computation arbitrarily chooses one of the transitions. An input string is accepted by a nondeterministic machine if there is at least one computation that terminates in an accepting state. The existence of other computations that halt in nonaccepting states or fail to halt altogether is irrelevant. As usual, the language of a machine is the set of strings accepted by the machine.

Example 8.7.1

The nondeterministic Turing machine



accepts strings containing a c preceded or followed by ab. The machine processes the input in state q_1 until a c is encountered. When this occurs, the computation may continue in state q_1, enter state q_2 to determine if the c is followed by ab, or enter q_5 to determine if the c is preceded by ab. In the language of nondeterminism, the computation chooses a c and then chooses one of the conditions to check. □

The machine constructed in Example 8.7.1 accepts strings by final state. As with standard machines, acceptance in nondeterministic Turing machines can be defined by final state or by halting alone. A nondeterministic machine accepts a string u by halting if there is at least one computation that halts normally when run with u. Exercise 24 establishes that these alternative approaches accept the same languages.

Nondeterminism does not increase the capabilities of Turing computation; the languages accepted by nondeterministic machines are precisely those accepted by deterministic machines. To accomplish the transformation of a nondeterministic Turing machine to an equivalent deterministic machine, we show that the multiple computations for a single input string can by sequentially generated and examined.

A nondeterministic Turing machine may produce multiple computations for a single input string. The computations can be systematically produced by ordering the alternative transitions for a state, symbol pair. Let n be the maximum number of transitions defined for any combination of state and tape symbol. The numbering assumes that $\delta(q_i, x)$ defines n, not necessarily distinct, transitions for every state q_i and tape symbol x with $\delta(q_i, x) \neq \emptyset$. If the transition function defines fewer than n transitions, one transition is assigned several numbers to complete the ordering.

A sequence $(m_1, \ldots, m_i, \ldots, m_k)$, where each m_i is a number from 1 to n, defines a unique computation in the nondeterministic machine. The computation associated with this sequence consists of k or fewer transitions. The jth transition is determined by the state, the tape symbol scanned, and m_j, the jth number in the sequence. Assume the $j-1$st transition leaves the machine in state q_i scanning x. If $\delta(q_i, x) = \emptyset$, the computation halts. Otherwise, the machine executes the transition in $\delta(q_i, x)$ numbered m_j.

TABLE 8.1 Ordering of Transitions

State	Symbol	Transition	State	Symbol	Transition
q_0	B	$1q_1, B, R$	q_2	a	$1q_3, a, R$
		$2q_1, B, R$			$2q_3, a, R$
		$3q_1, B, R$			$3q_3, a, R$
q_1	a	$1q_1, a, R$	q_3	b	$1q_4, b, R$
		$2q_1, a, R$			$2q_4, b, R$
		$3q_1, a, R$			$3q_4, b, R$
q_1	b	$1q_1, b, R$	q_5	b	$1q_6, b, L$
		$2q_1, b, R$			$2q_6, b, L$
		$3q_1, b, R$			$3q_6, b, L$
q_1	c	$1q_1, c, R$	q_6	a	$1q_7, a, L$
		$2q_2, c, R$			$2q_7, a, L$
		$3q_5, c, L$			$3q_7, a, L$

The transitions of the nondeterministic machine in Example 8.7.1 can be ordered as shown in Table 8.7.1. The computations defined by the input string $acab$ and the sequences (1, 1, 1, 1, 1), (1, 1, 2, 1, 1), and (2, 2, 3, 3, 1) are

$$
\begin{array}{lll}
q_0BacabB\ 1 & q_0BacabB\ 1 & q_0BacabB\ 2 \\
\vdash Bq_1acabB\ 1 & \vdash Bq_1acabB\ 1 & \vdash Bq_1acabB\ 2 \\
\vdash Baq_1cabB\ 1 & \vdash Baq_1cabB\ 2 & \vdash Baq_1cabB\ 3 \\
\vdash Bacq_1abB\ 1 & \vdash Bacq_2abB\ 1 & \vdash Bq_5acabB. \\
\vdash Bacaq_1bB\ 1 & \vdash Bacaq_3bB\ 1 & \\
\vdash Bacabq_1B & \vdash Bacabq_4B &
\end{array}
$$

The number on the right designates the transition used to obtain the subsequent configuration. The third computation terminates prematurely since no transition is defined when the machine is in state q_5 scanning an a. The string $acab$ is accepted since the computation defined by (1, 1, 2, 1, 1) terminates in state q_4.

Using the ability to sequentially produce the computations of a nondeterministic machine, we will now show that every nondeterministic Turing machine can be transformed into an equivalent deterministic machine. Let $M = (Q, \Sigma, \Gamma, \delta, q_0)$ be a nondeterministic machine that accepts strings by halting. We choose acceptance by halting because this reduces the number of potential outcomes of a computation from three to two—a

computation halts (and accepts) or does not halt. Thus we have fewer cases to consider in the proof. Assume that the transitions of M have been numbered according to the previous scheme, with n the maximum number of transitions for a state, symbol pair. A deterministic three-tape machine M′ is constructed to accept the language of M. Acceptance in M′ is also defined by halting.

The machine M′ is built to simulate the computations of M. The correspondence between sequences $(m_1, \ldots, m_k)$ and computations of M′ ensures that all possible computations are examined. The role of the three tapes of M′ are

Tape 1: stores the input string;

Tape 2: simulates the tape of M;

Tape 3: holds sequences of the form $(m_1, \ldots, m_k)$ to guide the simulation.

