CHAPTER 3

FINITE AUTOMATA AND REGULAR GRAMMARS

3.1 THE FINITE AUTOMATON

In Chapter 2, we were introduced to a generating scheme—the grammar. Grammars are finite specifications for languages. In this chapter we shall see another method of finitely specifying infinite languages—the recognizer. We shall consider what is undoubtedly the simplest recognizer, called a finite automaton. The finite automaton (fa) cannot define all languages defined by grammars, but we shall show that the languages defined are exactly the type 3 languages. In later chapters, the reader will be introduced to recognizers for type 0, 1, and 2 languages. Here we shall define a finite automaton as a formal system, then give the physical meaning of the definition.

A *finite automaton* M over an alphabet Σ is a system [illegible] where K is a finite, nonempty set of states, Σ is a finite *input alphabet*, δ is a mapping of $K \times \Sigma$ into K, q_0 in K is the *initial state*, and $F \subseteq K$ is the set of *final states*.

Our model in Fig. 3.1 represents a finite control which reads symbols from a linear input tape in a sequential manner from left to right. The set of states K consists of the states of the finite control. Initially, the finite control is in state q_0 and is scanning the leftmost symbol of a string of symbols in Σ which appear on the input tape. The interpretation of $\delta(q, a) = p$, for q and p in K and a in Σ, is that M, in state q and scanning the input symbol a, moves its input head one cell to the right and goes to state p.

The mapping δ is from $K \times \Sigma$ to K. We can extend δ to domain† $K \times \Sigma^*$ by defining a mapping $\hat{\delta}$ as follows:

$$\hat{\delta}(q, \epsilon) = q$$

$$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a) \qquad \text{for each } x \text{ in } \Sigma^* \text{ and } a \text{ in } \Sigma.$$

Thus the interpretation of $\hat{\delta}(q, x) = p$ is that M, starting in state q with the string x written on the input tape, will be in state p when the input head moves right from the portion of the input tape containing x. Since δ and $\hat{\delta}$

† The *domain* of a mapping is the set of valid arguments for the mapping. The set of values which the mapping could take is called the *range*.

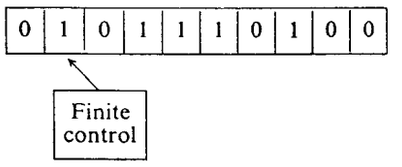


Fig. 3.1. A finite automaton.

agree wherever δ is defined, no confusion will arise if we fail to distinguish between δ and $\hat{\delta}$. Thus, for the remainder of the book, we shall use δ for both δ and $\hat{\delta}$.

A sentence x is said to be *accepted* by M if $\delta(q_0, x) = p$ for some p in F. The set of all x accepted by M is designated $T(M)$. That is,

$$T(M) = \{x | \delta(q, x) \text{ is in } F\}.$$

Any set of strings accepted by a finite automaton is said to be *regular*.

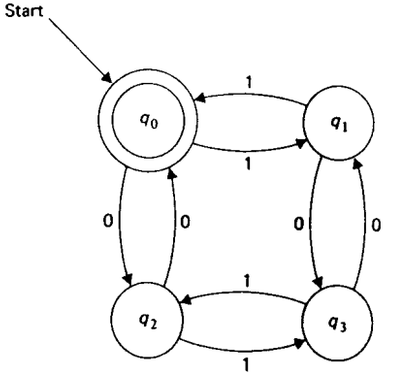
Example 3.1. The specifications for a finite automaton are given in Fig. 3.2(a). A *state diagram* for the automaton is shown in Fig. 3.2(b). The state diagram consists of a node for every state and a directed line from state q to state p with label a (in Σ) if the finite automaton, in state q, scanning the input symbol a, would go to state p. Final states, i.e., states in F, are indicated by a double circle. The initial state is marked by an arrow labeled start.

Consider the state diagram of Fig. 3.2(b). Suppose that 110101 is the input to M. Since $\delta(q_0, 1) = q_1$ and $\delta(q_1, 1) = q_0$, $\delta(q_0, 11) = q_0$. We might comment that thus, 11 is in $T(M)$, but we are interested in 110101. Now $\delta(q_0, 0) = q_2$, so $\delta(q_0, 110) = q_2$. Next $\delta(q_2, 1) = q_3$, so $\delta(q_0, 1101) = q_3$. Finally, $\delta(q_3, 0) = q_1$ and $\delta(q_1, 1) = q_0$, so $\delta(q_0, 110101) = q_0$, and thus 110101 is in $T(M)$. It is easily shown that $T(M)$ is the set of all sentences in $\{0, 1\}^*$ containing both an even number of 0's and an even number of 1's.

$M = (K, \Sigma, \delta, q_0, F)$
$\Sigma = \{0, 1\}$
$K = \{q_0, q_1, q_2, q_3\}$
$F = \{q_0\}$

$\delta(q_0, 0) = q_2$ $\quad \delta(q_0, 1) = q_1$
$\delta(q_1, 0) = q_3$ $\quad \delta(q_1, 1) = q_0$
$\delta(q_2, 0) = q_0$ $\quad \delta(q_2, 1) = q_3$
$\delta(q_3, 0) = q_1$ $\quad \delta(q_3, 1) = q_2$

(a)



(b)

Fig. 3.2. A finite automaton accepting the set of strings with an even number of 0's and an even number of 1's. (a) A finite automaton. (b) State diagram of the finite automaton.

3.2 EQUIVALENCE RELATIONS AND FINITE AUTOMATA

A *binary relation* R on a set S is a set of pairs of elements in S. If (a, b) is in R, then we are accustomed to seeing this fact written as aRb.

Example 3.2. For a familiar example, consider the relation "less than" usually denoted by the symbol $<$ on the set of integers. In the formal sense, this relation is the set: $\{(i, j) \mid i \text{ is less than } j\}$. Thus $3 < 4$, $2 < 17$, etc.

We are going to be concerned with some relations on sets of strings over a finite alphabet.

A binary relation R over a set S is said to be:

1. *reflexive* if for each s in S, sRs,
2. *symmetric* if for s and t in S, sRt implies tRs,
3. *transitive* if for s, t, and u in S, sRt and tRu imply sRu.

A relation which is reflexive, symmetric, and transitive is called an *equivalence relation*. An example of an equivalence relation over the set of positive integers is the relation E, given by: iEj if and only if $|i - j|$ is divisible by 3.

An important property of equivalence relations is that if R is an equivalence relation on the set S then we can divide S into k disjoint subsets, called *equivalence classes*, for some k between 1 and infinity, inclusive, such that aRb if and only if a and b are in the same subset.

The proof is simple. Define $[a]$ to be $\{b|aRb\}$. For any a and b in S, either $[a] = [b]$, or $[a]$ and $[b]$ are disjoint. Otherwise, let c be in $[a]$ and $[b]$, and d be in $[b]$ but not $[a]$. That is, aRc, bRc, and bRd, but not aRd. By symmetry, we have cRb. By transitivity, we can show cRd and aRd. The latter statement is a contradiction. The distinct sets that are $[a]$ for some a in S are the equivalence classes. Clearly, a and b are in the same set if and only if they are equivalent.

Example 3.3. The relation E given by iEj if and only if $|i - j|$ is divisible by 3 divides the set of positive integers into three classes $\{1, 4, 7, 10, \ldots\}$, $\{2, 5, 8, 11, \ldots\}$, and $\{3, 6, 9, 12, \ldots\}$. Any two elements from the same class are equivalent ($1E4$, $3E6$, etc.), and any two elements from different classes fail to satisfy the equivalence relation (not $7E9$, $1E5$, etc.).

The *index* of an equivalence relation is the number of equivalence classes generated. Thus the equivalence relation E has index 3.