A computation in M′ consists of the actions:

1. A sequence of integers $(m_1, \ldots, m_k)$ from 1 to n is written on tape 3.
2. The input string on tape 1 is copied to the standard input position on tape 2.
3. The computation of M defined by the sequence on tape 3 is simulated on tape 2.
4. If the simulation halts prior to executing k transitions, the computation of M′ halts and accepts the input.
5. If the computation did not halt in step 3, the next sequence is generated on tape 3 and the computation continues at step 2.

The simulation is guided by the sequence of values on tape 3. The deterministic Turing machine in Figure 8.1 generates all finite-length sequences of integers from 1 to n, where the symbols $1, 2, \ldots, n$ are individual tape symbols. Sequences of length 1 are generated in numeric order, followed by sequences of length 2, length 3, and so on. A computation begins in state q_0 at position zero. When the tape head returns to position zero the tape contains the next sequence of values. The notation i/i abbreviates $1/1, 2/2, \ldots, n/n$.

Using the exhaustive generation of numeric sequences, we now construct a deterministic three-tape machine M′ that accepts L(M). A computation of the machine M′ interweaves the generation of the sequences on tape 3 with the simulation of M on tape 2. M′ halts when the sequence on tape 3 defines a computation that halts in M. Recall that both M and M′ accept by halting.

Let Σ and Γ be the input and tape alphabets of M. The alphabets of M′ are

$$\Sigma_{M'} = \Sigma$$
$$\Gamma_{M'} = \{x, \#x \mid x \in \Gamma\} \cup \{1, \ldots, n\}.$$

Symbols of the form $\#x$ represent tape symbol x and are used to mark the leftmost square on tape 2 during the simulation of the computation of M. The transitions of M′ are naturally grouped by their function. States labeled $q_{s,j}$ are used in the generation of a sequence on tape

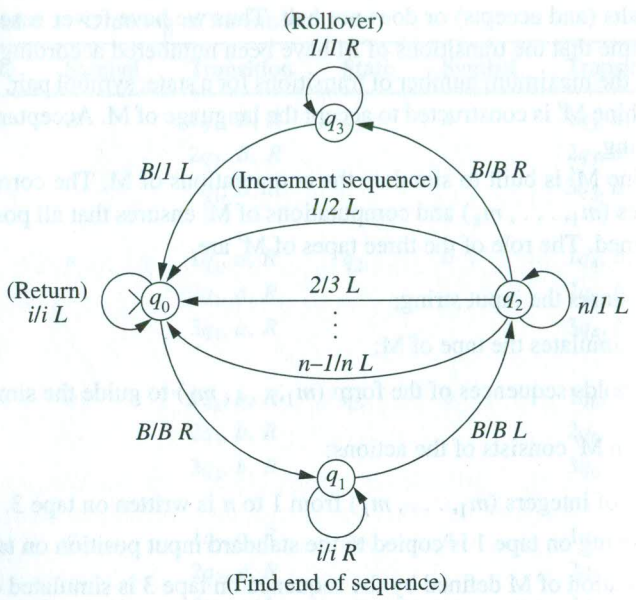


FIGURE 8.1 Turing machine generating $\{1, 2, \ldots, n\}^+$.

3. These transitions are obtained from the machine in Figure 8.1. The tape heads reading tapes 1 and 2 remain stationary during this operation.

$$\delta(q_{s,0}, B, B, B) = [q_{s,1}; B, S; B, S; B, R]$$
$$\delta(q_{s,1}, B, B, t) = [q_{s,1}; B, S; B, S; i, R] \qquad t = 1, \ldots, n$$
$$\delta(q_{s,1}, B, B, B) = [q_{s,2}; B, S; B, S; B, L]$$
$$\delta(q_{s,2}, B, B, n) = [q_{s,2}; B, S; B, S; 1, L]$$
$$\delta(q_{s,2}, B, B, t-1) = [q_{s,4}; B, S; B, S; t, L] \qquad t = 1, \ldots, n-1$$
$$\delta(q_{s,2}, B, B, B) = [q_{s,3}; B, S; B, S; B, R]$$
$$\delta(q_{s,3}, B, B, 1) = [q_{s,3}; B, S; B, S; 1, R]$$
$$\delta(q_{s,3}, B, B, B) = [q_{s,4}; B, S; B, S; 1, L]$$
$$\delta(q_{s,4}, B, B, t) = [q_{s,4}; B, S; B, S; t, L] \qquad t = 1, \ldots, n$$
$$\delta(q_{s,4}, B, B, B) = [q_{c,0}; B, S; B, S; B, S]$$

The next step is to make a copy of the input on tape 2. The symbol $\#B$ is written in position zero to designate the left boundary of the tape.

$$\delta(q_{c,0}, B, B, B) = [q_{c,1}; B, R; \#B, R; B, S]$$
$$\delta(q_{c,1}, x, B, B) = [q_{c,1}; x, R; x, R; B, S] \qquad \text{for all } x \in \Gamma - \{B\}$$
$$\delta(q_{c,1}, B, B, B) = [q_{c,2}; B, L; B, L; B, S]$$
$$\delta(q_{c,2}, x, x, B) = [q_{c,2}; x, L; x, L; B, S] \qquad \text{for all } x \in \Gamma$$
$$\delta(q_{c,2}, B, \#B, B) = [q_0; B, S; \#B, S; B, R]$$

The transitions that simulate the computation of M on tape 2 of M′ are obtained directly from the transitions of M. If $\delta(q_i, x) = [q_j, y, d]$ is a transition of M assigned the number t in the ordering, then

$$\delta(q_i, B, x, t) = [q_j; B, S; y, d; t, R]$$
$$\delta(q_i, B, \#x, t) = [q_j; B, S; \#y, d; t, R]$$

are the corresponding transitions of M′.

If the sequence on tape 3 consists of k numbers, the simulation processes at most k transitions. The computation of M′ halts if the computation of M specified by the sequence on tape 3 halts. When a blank is read on tape 3, the simulation has processed all of the transitions designated by the current sequence. Before the next sequence is processed, the result of the simulated computation must be erased from tape 2. To accomplish this, the tape heads on tapes 2 and 3 are repositioned at the leftmost position in state $q_{e,0}$ and $q_{e,1}$, respectively. The head on tape 2 then moves to the right, erasing the tape.

$$\delta(q_i, B, x, B) = [q_{e,0}; B, S; x, S; B, S] \qquad \text{for all } x \in \Gamma$$
$$\delta(q_i, B, \#x, B) = [q_{e,0}; B, S; \#x, S; B, S] \qquad \text{for all } x \in \Gamma$$
$$\delta(q_{e,0}, B, x, B) = [q_{e,0}; B, S; x, L; B, S] \qquad \text{for all } x \in \Gamma$$
$$\delta(q_{e,0}, B, \#x, B) = [q_{e,1}; B, S; B, S; B, L] \qquad \text{for all} x \in \Gamma$$
$$\delta(q_{e,1}, B, B, t) = [q_{e,1}; B, S; B, S; t, L] \qquad t = 1, \ldots, n$$
$$\delta(q_{e,1}, B, B, B) = [q_{e,2}; B, S; B, R; B, R]$$
$$\delta(q_{e,2}, B, x, i) = [q_{e,2}; B, S; B, R; i, R] \qquad \text{for all } x \in \Gamma \text{ and } i = 1, \ldots, n$$
$$\delta(q_{e,2}, B, B, B) = [q_{e,3}; B, S; B, L; B, L]$$
$$\delta(q_{e,3}, B, B, t) = [q_{e,3}; B, S; B, L; t, L] \qquad t = 1, \ldots, n$$
$$\delta(q_{e,3}, B, B, B) = [q_{s,0}; B, S; B, S; B, S]$$

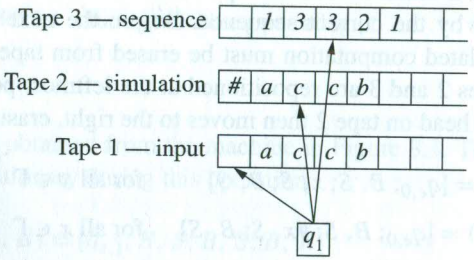
When a blank is read on tape 3, the entire segment of the tape that may have been accessed during the simulated computation has been erased. M′ then returns the tape heads to their initial position and enters $q_{s,0}$ to generate the next sequence and continue the simulation of computations.

The process of simulating computations of M, steps 2 through 5 of the algorithm, continues until a sequence of numbers is generated on tape 3 that defines a halting computation. The simulation of this computation causes M′ to halt, accepting the input. If the input string is not in L(M), the cycle of sequence generation and computation simulation in M′ will continue indefinitely.

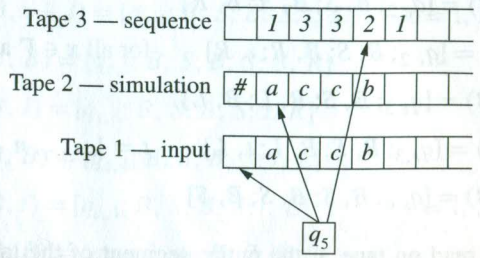
The actions of the deterministic machine constructed following the preceding strategy are illustrated using the nondeterministic machine from Example 8.7.1 and the numbering of the transitions in Table 8.7.1. The first three transitions of the computation M defined by the sequence (1, 3, 3, 2, 1) and input string *accb* are

$$\begin{aligned} & q_0BaccbB \quad 1 \\ \vdash\ & Bq_1accbB \quad 3 \\ \vdash\ & Baq_1ccbB \quad 3 \\ \vdash\ & Bq_5accbB. \end{aligned}$$

The sequence *1, 3, 3, 2, 1* that designates the particular computation of M is written on tape 3 of M′. The configuration of the three-tape machine M′ prior to the execution of the third transition of M is



Transition 3 from state q_1 with M scanning a c causes the machine to print c, enter state q_5, and move to the left. This transition is simulated in M′ by the transition $\delta'(q_1, B, c, 3) = [q_5; B, S; c, L; 3, R]$. The transition of M′ alters tape 2 as prescribed by the transition of M and moves the head on tape 3 to designate the number of the subsequent transition.

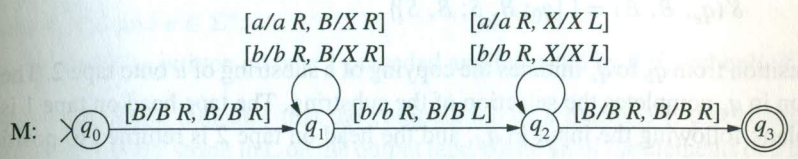


Nondeterministic Turing machines can be defined with a multitrack tape, two-way tape, or multiple tapes. Machines defined using these alternative configurations can also be shown to accept precisely the recursively enumerable languages.

Like their deterministic counterparts, nondeterministic machines that accept by final state can be used to show that a language is recursive. If every computation in the nondeterministic machine halts, so will every computation in the equivalent deterministic machine (Exerecise 23).