Consider the finite automaton of Example 3.1. For x and y in $\{0, 1\}^*$, let (x, y) be in R if and only if $\delta(q_0, x) = \delta(q_0, y)$. The relation R is reflexive, symmetric, and transitive, since "$=$" has these properties, and thus, R is an equivalence relation. R divides the set $\{0, 1\}^*$ into four equivalence classes corresponding to the four states. In addition, if xRy, then $xz\ R\ yz$ for all z in $\{0, 1\}^*$, since

$$\delta(q_0, xz) = \delta(\delta(q_0, x), z) = \delta(\delta(q_0, y), z) = \delta(q_0, yz).$$

Such an equivalence relation is said to be *right invariant*. We see that every finite automaton induces a right invariant equivalence relation defined as R was defined, on its set of input strings. This result is formalized in the following theorem.

Theorem 3.1. The following three statements are equivalent:

1. The set $L \subseteq \Sigma^*$ is accepted by some finite automaton.
2. L is the union of some of the equivalence classes of a right invariant equivalence relation of finite index.
3. Let equivalence relation R be defined by: xRy if and only if for all z in Σ^*, xz is in L exactly when yz is in L. Then R is of finite index.

Proof. (1) $\Rightarrow$ (2). Assume that L is accepted by some fa $M = (K, \Sigma, \delta, q_0, F)$. Let R' be the equivalence relation $xR'y$ if and only if $\delta(q_0, x) = \delta(q_0, y)$. R' is right invariant since, for any z, if $\delta(q_0, x) = \delta(q_0, y)$, then

$$\delta(q_0, xz) = \delta(q_0, yz).$$

The index of R' is finite since the index is at most the number of states in K. Furthermore, L is the union of those equivalence classes which include an element x such that $\delta(q_0, x)$ is in F.

(2) $\Rightarrow$ (3). We show that any equivalence relation R' satisfying (2) is a refinement of R; that is, every equivalence class of R' is entirely contained in some equivalence class of R. Thus the index of R cannot be greater than the index of R' and so is finite. Assume that $xR'y$. Then since R' is right invariant, for each z in Σ^*, $xzR'yz$, and thus yz is in L if and only if xz is in L. Thus xRy, and hence, the equivalence class of x in R' is contained in the equivalence class of x in R. We conclude that each equivalence class of R' is contained within some equivalence class of R.

(3) $\Rightarrow$ (1). Assume that xRy. Then for each w and z in Σ^*, xwz is in L if and only if ywz is in L. Thus $xwRyw$, and R is right invariant. Now let K' be the finite set of equivalence classes of R and $[x]$ the element of K' containing x. Define $\delta'([x], a) = [xa]$. The definition is consistent, since R is right invariant. Let $q_0' = [\epsilon]$ and let $F' = \{[x] \mid x \in L\}$. The finite automaton $M' = (K', \Sigma, \delta', q_0', F')$ accepts L since $\delta'(q_0', x) = [x]$, and thus x is in $T(M')$ if and only if $[x]$ is in F'.

Theorem 3.2. The minimum state automaton accepting L is unique up to an isomorphism (i.e., a renaming of the states) and is given by M' of Theorem 3.1.

Proof. In the proof of Theorem 3.1 we saw that any fa $M = (K, \Sigma, \delta, q_0, F)$ accepting L defines an equivalence relation which is a refinement of R. Thus the number of states of M is greater than or equal to the number of states of M' of Theorem 3.1. If equality holds, then each of the states of M can be identified with one of the states of M'. That is, let q be a state of M. There

3.2 EQUIVALENCE RELATIONS AND FINITE AUTOMATA

A *binary relation* R on a set S is a set of pairs of elements in S. If (a, b) is in R, then we are accustomed to seeing this fact written as aRb.

Example 3.2. For a familiar example, consider the relation "less than" usually denoted by the symbol $<$ on the set of integers. In the formal sense, this relation is the set: $\{(i, j) \mid i \text{ is less than } j\}$. Thus $3 < 4$, $2 < 17$, etc.

We are going to be concerned with some relations on sets of strings over a finite alphabet.

A binary relation R over a set S is said to be:

1. *reflexive* if for each s in S, sRs,
2. *symmetric* if for s and t in S, sRt implies tRs,
3. *transitive* if for s, t, and u in S, sRt and tRu imply sRu.

A relation which is reflexive, symmetric, and transitive is called an *equivalence relation*. An example of an equivalence relation over the set of positive integers is the relation E, given by: iEj if and only if $|i - j|$ is divisible by 3.

An important property of equivalence relations is that if R is an equivalence relation on the set S then we can divide S into k disjoint subsets, called *equivalence classes*, for some k between 1 and infinity, inclusive, such that aRb if and only if a and b are in the same subset.

The proof is simple. Define $[a]$ to be $\{b|aRb\}$. For any a and b in S, either $[a] = [b]$, or $[a]$ and $[b]$ are disjoint. Otherwise, let c be in $[a]$ and $[b]$, and d be in $[b]$ but not $[a]$. That is, aRc, bRc, and bRd, but not aRd. By symmetry, we have cRb. By transitivity, we can show cRd and aRd. The latter statement is a contradiction. The distinct sets that are $[a]$ for some a in S are the equivalence classes. Clearly, a and b are in the same set if and only if they are equivalent.

Example 3.3. The relation E given by iEj if and only if $|i - j|$ is divisible by 3 divides the set of positive integers into three classes $\{1, 4, 7, 10, \ldots\}$, $\{2, 5, 8, 11, \ldots\}$, and $\{3, 6, 9, 12, \ldots\}$. Any two elements from the same class are equivalent ($1E4$, $3E6$, etc.), and any two elements from different classes fail to satisfy the equivalence relation (not $7E9$, $1E5$, etc.).

The *index* of an equivalence relation is the number of equivalence classes generated. Thus the equivalence relation E has index 3.

Consider the finite automaton of Example 3.1. For x and y in $\{0, 1\}^*$, let (x, y) be in R if and only if $\delta(q_0, x) = \delta(q_0, y)$. The relation R is reflexive, symmetric, and transitive, since "$=$" has these properties, and thus, R is an equivalence relation. R divides the set $\{0, 1\}^*$ into four equivalence classes corresponding to the four states. In addition, if xRy, then $xz\,R\,yz$ for all z in $\{0, 1\}^*$, since

$$\delta(q_0, xz) = \delta(\delta(q_0, x), z) = \delta(\delta(q_0, y), z) = \delta(q_0, yz).$$

Such an equivalence relation is said to be *right invariant*. We see that every finite automaton induces a right invariant equivalence relation defined as R was defined, on its set of input strings. This result is formalized in the following theorem.

Theorem 3.1. The following three statements are equivalent:

1. The set $L \subseteq \Sigma^*$ is accepted by some finite automaton.
2. L is the union of some of the equivalence classes of a right invariant equivalence relation of finite index.
3. Let equivalence relation R be defined by: xRy if and only if for all z in Σ^*, xz is in L exactly when yz is in L. Then R is of finite index.

Proof. (1) $\Rightarrow$ (2). Assume that L is accepted by some fa $M = (K, \Sigma, \delta, q_0, F)$. Let R' be the equivalence relation $xR'y$ if and only if $\delta(q_0, x) = \delta(q_0, y)$. R' is right invariant since, for any z, if $\delta(q_0, x) = \delta(q_0, y)$, then

$$\delta(q_0, xz) = \delta(q_0, yz).$$

The index of R' is finite since the index is at most the number of states in K. Furthermore, L is the union of those equivalence classes which include an element x such that $\delta(q_0, x)$ is in F.

(2) $\Rightarrow$ (3). We show that any equivalence relation R' satisfying (2) is a refinement of R; that is, every equivalence class of R' is entirely contained in some equivalence class of R. Thus the index of R cannot be greater than the index of R' and so is finite. Assume that $xR'y$. Then since R' is right invariant, for each z in Σ^*, $xzR'yz$, and thus yz is in L if and only if xz is in L. Thus xRy, and hence, the equivalence class of x in R' is contained in the equivalence class of x in R. We conclude that each equivalence class of R' is contained within some equivalence class of R.