Example 8.7.2

The two-tape nondeterministic machine



accepts the set of strings over $\{a, b\}$ with a b in the middle. The transition from state q_1 to q_2 on reading a b on tape 1 represents a guess that the b is in the middle of the input. The loop in state q_2 compares the number of symbols following the b to the number preceding it. If a string is in L(M), one computation will enter q_3 upon reading the middle b and accept the input. The computations for strings with no b's halt in q_1, and strings that do not have a b in the middle halt in either q_1 or q_2. Since M halts for all inputs, L(M) is recursive. □

The next example illustrates the flexibility afforded by the combination of multitape machines and the guess and check strategy of nondeterminism.

Example 8.7.3

Let M = (Q, Σ, Γ, δ, q_0, F) be a standard Turing machine that accepts a language L. We will design a two-tape nondeterministic machine M′ that accepts strings over Σ* that have a substring of length two or more in L. That is, $L(M') = \{u \mid u = xyz,\ length(y) \geq 2 \text{ and } y \in L\}$. A computation of M′ with input u consists of the following steps:

1. Reading the input on tape 1 and nondeterministically choosing a position in the string to begin copying to tape 2;
2. Copying from tape 1 to tape 2 and nondeterministically choosing a position to stop copying;
3. Simulating the computation of M on tape 2.

The first two steps constitute the nondeterministic guess of a substring of u and the third checks whether the substring is in L.

The states of M′ are Q ∪ $\{q_s, q_b, q_c, q_d, q_e\}$ with start state q_s. The alphabets and final states are the same as those of M. The transitions for steps 1 and 2 use states q_s, q_b, q_c, q_d, and q_e.

$$\delta'(q_s, B, B) = \{[q_b; B, R; B, R]\}$$
$$\delta'(q_b, x, B) = \{[q_b; x, R; B, S], [q_c; x, R; x, R]\} \quad \text{for all } x \in \Sigma$$
$$\delta'(q_c, x, B) = \{[q_c; x, R; x, R], [q_d; x, R; x, R]\} \quad \text{for all } x \in \Sigma$$
$$\delta'(q_d, x, B) = \{[q_d; x, R; B, S]\} \quad \text{for all } x \in \Sigma$$
$$\delta'(q_d, B, B) = \{[q_e; B, S; B, L]\}$$
$$\delta'(q_e, B, x) = \{[q_e; B, S; x, L]\} \quad \text{for all } x \in \Sigma$$
$$\delta'(q_e, B, B) = \{[q_0; B, S; B, S]\}$$

The transition from q_b to q_c initiates the copying of a substring of u onto tape 2. The second transition in q_c completes the selection of the substring. The tape head on tape 1 is moved to the blank following the input in q_d, and the head on tape 2 is returned to position zero in q_e.

After the nondeterministic selection of a substring, the transitions of M are run on tape 2 to check whether the "guessed" substring is in L. The transitions for this part of the computation are obtained directly from δ, the transition function of M:

$$\delta'(q_i, B, x) = \{[q_j; B, S; y, d]\} \quad \text{whenever } \delta(q_i, x) = [q_j, y, d] \text{ is a transition of M.}$$

The tape head reading tape 1 remains stationary while the computation of M is run on tape 2. □

8.8 Turing Machines as Language Enumerators

In the preceding sections Turing machines have been formulated as language acceptors: A machine is provided with an input string, and the result of the computation indicates the acceptability of the input. Turing machines may also be designed to enumerate a language. The computation of such a machine sequentially produces an exhaustive listing of the elements of the language. An enumerating machine has no input; its computation continues until it has generated every string in the language.

Like Turing machines that accept languages, there are a number of equivalent ways to define an enumerating machine. We will use a k-tape deterministic machine, $k \geq 2$, as the underlying Turing machine model in the definition of enumerating machines. The first tape is the output tape and the remaining tapes are work tapes. A special tape symbol # is used on the output tape to separate the elements of the language that are generated during the computation.

The machines considered in this section perform two distinct tasks, acceptance and enumeration. To distinguish them, a machine that accepts a language will be denoted M while an enumerating machine will be denoted E.

Definition 8.8.1

A k-tape Turing machine $E = (Q, \Sigma, \Gamma, \delta, q_0)$ **enumerates** the language L if

i) the computation begins with all tapes blank;

ii) with each transition, the tape head on tape 1 (the output tape) remains stationary or moves to the right;

iii) at any point in the computation, the nonblank portion of tape 1 has the form

$$B\#u_1\#u_2\# \ldots \#u_k\# \quad \text{or} \quad B\#u_1\#u_2\# \ldots \#u_k\#v,$$