(3) $\Rightarrow$ (1). Assume that xRy. Then for each w and z in Σ^*, xwz is in L if and only if ywz is in L. Thus $xwRyw$, and R is right invariant. Now let K' be the finite set of equivalence classes of R and $[x]$ the element of K' containing x. Define $\delta'([x], a) = [xa]$. The definition is consistent, since R is right invariant. Let $q_0' = [\epsilon]$ and let $F' = \{[x] \mid x \in L\}$. The finite automaton $M' = (K', \Sigma, \delta', q_0', F')$ accepts L since $\delta'(q_0', x) = [x]$, and thus x is in $T(M')$ if and only if $[x]$ is in F'.

Theorem 3.2. The minimum state automaton accepting L is unique up to an isomorphism (i.e., a renaming of the states) and is given by M' of Theorem 3.1.

Proof. In the proof of Theorem 3.1 we saw that any fa $M = (K, \Sigma, \delta, q_0, F)$ accepting L defines an equivalence relation which is a refinement of R. Thus the number of states of M is greater than or equal to the number of states of M' of Theorem 3.1. If equality holds, then each of the states of M can be identified with one of the states of M'. That is, let q be a state of M. There

must be some x in Σ^*, such that $\delta(q_0, x) = q$, otherwise q could be removed from K, and a smaller automaton found. Identify q with the state $\delta'(q_0', x)$, of M'. This identification will be consistent. If $\delta(q_0, x) = \delta(q_0, y) = q$, then, by Theorem 3.1, x and y are in the same equivalence class of R. Thus $\delta'(q_0', x) = \delta'(q_0', y)$.

3.3 NONDETERMINISTIC FINITE AUTOMATA

We now introduce the notion of a nondeterministic finite automaton. It will turn out that any set accepted by a nondeterministic finite automaton can also be accepted by a deterministic finite automaton.

However, the nondeterministic finite automaton is a useful concept in proving theorems. Also, the concept of a nondeterministic device is not an easy one to grasp. It is well to begin with a simple device. Later we deal with nondeterministic devices that are not equivalent to their deterministic counterparts. It is hoped that the study of nondeterministic finite automata will help in the understanding of those devices.

A *nondeterministic finite automaton* M is a system $(K, \Sigma, \delta, q_0, F)$, where K is a finite nonempty set of states, Σ the finite input alphabet, δ is a mapping of $K \times \Sigma$ into subsets of K, q_0 in K is the initial state, and $F \subseteq K$ is the set of final states.

The important difference between the deterministic and nondeterministic case is that $\delta(q, a)$ is a (possibly empty) set of states rather than a single state. The interpretation of $\delta(q, a) = \{p_1, p_2, \ldots, p_k\}$ is that M, in state q, scanning a on its input tape, moves its head one cell to the right and chooses any one of $p_1, p_2, \ldots, p_k$ as the next state.

The mapping δ can be extended to domain $K \times \Sigma^*$ by defining

$$\delta(q, \epsilon) = \{q\} \quad \text{and} \quad \delta(q, xa) = \bigcup_{p \text{ in } \delta(q,x)} \delta(p, a),$$

for each x in Σ^*, and a in Σ.

The mapping δ can be further extended to domain $2^K \times \Sigma^*$† by defining

$$\delta(\{p_1, p_2, \ldots, p_k\}, x) = \bigcup_{i=1}^{k} \delta(p_i, x).$$

A sentence x is *accepted* by M if there is a state p in both F and $\delta(q_0, x)$. The set of all x accepted by M is denoted $T(M)$.

Example 3.4. A nondeterministic fa which accepts the set of all sentences with either two consecutive 0's or two consecutive 1's is given in Fig. 3.3. The fa will make many choices upon reading an input string. Thus, suppose that 010110 is the input. After reading the first 0, M may stay in state q_0 or go to q_3. Next, with a 1 input, M can go nowhere from q_3 but from q_0 can

† 2^K, for any set K, denotes the *power set* or set of all subsets of K.

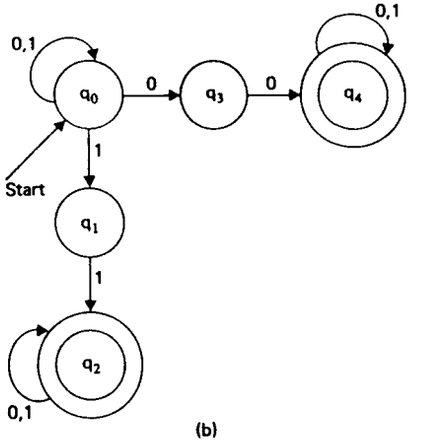
go to q_0 or q_1. Similarly, by the time the fourth input symbol is read, M can still be in only q_0 or q_1. When the fifth symbol, a 1, is read, M can go from q_1 to q_2 and from q_0 to q_0 or q_1. Thus M may be in state q_0, q_1, or q_2. Since there is a sequence of states leading to q_2, 01011 is accepted. Likewise, after the sixth symbol is read, M can be in state q_0, q_2, or q_3. Thus 010110 is also accepted.

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \delta, q_0, \{q_2, q_4\})$

$\delta(q_0, 0) = \{q_0, q_3\}$;
$\delta(q_1, 0) = \varphi$;
$\delta(q_2, 0) = \{q_2\}$;
$\delta(q_3, 0) = \{q_4\}$;
$\delta(q_4, 0) = \{q_4\}$;

$\delta(q_0, 1) = \{q_0, q_1\}$.
$\delta(q_1, 1) = \{q_2\}$.
$\delta(q_2, 1) = \{q_2\}$.
$\delta(q_3, 1) = \varphi$.
$\delta(q_4, 1) = \{q_4\}$.

(a)



(b)

Fig. 3.3. A nondeterministic finite automaton which accepts the set of all sentences containing either two consecutive 0's or two consecutive 1's. (a) Specification. (b) State diagram.

Theorem 3.3. Let L be a set accepted by a nondeterministic finite automaton. Then there exists a deterministic finite automaton that accepts L.

Proof. Let $M = (K, \Sigma, \delta, q_0, F)$ be a nondeterministic fa accepting L. Define a deterministic fa, $M' = (K', \Sigma, \delta', q_0', F')$ as follows. The states of M' are all the subsets of the set of states of M. That is, $K' = 2^K$. M' will keep track of all the states M could be in at any given time. F' is the set of all states in K' containing a state of F. An element of K' will be denoted by $[q_1, q_2, \ldots, q_i]$, where $q_1, q_2, \ldots, q_i$ are in K. Note that $q_0' = [q_0]$.

We define

$$\delta'([q_1, q_2, \ldots, q_i], a) = [p_1, p_2, \ldots, p_j]$$

if and only if

$$\delta(\{q_1, q_2, \ldots, q_i\}, a) = \{p_1, p_2, \ldots, p_j\}.$$

That is, δ' applied to an element Q of K' is computed by applying δ to each state of K represented by $Q = [q_1, q_2, \ldots, q_i]$. On applying δ to each of $q_1, q_2, \ldots, q_i$ and taking the union, we get some new set of states, $p_1, p_2, \ldots, p_j$. This new set of states has a representative, $[p_1, p_2, \ldots, p_j]$ in K', and that element is the value of $\delta'([q_1, q_2, \ldots, q_i], a)$.

It is easy to show by induction on the length of the input string x that

$$\delta'(q_0', x) = [q_1, q_2, \ldots, q_i]$$

if and only if

$$\delta(q_0, x) = \{q_1, q_2, \ldots, q_i\}.$$

The result is trivial for $|x| = 0$, since $q_0' = [q_0]$. Suppose that it is true for $|x| \leqq l$. Then, for a in Σ,

$$\delta'(q_0', xa) = \delta'(\delta'(q_0', x), a).$$

By the inductive hypothesis,

$$\delta'(q_0', x) = [p_1, p_2, \ldots, p_j]$$

if and only if

$$\delta(q_0, x) = \{p_1, p_2, \ldots, p_j\}.$$

But by definition,

$$\delta'([p_1, p_2, \ldots, p_j], a) = [r_1, r_2, \ldots, r_k]$$

if and only if

$$\delta(\{p_1, p_2, \ldots, p_j\}, a) = \{r_1, r_2, \ldots, r_k\}.$$

Thus,

$$\delta'(q_0', xa) = [r_1, r_2, \ldots, r_k]$$

if and only if

$$\delta(q_0, xa) = \{r_1, r_2, \ldots, r_k\}.$$

To complete the proof, we have only to add that $\delta'(q_0', x)$ is in F' exactly when $\delta(q_0, x)$ contains a state of K which is in F. Thus $T(M) = T(M')$.

Since the deterministic and nondeterministic finite automata accept the same sets, we shall not distinguish between them unless it becomes necessary, but shall simply refer to both as finite automata.

Example 3.5. Let $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ be a nondeterministic fa, where:

$$\delta(q_0, 0) = \{q_0, q_1\} \qquad \delta(q_0, 1) = \{q_1\} \qquad \delta(q_1, 0) = \varphi \qquad \delta(q_1, 1) = \{q_0, q_1\}.$$

We can construct a deterministic fa, $M' = (K, \{0, 1\}, \delta', [q_0], F)$, accepting $T(M)$ as follows. K consists of all subsets of $\{q_0, q_1\}$. We denote the elements of K by $[q_0]$, $[q_1]$, $[q_0, q_1]$ and φ. Since $\delta(q_0, 0) = \{q_0, q_1\}$,

$$\delta'([q_0], 0) = [q_0, q_1].$$

Likewise,

$$\delta'([q_0], 1) = [q_1], \ \delta'([q_1], 0) = \varphi \quad \text{and} \quad \delta'([q_1], 1) = [q_0, q_1].$$

Naturally, $\delta'(\varphi, 0) = \delta'(\varphi, 1) = \varphi$. Lastly,

$$\delta'([q_0, q_1], 0) = [q_0, q_1],$$

since

$$\delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \varphi = \{q_0, q_1\};$$

and

$$\delta'([q_0, q_1], 1) = [q_0, q_1],$$

since

$$\delta(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_1\} \cup \{q_0, q_1\} = \{q_0, q_1\}.$$

The set F of final states is $\{[q_1], [q_0, q_1]\}$.

3.4 FINITE AUTOMATA AND TYPE 3 LANGUAGES

We now turn to the relationship between the languages generated by type 3 grammars and the sets accepted by finite automata.

Theorem 3.4. Let $G = (V_N, V_T, P, S)$ be a type 3 grammar. Then there exists a finite automaton $M = (K, V_T, \delta, S, F)$ with $T(M) = L(G)$.

Proof. M will be a nondeterministic fa. The states of M are the variables of G, plus an additional state A, not in V_N. Thus, $K = V_N \cup \{A\}$. The initial state of M is S. If P contains the production $S \to \epsilon$, then $F = \{S, A\}$. Otherwise, $F = \{A\}$. Recall that S will not appear on the right of any production if $S \to \epsilon$ is in P. The state A is in $\delta(B, a)$ if $B \to a$ is in P. In addition, $\delta(B, a)$ contains all C such that $B \to aC$ is in P. $\delta(A, a) = \varphi$ for each a in V_T.

The fa M, when accepting a sentence x, simulates a derivation of x by the grammar G. We shall show that $T(M) = L(G)$. Let $x = a_1 a_2 \ldots a_n$ be in $L(G)$, $n \geqq 1$. Then

$$S \Rightarrow a_1 A_1 \Rightarrow \cdots \Rightarrow a_1 a_2 \ldots a_{n-1} A_{n-1} \Rightarrow a_1 a_2 \ldots a_{n-1} a_n$$

for some sequence of variables $A_1, A_2, \ldots, A_{n-1}$. From the definition of δ, we can see that $\delta(S, a_1)$ contains A_1, that $\delta(A_1, a_2)$ contains A_2, etc., and that $\delta(A_{n-1}, a_n)$ contains A. Thus x is in $T(M)$, since $\delta(S, x)$ contains A, and A is in F. If ϵ is in $L(G)$, then S is in F, so ϵ is in $T(M)$.

Likewise, if x is in $T(M)$, $|x| \geqq 1$, then there exists a sequence of states $S, A_1, A_2, \ldots, A_{n-1}, A$ such that $\delta(S, a_1)$ contains A_1, $\delta(A_1, a_2)$ contains A_2, and so forth. Thus, P contains rules $S \to a_1 A_1$, $A_1 \to a_2 A_2, \ldots$ and $A_{n-1} \to a_n$. Therefore, $S \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \cdots \Rightarrow a_1 a_2 \ldots a_{n-1} A_{n-1} \Rightarrow a_1 a_2 \ldots a_n$ is a derivation in G and x is in $L(G)$. If ϵ is in $T(M)$, then S is in F, so $S \to \epsilon$ is a production in P, and ϵ is in $L(G)$.

Theorem 3.5. Given a finite automaton M, there exists a type 3 grammar G, such that $L(G) = T(M)$.

Proof. Without loss of generality let $M = (K, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton. Define a type 3 grammar $G = (K, \Sigma, P, q_0)$ as follows.

1. $B \to aC$ is in P if $\delta(B, a) = C$.
2. $B \to a$ is in P if $\delta(B, a) = C$ and C is in F.

The proof that $q_0 \overset{*}{\underset{G}{\Rightarrow}} w$ if and only if $\delta(q_0, w)$ is in F, for $|w| \geqq 1$, is similar to the proof of Theorem 3.4, and will be left to the reader. If q_0 is in F, then ϵ is in $T(M)$. In that case, $L(G) = T(M) - \{\epsilon\}$. By Theorem 2.1, we can obtain from G, a new type 3 grammar G_1, where

$$L(G_1) = L(G) \cup \{\epsilon\} = T(M).$$

If q_0 is not in F, then ϵ is not in $T(M)$, so $L(G) = T(M)$.

Example 3.6. Consider the following regular grammar, $G = (\{S, B\}, \{0, 1\}, P, S)$, where P consists of: $S \to 0B$, $B \to 0B$, $B \to 1S$, $B \to 0$.

We can construct a nondeterministic finite automaton $M = (\{S, B, A\}, \{0, 1\}, \delta, S, \{A\})$, where δ is given by:

1. $\delta(S, 0) = \{B\}$, since $S \to 0B$ is the only production in P with S on the left and 0 on the right.
2. $\delta(S, 1) = \varphi$, since no production has S on the left and 1 on the right.
3. $\delta(B, 0) = \{B, A\}$, since $B \to 0B$ and $B \to 0$ are in P.
4. $\delta(B, 1) = \{S\}$, since $B \to 1S$ is in P.
5. $\delta(A, 0) = \delta(A, 1) = \varphi$.

By Theorem 3.4, $T(M) = L(G)$, as one can easily verify.

We now use the construction of Theorem 3.3 to find a deterministic finite automaton M_1 equivalent to M. Then, we use the construction of Theorem 3.5 to find a grammar G_1, generating $L(G)$.

Let $M_1 = (K, \{0, 1\}, \delta', [S], F)$.

$$K = \{\varphi, [S], [A], [B], [A, S], [A, B], [B, S], [A, B, S]\}.$$

$$F = \{[A], [A, S], [A, B], [A, B, S]\}.$$

$$\begin{aligned} \delta'([S], 0) &= [B] & \qquad \delta'([S], 1) &= \varphi \\ \delta'([B], 0) &= [A, B] & \qquad \delta'([B], 1) &= [S] \\ \delta'([A, B], 0) &= [A, B] & \qquad \delta'([A, B], 1) &= [S] \\ \delta'(\varphi, 0) &= \delta'(\varphi, 1) = \varphi \end{aligned}$$

There are other rules of δ'. However, no states other than φ, $[S]$, $[B]$, and $[A, B]$ will ever be entered by M_1, and the other states can be removed from K and F.