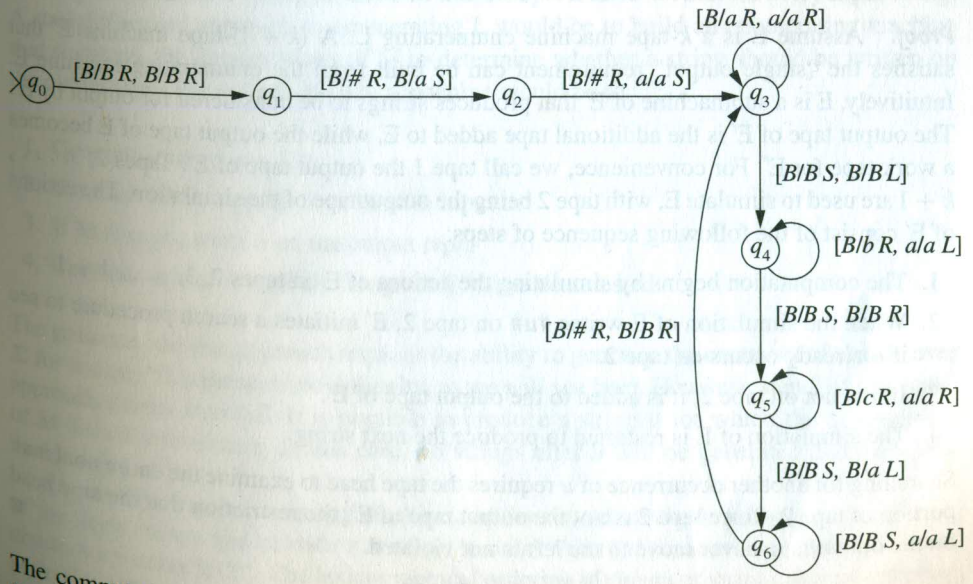
where $u_i \in L$ and $v \in \Sigma^*$;

iv) a string u will be written on tape 1 preceded and followed by # if, and only if, $u \in L$.

The last condition indicates that the computation of a machine E that enumerates L eventually writes every string in L on the output tape. Since all of the elements of a language must be produced, a computation enumerating an infinite language will never halt. The definition does not require a machine to halt even if it is enumerating a finite language. Such a machine may continue indefinitely after writing the last element on the output tape.

Example 8.8.1

The machine E enumerates the language $L = \{a^i b^i c^i \mid i \geq 0\}$. A Turing machine accepting this language was given in Example 8.2.2.



The computation of E begins by writing ## on the output tape, indicating that $\lambda \in L$. Simultaneously, an a is written in position one of tape 2, with the head returning to tape

position zero. At this point, E enters the nonterminating loop described by the following actions.

1. The tape heads move to the right, writing an a on the output tape for every a on the work tape.
2. The head on the work tape then moves right to left through the a's and a b is written on the output tape for each a.
3. The tape heads move to the right, writing a c on the output tape for every a on the work tape.
4. An a is added to the end of the work tape and the head is moved to position one.
5. A # is written on the output tape.

After a string is completed on the output tape, the work tape contains the information required to construct the next string in the enumeration. □

The definition of enumeration requires that each string in the language appear on the output tape but permits a string to appear multiple times. Theorem 8.8.2 shows that any language that is enumerated by a Turing machine can be enumerated by one in which each string is written only once on the output tape.

Theorem 8.8.2

Let L be a language enumerated by a Turing machine E. Then there is a Turing machine E′ that enumerates L and each string in L appears only once on the output tape of E′.

Proof. Assume E is a k-tape machine enumerating L. A $(k+1)$-tape machine E′ that satisfies the "single output" requirement can be built from the enumerating machine E. Intuitively, E is a submachine of E′ that produces strings to be considered for output by E′. The output tape of E′ is the additional tape added to E, while the output tape of E becomes a work tape for E′. For convenience, we call tape 1 the output tape of E′. Tapes 2, 3, . . . , $k+1$ are used to simulate E, with tape 2 being the output tape of the simulation. The actions of E′ consist of the following sequence of steps:

1. The computation begins by simulating the actions of E on tapes 2, 3, . . . , $k+1$.
2. When the simulation of E writes #u# on tape 2, E′ initiates a search procedure to see if u already occurs on tape 2.
3. If u is not on tape 2, it is added to the output tape of E′.
4. The simulation of E is restarted to produce the next string.

Searching for another occurrence of u requires the tape head to examine the entire nonblank portion of tape 2. Since tape 2 is not the output tape of E′, the restriction that the tape head on the output tape never move to the left is not violated. ■

Theorem 8.8.2 justifies the selection of the term *enumerate* to describe this type of computation. The computation sequentially and exhaustively lists the strings in the language. The order in which the strings are produced defines a mapping from an initial sequence of the natural numbers onto L. Thus we can talk about the zeroth string in L, the first string in L, and so on. This ordering is machine-specific; another enumerating machine may produce a completely different ordering.

Turing machine computations now have two distinct ways of defining a language: by acceptance and by enumeration. We show that these two approaches produce the same languages.

Lemma 8.8.3

If L is enumerated by a Turing machine, then L is recursively enumerable.

Proof. Assume that L is enumerated by a k-tape Turing machine E. A $(k+1)$-tape machine M accepting L can be constructed from E. The additional tape of M is the input tape; the remaining k tapes allow M to simulate the computation of E. The computation of M begins with a string u on its input tape. Next M simulates the computation of E. When the simulation of E writes #, a string $w \in$ L has been generated. M then compares u with w and accepts u if $u = w$. Otherwise, the simulation of E is used to generate another string from L and the comparison cycle is repeated. If $u \in$ L, it will eventually be produced by E and consequently accepted by M. ■

The proof that any recursively enumerable language L can be enumerated is complicated by the fact that a Turing machine M that accepts L need not halt for every input string. A straightforward approach to enumerating L would be to build an enumerating machine that simulates the computations of M to determine whether a string should be written on the output tape. The actions of such a machine would be to

1. Generate a string $u \in \Sigma^*$.
2. Simulate the computation of M with input u.
3. If M accepts, write u on the output tape.
4. Continue at step 1 until all strings in Σ^* have been tested.