Now, let us construct grammar $G_1 = (K, \{0, 1\}, P_1, [S])$ from M_1. From $\delta'([S], 0) = [B]$ we get the production $[S] \to 0[B]$. From $\delta'([B], 0) = [A, B]$, we get $[B] \to 0[A, B]$ and, since $[A, B]$ is a final state of M_1, we place production $[B] \to 0$ in P_1, and so on. A complete list of the productions of P_1 is:

$$\begin{array}{lll} [S] \to 0[B] & [S] \to 1\varphi & \\ [B] \to 0[A, B] & [B] \to 1[S] & [B] \to 0 \\ [A, B] \to 0[A, B] & [A, B] \to 1[S] & [A, B] \to 0 \\ \varphi \to 0\varphi & \varphi \to 1\varphi & \end{array}$$

The grammar G_1 is much more complicated than is G, but $L(G_1) = L(G)$. The reader can simplify grammar G_1 so that its equivalence to G is readily observable.

3.5 PROPERTIES OF TYPE 3 LANGUAGES

Since the class of languages generated by type 3 grammars is equivalent to the class of sets accepted by finite automata, we shall use both formulations in establishing the properties of the class of type 3 languages. First we intend to show that the type 3 languages form a Boolean algebra† of sets.

Lemma 3.1. The class of type 3 languages is closed under union.

Proof. Two proofs are possible. One involves the use of nondeterministic finite automata. We leave this proof to the reader. A proof using grammars is also easy, and is given here.

Let L_1 and L_2 be type 3 languages generated by type 3 grammars

$$G_1 = (V_N^{(1)}, V_T^{(1)}, P_1, S_1) \quad \text{and} \quad G_2 = (V_N^{(2)}, V_T^{(2)}, P_2, S_2),$$

respectively. By renaming symbols, if necessary, we can assume that $V_N^{(1)}$ and $V_N^{(2)}$ contain no symbols in common, and that S is in neither. We construct a new grammar,

$$G_3 = (V_N^{(1)} \cup V_N^{(2)} \cup \{S\}, V_T^{(1)} \cup V_T^{(2)}, P_3, S),$$

where P_3 consists of the productions of P_1 and P_2 except for $S_1 \to \epsilon$ or $S_2 \to \epsilon$, plus all productions of the form $S \to \alpha$ such that either $S_1 \to \alpha$ is in P_1 or $S_2 \to \alpha$ is in P_2.

It should be obvious that $S \underset{G_3}{\Rightarrow} \alpha$ if and only if $S_1 \underset{G_1}{\Rightarrow} \alpha$ or $S_2 \underset{G_2}{\Rightarrow} \alpha$. In the first case, only strings in alphabet $V_N^{(1)} \cup V_T^{(1)}$ can be derived from α. In the second case, only strings in $V_N^{(2)} \cup V_T^{(2)}$ can be derived from α. Formally, if $S_1 \underset{G_1}{\Rightarrow} \alpha$, then $\alpha \overset{*}{\underset{G_3}{\Rightarrow}} w$ if and only if $\alpha \overset{*}{\underset{G_1}{\Rightarrow}} w$, and if $S_2 \underset{G_2}{\Rightarrow} \alpha$, then $\alpha \overset{*}{\underset{G_3}{\Rightarrow}} w$ if and only if $\alpha \overset{*}{\underset{G_2}{\Rightarrow}} w$. Putting the above together, $S \overset{*}{\underset{G_3}{\Rightarrow}} w$ if and only if either $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w$ or $S_2 \overset{*}{\underset{G_2}{\Rightarrow}} w$. That is, $L(G_3) = L(G_1) \cup L(G_2)$.

† For our purposes a Boolean algebra of sets is a collection of sets closed under union, complement, and intersection. By the complement $\bar{L}$ of a language L, we mean $\Sigma^* - L$, for a finite set of symbols Σ, such that $L \subseteq \Sigma^*$.

Lemma 3.2. The class of sets accepted by finite automata (generated by type 3 grammars) is closed under complement.

Proof. Let $M_1 = (K, \Sigma_1, \delta_1, q_0, F)$ be a deterministic fa accepting a set S_1. Let Σ_2 be a finite alphabet containing Σ_1 and let d be a new state not in K. We construct M_2 to accept $\Sigma_2^* - S_1$. Let

$$M_2 = (K \cup \{d\}, \Sigma_2, \delta_2, q_0, (K - F) \cup \{d\}),$$

where $\delta_2(q, a) = \delta_1(q, a)$ for each q in K and a in Σ_1, $\delta_2(q, a) = d$ for each q in K and a in $\Sigma_2 - \Sigma_1$, and $\delta_2(d, a) = d$ for each a in Σ_2. Intuitively, M_2 is obtained by extending the input alphabet of M_1 to Σ_2, adding the "trap" state d and then interchanging final and nonfinal states. Clearly, M_2 accepts $\Sigma_2^* - S_1$.

Theorem 3.6. The class of sets accepted by finite automata forms a Boolean algebra.

Proof. Immediate from Lemmas 3.1 and 3.2 and the fact that

$$L_1 \cap L_2 = \overline{\overline{L}_1 \cup \overline{L}_2}.$$

We now give some additional theorems which will culminate in the characterization of the type 3 languages.

Theorem 3.7. All finite sets are accepted by finite automata.

Proof. Consider the set containing only the sentence $x = a_1 a_2 \ldots a_n$. We can design a finite automaton M with $n + 2$ states $q_0, q_1, q_2, \ldots, q_n$, and p. The initial state is q_0, and q_n is the only final state. As M sees successive symbols of x, it moves to successively higher-numbered states. If M sees a symbol which is not the next symbol of x, M goes to state p which is a "trap state" with no exit. Formally,

$$\delta(q_{i-1}, a_i) = q_i, \qquad 1 \leqq i \leqq n,$$

$$\delta(q_{i-1}, a) = p, \qquad 1 \leqq i \leqq n, \qquad \text{if } a \neq a_i$$

and

$$\delta(q_n, a) = \delta(p, a) = p \qquad \text{for all } a.$$

The reader should be able to supply the steps necessary to show that M accepts the sentence x. The set containing only the empty sentence is accepted by $M = (\{q_0, p\}, \Sigma, \delta, q_0, \{q_0\})$ where $\delta(q_0, a) = \delta(p, a) = p$ for each a in Σ. The empty set is accepted by $M = (\{q_0\}, \Sigma, \delta, q_0, \varphi)$ where $\delta(q_0, a) = q_0$ for each a in Σ.

The theorem follows immediately from the closure of type 3 languages under union.

We now define the *product*† UV of two languages U and V by

$$UV = \{x | x = uv, u \text{ is in } U \text{ and } v \text{ is in } V\}.$$

That is, each string in the set UV is formed by concatenating a string in U with a string in V. As an example, if $U = \{01, 11\}$ and $V = \{1, 0, 101\}$, then the set UV is $\{011, 010, 01101, 111, 110, 11101\}$.

Theorem 3.8. The class of sets accepted by finite automata (generated by type 3 grammars) is closed under product.

Proof. Let $M_1 = (K_1, \Sigma_1, \delta_1, q_1, F_1)$ and $M_2 = (K_2, \Sigma_2, \delta_2, q_2, F_2)$ be deterministic finite automata accepting languages L_1 and L_2, respectively. Assume that K_1 and K_2 are disjoint. Furthermore, without loss of generality, we can assume that $\Sigma_1 = \Sigma_2 = \Sigma$. (Otherwise, we can add "dead" states to K_1 and K_2 as in the proof of Lemma 3.2.) We construct a nondeterministic finite automaton M_3, accepting $L_1 L_2$, which operates as follows. If the input string is x, M_3 behaves as M_1 until some initial portion (possibly ϵ) of x has been scanned. At this point, if M_1 would accept, M_3 guesses whether the end of the string from L_1 has been reached, or whether a longer initial portion is the string from L_1. In the former case, M_3 acts subsequently as M_2, and in the latter case, M_3 continues to behave as M_1.

Formally, let $M_3 = (K_1 \cup K_2, \Sigma, \delta_3, q_1, F)$. For each a in Σ let:

1. $\delta_3(q, a) = \{\delta_1(q, a)\}$ for each q in $K_1 - F_1$.
2. $\delta_3(q, a) = \{\delta_1(q, a), \delta_2(q_2, a)\}$ for each q in F_1.
3. $\delta_3(q, a) = \{\delta_2(q, a)\}$ for each q in K_2.