The generate-and-test approach requires the ability to generate the entire set of strings over Σ for testing. This presents no difficulty, as we will see later. However, step 2 of this naive approach causes it to fail. It is possible to produce a string u for which the computation of M does not terminate. In this case, no strings after u will be generated and tested for membership in L.

To construct an enumerating machine, we first introduce the lexicographical ordering of the input strings and provide a strategy to ensure that the enumerating machine E will check every string in Σ^*. The lexicographical ordering of the set of strings over a nonempty alphabet Σ defines a one-to-one correspondence between the natural numbers and the strings in Σ^*.

Definition 8.8.4

Let $\Sigma = \{a_1, \ldots, a_n\}$ be an alphabet. The lexicographical ordering lo of Σ^* is defined recursively as follows:

i) Basis: $lo(\lambda) = 0, lo(a_i) = i$ for $i = 1, 2, \ldots, n$.

ii) Recursive step: $lo(a_i u) = lo(u) + i \cdot n^{length(u)}$.

The values assigned by the function lo define a total ordering on the set Σ^*. Strings u and v are said to satisfy $u < v$, $u = v$, and $u > v$ if $lo(u) < lo(v)$, $lo(u) = lo(v)$, and $lo(u) > lo(v)$, respectively.

Example 8.8.2

Let $\Sigma = \{a, b, c\}$ and let a, b, and c be assigned the values 1, 2, and 3, respectively. The lexicographical ordering produces

$$\begin{array}{lllllll} lo(\lambda) = 0 & lo(a) = 1 & lo(aa) = 4 & lo(ba) = 7 & lo(ca) = 10 & lo(aaa) = 13 \\ & lo(b) = 2 & lo(ab) = 5 & lo(bb) = 8 & lo(cb) = 11 & lo(aab) = 14 \\ & lo(c) = 3 & lo(ac) = 6 & lo(bc) = 9 & lo(cc) = 12 & lo(aac) = 15. \end{array}$$ □

Lemma 8.8.5

For any alphabet Σ, there is a Turing machine E_{Σ^*} that enumerates Σ^* in lexicographical order.

The construction of a machine that enumerates the set of strings over the alphabet $\{0, 1\}$ is left as an exercise.

The lexicographical ordering and a dovetailing technique are used to show that a recursively enumerable language L can be enumerated by a Turing machine. Let M be a Turing machine that accepts L. Recall that M need not halt for all input strings. The lexicographical ordering produces a listing $u_0 = \lambda, u_1, u_2, u_3, \ldots$ of the strings of Σ^*. A two-dimensional table is constructed whose columns are labeled by the strings of Σ^* and rows by the natural numbers.

.	.	.	.	
.	.	.	.	
.	.	.	.	
3	$[\lambda, 3]$	$[u_1, 3]$	$[u_2, 3]$	. . .
2	$[\lambda, 2]$	$[u_1, 2]$	$[u_2, 2]$	. . .
1	$[\lambda, 1]$	$[u_1, 1]$	$[u_2, 1]$	. . .
0	$[\lambda, 0]$	$[u_1, 0]$	$[u_2, 0]$	. . .
	λ	u_1	u_2	. . .

The $[i, j]$ entry in this table is interpreted to mean "run machine M on input u_i for j steps." Using the technique presented in Example 1.4.2, the ordered pairs in the table can be enumerated in a "diagonal by diagonal" manner (Exercise 33).

The machine E built to enumerate L interleaves the enumeration of the ordered pairs with the computations of M. The computation of E is a loop that consists of the following steps:

1. Generate an ordered pair $[i, j]$.
2. Run a simulation of M with input u_i for j transitions or until the simulation halts.
3. If M accepts, write u_i on the output tape.
4. Continue with step 1.

If $u_i \in$ L, then the computation of M with input u_i halts and accepts after k transitions, for some number k. Thus u_i will be written to the output tape of E when the ordered pair $[i, k]$ is processed. The second element in an ordered pair $[i, j]$ ensures that the simulation M is terminated after j steps. Consequently, no nonterminating computations are allowed and each string in Σ^* is examined.

Combining the preceding argument with Lemma 8.8.3 yields

Theorem 8.8.6

A language is recursively enumerable if, and only if, it can be enumerated by a Turing machine.

A Turing machine that accepts a recursively enumerable language halts and accepts every string in the language but is not required to halt when an input is a string that is not in the language. A language L is recursive if it is accepted by a machine that halts for all input. Since every computation halts, such a machine provides a decision procedure for determining membership in L. The family of recursive languages can also be defined by enumerating Turing machines.

The definition of an enumerating Turing machine does not impose any restrictions on the order in which the strings of the language are generated. Requiring the strings to be generated in a predetermined computable order provides the additional information needed to obtain negative answers to the membership question. Intuitively, the strategy to determine whether a string u is in the language is to begin the enumerating machine and compare u with each string that is produced. Eventually either u is output, in which case it is accepted, or some string beyond u in the ordering is generated. Since the output strings are produced according to the ordering, u has been passed and is not in the language. Thus we are able to decide membership, and the language is recursive. Theorem 8.8.7 shows that recursive languages may be characterized as the family of languages whose elements can be enumerated in order.

Theorem 8.8.7

L is recursive if, and only if, L can be enumerated in lexicographical order.

Proof. We first show that every recursive language can be enumerated in lexicographical order. Let L be a recursive language over an alphabet Σ. Then it is accepted by some machine M that halts for all input strings. A machine E that enumerates L in lexicographical order can be constructed from M and the machine E_{Σ^*} that enumerates Σ^* in lexicographical order. The machine E is a hybrid, interleaving the computations of M and E_{Σ^*}. The computation of E consists of the following loop:

1. The machine E_{Σ^*} is run, producing a string $u \in \Sigma^*$.
2. M is run with input u.
3. If M accepts u, u is written on the output tape of E.
4. The generate-and-test loop continues with step 1.