The purpose of Rule 1 is to allow M_3 to act like M_1 for some initial segment of the input (possibly ϵ). Rule 2 allows M_3 to continue the simulation of M_1 or to guess that a given symbol starts a word in L_2, provided that the previous symbol completed a word in L_1. Rule 3 allows only the simulation of M_2 after M_3 has guessed that the word from L_2 has been started.

If ϵ is not in L_2, then $F = F_2$. If ϵ is in L_2, then $F = F_1 \cup F_2$.

The *closure* of a language L, denoted by L^*, is the set consisting of the empty string and all finite-length strings formed by concatenating words in L. Thus, if $L = \{01, 11\}$, then $L^* = \{\epsilon, 01, 11, 0101, 0111, 1101, 1111, 010101, \ldots\}$. An alternative definition is $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$, where $L^0 = \{\epsilon\}$ and $L^i = L^{i-1}L$, for $i > 0$.

Theorem 3.9. The class of sets accepted by finite automata is closed under set closure.

---

† Also known as *concatenation* of sets.

Proof. Let $M = (K, \Sigma, \delta, q_0, F)$ be a finite automaton accepting L. We construct a nondeterministic finite automaton M', which behaves as M until an initial portion of a sentence x takes M to a final state. At this time, M' will guess whether or not this point corresponds to a point where a new string from L starts. Formally,

$$M' = (K \cup \{q_0'\}, \Sigma, \delta', \{q_0'\}, F \cup \{q_0'\}),$$

where q_0' is a new state, and

$$\begin{aligned} \delta'(q_0', a) &= \{\delta(q_0, a), q_0\}, && \text{if } \delta(q_0, a) \text{ is in } F, \\ &= \{\delta(q_0, a)\}, && \text{otherwise.} \\ \delta'(q, a) &= \{\delta(q, a), q_0\}, && \text{if } \delta(q, a) \text{ is in } F, \\ &= \{\delta(q, a)\}, && \text{otherwise, for all } q \text{ in } K. \end{aligned}$$

The purpose of the new initial state q_0' is to accept the empty string. If q_0 is not in F, we cannot simply make q_0 a final state since M may come back to q_0 for some input strings. Since the proof is somewhat more difficult than that of the previous theorems, we give a formal proof.

Assume that x is in L^*. Then either $x = \epsilon$, or $x = x_1x_2 \ldots x_n$, where x_i is in L for all i between 1 and n. Clearly M' accepts ϵ. Now x_i in L implies $\delta(q_0, x_i)$ is in F. Thus $\delta'(q_0', x_i)$ and $\delta'(q_0, x_i)$ each contain q_0 and some p (possibly $p = q_0$) in F. Hence, $\delta'(q_0', x)$ contains some state in F, and x is in $T(M')$.

Now assume that $x = a_1a_2 \ldots a_m$ is in $T(M')$. Then there exists some sequence of states $q_1, q_2, \ldots, q_m$ such that $\delta'(q_0', a_1)$ contains q_1, and $\delta'(q_i, a_{i+1})$ contains q_{i+1}, $1 \leqq i < m$, and q_m is in F. Thus, for each i, either $q_{i+1} = q_0$ and $\delta(q_i, a_{i+1})$ is in F or $\delta(q_i, a_{i+1}) = q_{i+1}$. Thus x can be written as $x_1x_2 \ldots x_n$, so that $\delta(q_0, x_i)$ is in F for $1 \leqq i \leqq n$, implying that x_i is in L.

Theorem 3.10. The class of sets accepted by finite automata is the smallest class containing all finite sets and closed under union, product, and closure.

Proof. That the class of sets accepted by finite automata contains the smallest class containing all finite sets and closed under union, product, and closure, is an immediate consequence of Lemma 3.1 and Theorems 3.7, 3.8, and 3.9. It remains to show that the smallest class containing all finite sets and closed under union, product, and closure contains the class of sets accepted by finite automata.

Let L_1 be a set accepted by some finite automaton,

$$M = (\{q_1, \ldots, q_n\}, \Sigma, \delta, q_1, F).$$

Let R_{ij}^k denote the set of all strings x such that $\delta(q_i, x) = q_j$, and if $\delta(q_i, y) = q_l$, for any y which is an initial segment of x other than x or ϵ, then $l \leqq k$.

That is, R_{ij}^k is the set of all strings which take the finite automaton from state q_i to state q_j without going through any state q_l, $l > k$. Note that by "going through a state," we mean both entering and leaving. Thus i or j may be greater than k. We can define R_{ij}^k recursively:

$$\begin{aligned} R_{ij}^k &= R_{ik}^{k-1}(R_{kk}^{k-1})^*R_{kj}^{k-1} \cup R_{ij}^{k-1} \\ R_{ij}^0 &= \{a \mid \delta(q_i, a) = q_j\}. \end{aligned}$$

Informally, the definition of R_{ij}^k above means that the inputs that cause M to go from q_i to q_j without passing through a state higher than q_k are either:

1. in R_{ij}^{k-1}, that is, they never reach a state as high as q_k.
2. composed of a string in R_{ik}^{k-1} (which takes M to q_k for the first time) followed by some number of strings in R_{kk}^{k-1} (which take M from q_k back to q_k without passing through q_k otherwise) followed by a string in R_{kj}^{k-1} (which takes M from state q_k to q_j).

We can show, by induction on k, that R_{ij}^k, $0 \leqq k \leqq l$, is, for all i and j, within the smallest class containing all finite sets and closed under union, complex product, and closure. The induction hypothesis is true for $l = 0$, since all R_{ij}^0 are finite sets. If true for all $k \leqq l$, then it is true for $k = l + 1$ since we can express R_{ij}^{l+1} in terms of union, concatenation, and closure of various sets of the form R_{mn}^l, each of which is presumed to be in the smallest class of sets containing the finite sets and closed under union, concatenation, and closure. Now $L_1 = \bigcup_{q_j \text{ in } F} R_{1j}^n$. Thus L_1 is in the smallest class of sets containing the finite sets, and closed under union, concatenation, and closure.

As a result of Theorem 3.10, we know that any expression made up of finite subsets of Σ^* for some finite alphabet Σ, and a finite number of the operators $\cup$, $\cdot$† and $*$, with parentheses to determine the order of operations, denotes a set that is accepted by a finite automaton. Furthermore, every set accepted by some fa can be so expressed. This provides us with a good notation for describing regular sets. For example, $\{0, 1\}^*\{000\}\{0, 1\}^*$ denotes the set of all strings with three consecutive 0's, and $(\{0, 1\}\{0, 1\})^* \cup (\{0, 1\}\{0,1\}\{0, 1\})^*$ denotes the set of all strings whose length is divisible by two or three.

3.6 SOLVABLE PROBLEMS CONCERNING FINITE AUTOMATA

In this section we show that there are algorithms to answer many questions concerning finite automata and type 3 languages. In Chapter 14 we shall see that no such algorithms can possibly exist to answer some of these questions for the other types of languages discussed in Chapter 2.

† $S_1 \cdot S_2$ is the product S_1S_2.

Theorem 3.11. The set of sentences accepted by a finite automaton with n states is:

1. nonempty if and only if the finite automaton accepts a sentence of length less than n.
2. infinite if and only if the automaton accepts a sentence of length l, $n \leqq l < 2n$.

Thus, there is an algorithm to determine if a finite automaton accepts zero, a finite number, or an infinite number of sentences.

Proof. (1) The "if" portion is obvious. Suppose that a finite automaton $M = (K, \Sigma, \delta, q_0, F)$, with n states accepts some word. Let w be a word as short as any other word accepted. We might as well assume that $|w| \geqq n$, else the result is proven for M. Since there are but n states, M must pass through the same state twice in accepting w. Formally, we can find q in K such that we can write $w = w_1w_2w_3$, with $w_2 \neq \epsilon$, $\delta(q_0, w_1) = q$, $\delta(q, w_2) = q$, and $\delta(q, w_3)$ in F. Then w_1w_3 is in $T(M)$, since

$$\delta(q_0, w_1w_3) = \delta(q_0, w_1w_2w_3).$$

But $|w_1w_3| < |w|$, contradicting the assumption that w is as short as any word in $T(M)$.

(2) We leave most of this part to the reader. We merely observe that if w is in $T(M)$ and $n \leqq |w| < 2n$, then we can write $w = w_1w_2w_3$, $w_2 \neq \epsilon$, and for all i, $w_1w_2^iw_3$ is in $T(M)$. Next, if M accepts an infinity of words, and none is of length between n and $2n - 1$, then let w be of length at least $2n$, but as short as any word in $T(M)$ whose length is $\geqq 2n$. Then, we can write $w = w_1w_2w_3$, with $1 \leqq |w_2| \leqq n$, and w_1w_3 in $T(M)$, thus deriving a contradiction.

In part (1), the algorithm to decide if $T(M)$ is empty is: "See if any word of length up to n is in $T(M)$." Clearly there is such a procedure which is guaranteed to halt. In part (2), the algorithm to decide if $T(M)$ is infinite is: "See if any word of length between n and $2n - 1$ is in $T(M)$." Again, clearly there is such a procedure which is guaranteed to halt.

We now show that there is an algorithm to determine if two type 3 grammars generate the same language. As we shall see later, no such algorithm exists for type 0, 1, or 2 grammars.

Theorem 3.12. There is an algorithm to determine if two finite automata are equivalent (i.e., if they accept the same language).

Proof. Let M_1 and M_2 be fa, accepting L_1 and L_2, respectively. By Theorem 3.6, $(L_1 \cap \bar{L}_2) \cup (\bar{L}_1 \cap L_2)$ is accepted by some finite automaton, M_3. It is easy to see that M_3 accepts a word if and only if $L_1 \neq L_2$. Hence, by Theorem 3.11, there is an algorithm to determine if $L_1 = L_2$.

3.7 TWO-WAY FINITE AUTOMATA

We now turn our attention to finite automata that can move two ways on their input tapes. Our reason for studying these is twofold. First it is easier to introduce the concept of moving two ways on the input with finite automata than with more complicated automata. Second, we wish to introduce the concept of a finite table being stored in a finite control, a tool we shall find useful later on.

A two-way finite automaton M will be represented by a 5-tuple $(K, \Sigma, \delta, q_0, F)$, where K is a set of states, Σ is the set of input symbols, δ is a mapping from $K \times \Sigma$ to $K \times \{L, R, S\}$, q_0 in K is the initial state, and $F \subseteq K$ is the set of final states. The interpretation of $\delta(q, a) = (p, D)$, p in K and D in $\{L, R, S\}$ is that M, in state q, scanning the input symbol a, will move its input head one cell to the left, to the right, or not move its input head at all, depending on whether D equals L, R, or S, respectively. The state of M will also be changed to state p. Note that δ cannot be extended from $K \times \Sigma$ to $K \times \Sigma^*$ in the obvious manner, since one must keep track of the net change in input position.

We define a *configuration* of M to be a state and a number (q, i), where q is the present state of M and i is the location of the input head. That is, i is the number of cells the input head is from the left end of the input.

A two-way finite automaton will start in state q_0 with its input head scanning the leftmost cell on the input tape. Should M ever move off either end of x, M halts. An input word will be *accepted* by M if and only if M eventually moves off the right end of x at the same time it enters a final state.

M can reject a word x by:

1. moving off the left end of x.
2. moving off the right end of x in a nonfinal state.
3. looping.

Theorem 3.13. The class of sets accepted by two-way finite automata is the same as the class of sets accepted by one-way finite automata.

Proof. Let $M = (K, \Sigma, \delta, q_0, F)$ be a two-way finite automaton and $x = a_1a_2 \ldots a_n$ be the input to M. We can associate with each initial segment of x, $a_1a_2 \ldots a_i$, a mapping Δ_i of K to $K \cup \{R\}$. We say that Δ_i is *associated* with $a_1a_2 \ldots a_i$. The interpretation of $\Delta_i(q) = p$, p in K, is that if M is started in state q, scanning the ith cell, M will eventually move right to the $i + 1$st cell. On the move which takes M to the $i + 1$st cell for the first time, M enters state p. Note that before reaching the $i + 1$st cell M may move its input head back and forth on cells 1 through i many times.

The interpretation of $\Delta_i(q) = R$ is that if M is started in state q scanning the ith cell, M will reject x without ever having moved right from the ith cell to the $i + 1$st cell. That is, M will either enter a loop or move off the input tape to the left.

The number of distinct mappings of K to $K \cup \{R\}$ is $(s + 1)^s$, where s is the number of elements in K. We can define a one-way finite automaton $M' = (K', \Sigma, \delta', q_0', F')$, whose states except one are ordered pairs $[q, \Delta]$ where q is in K and Δ is a mapping of K to $K \cup \{R\}$.† Also in K' is a "trap state" t, having the property that $\delta'(t, a) = t$ for all a in Σ. The interpretation of the state $[q, \Delta]$ is that the one-way fa M' will be in state $[q, \Delta]$ after reading an input x if and only if the two-way finite automaton M would be in state q the first time M moves right from string x and the mapping Δ is the mapping associated with the string x. Thus M' carries in its finite control a table which contains the information as to the eventual outcome if M moves left into x in any state.

We define the mapping δ' as follows.

$$\delta'([q_1, \Delta_1], a) = [q_2, \Delta_2]$$

exactly when we can compute q_2 and Δ_2 from q_1 and Δ_1 by:

1. $\Delta_2(p_1) = p$ if there exists a sequence of states $p_2, p_3, \ldots, p_n, p$, such that either
$$\delta(p_i, a) = (p_{i+1}, S)$$
or
$$\delta(p_i, a) = (p_i', L)$$
and
$$\Delta_1(p_i') = p_{i+1}$$
for all i, where $1 \leqq i < n$. Finally, $\delta(p_n, a) = (p, R)$.
2. $\Delta_2(p_1) = R$ if there is no finite sequence satisfying (1). Note that if any sequence $p_1, p_2, \ldots, p_n, p$ satisfies (1), then there is a sequence satisfying (1) such that no state appears twice in the sequence.
3. $q_2 = \Delta_2(q_1)$. If $\Delta_2(q_1) = R$, however, there is no such q_2, therefore $\delta'([q_1, \Delta_1], a)$ is t, the trap state.

Informally, to compute $\Delta_2(p_1)$, assume that Δ_1 is associated with the first $j - 1$ symbols. We begin constructing a sequence of states $p_1, p_2, \ldots$ by adding to the sequence, each time M scans the jth cell, the state of M at that time. Hopefully, from one of these states, M will move to the right and enter some state p. Then we can end the sequence, since we have $\Delta_2(p_1) = p$. Suppose that we have constructed the sequence $p_1, p_2, \ldots, p_i$. Then the fa M will be in configuration (p_i, j), scanning an a on its input. Three cases arise:

1. $\delta(p_i, a) = (q, S)$. Here M will again scan cell j, this time in state q. So $p_{i+1} = q$.

† Do not confuse R in the range of Δ, which means reject, with R in the range of δ which means move right.

2. $\delta(p_i, a) = (q, R)$. Here, we have our answer. M has moved to cell $j + 1$ for the first time and entered state q. Thus, q has the role of p in the sequence, i.e., $\Delta_2(p_1) = q$.
3. $\delta(p_i, a) = (q, L)$. Here, M next moves left to cell $j - 1$. We consult Δ_1 to see what happens next. If $\Delta_1(q) = R$, then M will never again scan cell j, so it cannot scan $j + 1$ either. We therefore let $\Delta_2(p_1) = R$. If $\Delta_1(q) = q'$, then $p_{i+1} = q'$.