Since M halts for all inputs, E cannot enter a nonterminating computation in step 2. Thus each string $u \in \Sigma^*$ will be generated and tested for membership in L.

Now we show that any language L that can be enumerated in lexicographical order is recursive. This proof is divided into two cases based on the cardinality of L.

Case 1: L is finite. Then L is recursive since every finite language is recursive.

Case 2: L is infinite. The argument is similar to that given in Theorem 8.8.2 except that the ordering is used to terminate the computation. As before, a $(k + 1)$-tape machine M accepting L can be constructed from a k-tape machine E that enumerates L in lexicographical order. The additional tape of M is the input tape; the remaining k tapes allow M to simulate the computations of E. The ordering of the strings produced by E provides the information needed to halt M when the input is not in the language. The computation of M begins with a string u on its input tape. Next M simulates the computation of E. When the simulation produces a string w, M compares u with w. If $u = w$, then M halts and accepts. If w is greater than u in the ordering, M halts rejecting the input. Finally, if w is less than u in the ordering, then the simulation of E is restarted to produce another element of L and the comparison cycle is repeated. ■

Exercises

1. Let M be the Turing machine defined by

δ	B	a	b	c
q_0	q_1, B, R			
q_1	q_2, B, L	q_1, a, R	q_1, c, R	q_1, c, R
q_2		q_2, c, L		q_2, b, L

a) Trace the computation for the input string $aabca$.

b) Trace the computation for the input string $bcbc$.

c) Give the state diagram of M.

d) Describe the result of a computation in M.

2. Let M be the Turing machine defined by

δ	B	a	b	c
q_0	q_1, B, R			
q_1	q_1, B, R	q_1, a, R	q_1, b, R	q_2, c, L
q_2		q_2, b, L	q_2, a, L	

a) Trace the computation for the input string $abcab$.

b) Trace the first six transitions of the computation for the input string $abab$.

c) Give the state diagram of M.

d) Describe the result of a computation in M.

3. Construct a Turing machine with input alphabet $\{a, b\}$ to perform each of the following operations. Note that the tape head is scanning position zero in state q_f whenever a computation terminates.

a) Move the input one space to the right. Input configuration q_0BuB, result q_fBBuB.

b) Concatenate a copy of the reversed input string to the input. Input configuration q_0BuB, result q_fBuu^RB.

* c) Insert a blank between each of the input symbols. For example, input configuration q_0BabaB, result $q_fBaBbBaB$.

d) Erase the b's from the input. For example, input configuration $q_0BbabaababB$, result $q_fBaaaaB$.

4. Construct a Turing machine with input alphabet $\{a, b, c\}$ that accepts strings in which the first c is preceded by the substring aaa. A string must contain a c to be accepted by the machine.

5. Construct a Turing machine with input alphabet $\{a, b\}$ to accept each of the following languages by final state.

a) $\{a^ib^j \mid i \geq 0,\ j \geq i\}$

b) $\{a^ib^ja^ib^j \mid i,\ j > 0\}$

c) Strings with the same number of a's and b's

d) $\{uu^R \mid u \in \{a, b\}^*\}$

e) $\{uu \mid u \in \{a, b\}^*\}$

6. Modify your solution to Exercise 5(a) to obtain a Turing machine that accepts the language $\{a^ib^j \mid i \geq 0,\ j \geq i\}$ by halting.

7. An alternative method of acceptance by final state can be defined as follows: A string u is accepted by a Turing machine M if the computation of M with input u enters

(but does not necessarily terminate in) a final state. With this definition, a string may be accepted even though the computation of the machine does not terminate. Prove that the languages accepted by this definition are precisely the recursively enumerable languages.

8. The transitions of a one-tape deterministic Turing machine may be defined by a partial function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R, S\}$, where S indicates that the tape head remains stationary. Prove that machines defined in this manner accept precisely the recursively enumerable languages.

9. An **atomic** Turing machine is one in which every transition consists of a change of state and one other action. The transition may write on the tape or move the tape head, but not both. Prove that the atomic Turing machines accept precisely the recursively enumerable languages.

* 10. A **context-sensitive** Turing machine is one in which the applicability of a transition is determined not only by the symbol scanned but also by the symbol in the tape square to the right of the tape head. A transition has the form

$$\delta(q_i, xy) = [q_j, z, d] \qquad x, y, z \in \Gamma;\ d \in \{L, R\}.$$

When the machine is in state q_i scanning an x, the transition may be applied only when the tape position to the immediate right of the tape head contains a y. In this case the x is replaced by z, the machine enters state q_j, and the tape head moves in direction d.

a) Let M be a standard Turing machine. Define a context-sensitive Turing machine M′ that accepts L(M). *Hint:* Define the transition function of M′ from that of M.

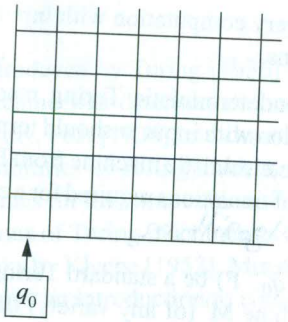
b) Let $\delta(q_i, xy) = [q_j, z, d]$ be a context-sensitive transition. Show that the result of the application of this transition can be obtained by a sequence of standard Turing machine transitions. You must consider the case both when transition $\delta(q_i, xy)$ is applicable and when it isn't.

c) Use parts (a) and (b) to conclude that context-sensitive machines accept precisely the recursively enumerable languages.