The above process will produce a value for $\Delta_2(p_1)$ in all cases, provided that the sequence $p_1, p_2, \ldots$ never repeats a state. If a state is repeated, M is in a loop and will never reach cell $j + 1$. In this case, $\Delta_2(p_1) = R$. We have now covered all contingencies. We leave it to the reader to see that if Δ_1 is the table associated with input x, then Δ_2 will be the table associated with xa.

If M' is to be a one-way finite automaton simulating M, we must let its initial state, q_0', be $[q_0, \Delta_0]$, where Δ_0 is the table associated with ϵ, i.e., $\Delta_0(q) = R$ for all q. Also,

$$F' = \{[q, \Delta] | q \text{ is in } F\}.$$

We must now prove by induction on the length of input x that M' moves right in state $[q, \Delta]$ (for some Δ) from x if and only if M moves right from x in q. Thus M' accepts x if and only if M does. If x is of length 1, the result follows from the way $\delta'([q_0, \Delta_0], x)$ is constructed. That is, if and only if $\delta'([q_0, \Delta_0], x) = [q, \Delta]$, will M eventually move to the right from its first input symbol and enter state q at that time. Note $[q, \Delta]$ is accepting if and only if q is accepting for M.

Suppose that the result is true for $|x| < k$, and let xa be an input of length k. Then M will scan the final symbol, a, of xa for the first time in state q if and only if $\delta'([q_0, \Delta_0], x) = [q, \Delta_1]$ for some Δ_1. But then M will move to the right from xa and enter state p if and only if $\delta'([q, \Delta_1], a) = [p, \Delta_2]$, for the proper Δ_2. This follows from the construction of Δ_2 from Δ_1 just given. The acceptance of xa by M occurs if and only if p is in F. But then $[p, \Delta_2]$ is in F', so M' accepts xa.

Example 3.7. Consider the two-way finite automaton.

$$M = (\{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_1\})$$

where δ is defined by:

$$\delta(q_0, a) = (q_0, R) \quad \delta(q_0, b) = (q_1, S) \quad \delta(q_1, a) = (q_0, L) \quad \delta(q_1, b) = (q_0, L).$$

We can construct a one-way finite automaton $M' = (K, \{a, b\}, \delta', q_0', F)$ equivalent to M. Here K is the set of objects of the form $[q, \Delta]$, where $q = q_0$ or q_1 and Δ is a map from $\{q_0, q_1\}$ to $\{q_0, q_1, R\}$, plus a trap state. There are nine possible values of Δ. The set of all $[q_1, \Delta]$ is F.

$$q_0' = [q_0, \Delta_0], \qquad \text{where } \Delta_0(q_0) = \Delta_0(q_1) = R.$$

Since M' has 19 states, we shall not construct δ' for all of these, but simply give one case. We compute

$$\delta'([q_0, \Delta_1], a) \quad \text{and} \quad \delta'([q_0, \Delta_1], b),$$

where $\Delta_1(q_0) = q_0$ and $\Delta_1(q_1) = q_1$. Let

$$\delta'([q_0, \Delta_1], a) = [q_a, \Delta_a] \quad \text{and} \quad \delta'([q_0, \Delta_1], b) = [q_b, \Delta_b].$$

To compute $\Delta_a(q_0)$, we note that $\delta(q_0, a) = (q_0, R)$. Thus M immediately moves to the right from a and enters state q_0. We have $\Delta_a(q_0) = q_0$. For $\Delta_a(q_1)$, we note that $\delta(q_1, a) = (q_0, L)$. Thus, we must consult $\Delta_1(q_0)$ to see if M will ever return to the symbol a. We have $\Delta_1(q_0) = q_0$, so M does return in state q_0. From a, in state q_0, M next moves to the right, remaining in q_0, since $\delta(q_0, a) = (q_0, R)$. Hence

$$\Delta_a(q_1) = q_0 \quad \text{and} \quad q_a = \Delta_a(q_0) = q_0.$$

To compute $\Delta_b(q_0)$, note that $\delta(q_0, b) = (q_1, S)$, so M would remain scanning the b, but in state q_1. Next, $\delta(q_1, b) = (q_0, L)$. Now, $\Delta_1(q_0) = q_0$, so M would return to a in state q_0, where it started. M is in a loop, so $\Delta_b(q_0) = R$. Likewise, $\Delta_b(q_1) = R$. But $\Delta_b(q_0) = R$, so $\delta'([q_0, \Delta_1], b)$ is the trap state.

PROBLEMS

3.1 Find a one-way, deterministic finite automaton accepting all strings in $\{0, 1\}^*$ such that every 0 has a 1 immediately to its right.

3.2 From the finite automaton of Problem 3.1, construct a type 3 grammar generating the language of that problem.

3.3 Give an example of a relation which is:

a) reflexive and symmetric, but not transitive.
b) symmetric and transitive, but not reflexive.
c) reflexive and transitive, but not symmetric.

3.4 Let

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$$

be a nondeterministic finite automaton, with

$$\delta(q_0, a) = \{q_1, q_2\} \quad \delta(q_1, a) = \{q_0, q_1\} \quad \delta(q_2, a) = \{q_0, q_2\}$$
$$\delta(q_0, b) = \{q_0\} \quad \delta(q_1, b) = \varphi \quad \delta(q_2, b) = \{q_1\}.$$

Find a deterministic finite automaton accepting $T(M)$.

3.5 Complete the specification of the one-way finite automaton in Example 3.7.

3.6 Use the notion of a nondeterministic finite automaton to show that if L is a type 3 language, then

$$L^R = \{w | w \text{ reversed is in } L\}$$

is a type 3 language.

3.7 From Problem 3.6, show that grammars where all productions are of the form $A \to Bb$ or $A \to b$, A and B variables, b terminal, generate all and only the type 3 languages.

3.8 A nondeterministic two-way finite automaton is denoted by $M = (K, \Sigma, \delta, q_0, F)$ as the two-way deterministic finite automaton, except that $\delta(q, a)$, for q in K and a in Σ, is a subset of $K \times \{L, R, S\}$. Each (p, D) in $\delta(q, a)$ represents a possible move of M when the automaton is in state q scanning a on its input. If any sequence of choices of moves causes M to move off the right end of its input in an accepting state, M accepts. Show that only type 3 languages are accepted by nondeterministic, two-way finite automata.

3.9 Let L be a type 3 language. Let Init (L) be the set of words x, such that for some word y, xy is in L. Show that Init (L) is a type 3 language.

3.10 Let R be a type 3 language consisting only of words whose length is divisible by 3. Consider the language formed by taking the first third of each sentence in R. Is this language a type 3 language? What about the last third? Middle third? What about the language formed by concatenating the first and last third of each word in R?

3.11 Some people allow a two-way finite automaton to have an end marker, $\not\subset$, at the left end of each tape. This finite automaton is said to accept w if it moves off the right end of $\not\subset w$ while entering a final state. Show that under this definition, it is still only regular sets which are accepted.

REFERENCES

The notion of a finite state device is usually attributed to McCulloch and Pitts [1943]. The formalism we have used was suggested in Moore [1956] and is found in Rabin and Scott [1959].

Theorem 3.1 is from Nerode [1958] and was proven in a slightly weaker form by Myhill. The minimization of finite automata (Theorem 3.2) appeared originally in Huffman [1954] and Moore [1956]. Nondeterministic finite automata and Theorem 3.3 are from Rabin and Scott [1959]; Theorem 3.13 and two-way finite automata are found in Rabin and Scott [1959] and Shepherdson [1959]. Theorem 3.10 is from Kleene [1956]. The description of regular languages that follows Theorem 3.10 is known as a "regular expression," and is from Kleene [1956]. Many results concerning regular expressions can be found in Brzozowski [1962] and McNaughton and Yamada [1960]. The algorithms in Section 3.6 are from Moore [1956], and the results of Section 3.4 relating type 3 grammars and finite automata are from Chomsky and Miller [1958].

Many books have been written on the subject of finite automata. Among them are Gill [1962] and Ginsburg [1962]. Finite automata are also covered extensively by Harrison [1965], Booth [1967], and Minsky [1967].