11. Prove that every recursively enumerable language is accepted by a Turing machine with a single accepting state.

12. Construct a Turing machine with two-way tape and input alphabet $\{a\}$ that halts if the tape contains a nonblank square. The symbol a may be anywhere on the tape, not necessarily to the immediate right of the tape head.

13. A **two-dimensional** Turing machine is one in which the tape consists of a two-dimensional array of tape squares.



A transition consists of rewriting a square and moving the head to any one of the four adjacent squares. A computation begins with the tape head reading the corner position. The transitions of the two-dimensional machine are written $\delta(q_i, x) = [q_j, y, d]$, where d is U (up), D (down), L (left), or R (right). Design a two-dimensional Turing machine with input alphabet $\{a\}$ that halts if the tape contains a nonblank square.

14. Let L be the set of palindromes over $\{a, b\}$.

a) Build a standard Turing machine that accepts L.

b) Build a two-tape machine that accepts L in which the computation with input u should take no more than $3\,length(u) + 4$ transitions.

15. Construct a two-tape Turing machine with input alphabet $\{a, b\}$ that accepts the language $\{a^i b^{2i} \mid i \geq 0\}$ in which the tape head on the input tape only moves from left to right.

16. Construct a two-tape Turing machine with input alphabet $\{a, b, c\}$ that accepts the language $\{a^i b^i c^i \mid i \geq 0\}$. diag, δ function

17. Construct a two-tape Turing machine with input alphabet $\{a, b\}$ that accepts strings with the same number of a's and b's. The computation with input u should take no more than $2\,length(u) + 3$ transitions.

18. Construct a two-tape Turing machine that accepts strings in which each a is followed by an increasing number of b's; that is, the strings are of the form

$$ab^{n_1}ab^{n_2}\ldots ab^{n_k}, k > 0,$$

where $n_1 < n_2 < \cdots < n_k$.

19. Construct a nondeterministic Turing machine whose language is the set of strings over $\{a, b\}$ that contain a substring u satisfying the following two properties:

i) $length(u) \geq 3$;

ii) u contains the same number of a's and b's.

20. Construct a two-tape nondeterministic Turing machine that accepts $L = \{uvuw \mid u \in \{a, b\}^5, v, w \in \{a, b\}^*\}$. A string is in L if it contains two nonoverlapping identical

substrings of length 5. Every computation with input w should terminate after at most $2\ length(w) + 2$ transitions.

21. Construct a two-tape nondeterministic Turing machine that accepts $L = \{uu \mid u \in \{a, b\}^*\}$. Every computation with input w should terminate after at most $2\ length(w) + 2$ transitions. Using the deterministic machine from Example 8.6.2 that accepts L, what is the maximum number of transitions required for a computation with an input of length n?

22. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a standard Turing machine that accepts a language L. Design a Turing machine M' (of any variety) that accepts a string $w \in \Sigma^*$ if, and only if, there is a substring of w in L.

23. Let L be a language accepted by a nondeterministic Turing machine in which every computation terminates. Prove that L is recursive.

24. Prove the equivalent of Theorem 8.3.2 for nondeterministic Turing machines.

25. Prove that every finite language is recursive.

26. Prove that a language L is recursive if, and only if, L and $\overline{L}$ are recursively enumerable.

27. Prove that the recursive languages are closed under union, intersection, and complement.

28. A machine that generates all sequences made up of integers from 1 to n was given in Figure 8.1. Trace the first seven cycles of the machine for $n = 3$. A cycle consists of the tape head returning to the initial position in state q_0.

29. Build a Turing machine that enumerates the set of even length strings over $\{a\}$.

30. Build a Turing machine that enumerates the set $\{a^i b^j \mid 0 \leq i \leq j\}$.

31. Build a Turing machine that enumerates the set $\{a^{2^n} \mid n \geq 0\}$.

32. Build a Turing machine E_{Σ^*} that enumerates Σ^* where $\Sigma = \{0, 1\}$. *Note:* This machine may be thought of as enumerating all finite-length bit strings.

* 33. Build a machine that enumerates the ordered pairs $\mathbf{N} \times \mathbf{N}$. Represent a number n by a string of $n + 1$ 1's. The output for ordered pair $[i, j]$ should consist of the representation of the number i followed by a blank followed by the representation of j. The markers # should surround the entire ordered pair.

34. In Theorem 8.8.7, the proof that every recursive language can be enumerated in lexicographical order considered the cases of finite and infinite languages separately. The argument for an infinite language may not be sufficient for a finite language. Why?

35. Define the components of a two-track nondeterministic Turing machine. Prove that these machines accept precisely the recursively enumerable languages.

36. Prove that every context-free language is recursive. *Hint:* Construct a two-tape nondeterministic Turing machine that simulates the computation of a pushdown automaton.

Bibliographic Notes

The Turing machine was introduced by Turing [1936] as a model for algorithmic computation. Turing's original machine was deterministic, consisting of a two-way tape and a single tape head. Independently, Post [1936] introduced a family of abstract machines with the same computational capabilities as Turing machines.

The use of Turing machines for the computation of functions is presented in Chapter 9. The capabilities and limitations of Turing machines as language acceptors are examined in Chapters 10 and 11. The books by Kleene [1952], Minsky [1967], Brainerd and Landweber [1974], and Hennie [1977] give an introduction to computability and Turing machines.