Chapter 4

Properties of Regular Languages

We have defined regular languages, studied some ways in which they can be represented, and have seen a few examples of their usefulness. We now raise the question of how general regular languages are. Could it be that every formal language is regular? Perhaps any set we can specify can be accepted by some, albeit very complex, finite automaton. As we will see shortly, the answer to this conjecture is definitely no. But to understand why this is so, we must inquire more deeply into the nature of regular languages and see what properties the whole family has.

The first question we raise is what happens when we perform operations on regular languages. The operations we consider are simple set operations, such as concatenation, as well as operations in which each string of a language is changed, as for instance in Exercise 22, Section 2.1. Is the resulting language still regular? We refer to this as a **closure** question. Closure properties, although mostly of theoretical interest, help us in discriminating between the various language families we will encounter.

A second set of questions about language families deals with our ability to decide on certain properties. For example, can we tell whether a language is finite or not? As we will see, such questions are easily answered for regular languages, but are not as easily answered for other language families.

Finally we consider the important question: How can we tell whether a given language is regular or not? If the language is in fact regular, we can always show it by giving some dfa, regular expression, or regular grammar for it. But if it is not, we need another line of attack. One way to show a language is not regular is to study the general properties of regular languages, that is, characteristics that are shared by all regular languages. If we know of some such property, and if we can show that the candidate language does not have it, then we can tell that the language is not regular.

In this chapter, we look at a variety of properties of regular languages. These properties tell us a great deal about what regular languages can and cannot do. Later, when we look at the same questions for other language families, similarities and differences in these properties will allow us to contrast the various language families.

4.1 Closure Properties of Regular Languages

Consider the following question: Given two regular languages L_1 and L_2, is their union also regular? In specific instances, the answer may be obvious, but here we want to address the problem in general. Is it true for all regular L_1 and L_2? It turns out that the answer is yes, a fact we express by saying that the family of regular languages is **closed** under union. We can ask similar questions about other types of operations on languages; this leads us to the study of the closure properties of languages in general.

Closure properties of various language families under different operations are of considerable theoretical interest. At first sight, it may not be clear what practical significance these properties have. Admittedly, some of them have very little, but many results are useful. By giving us insight into the general nature of language families, closure properties help us answer other, more practical questions. We will see instances of this (Theorem 4.7 and Example 4.13) later in this chapter.

Closure under Simple Set Operations

We begin by looking at the closure of regular languages under the common set operations, such as union and intersection.

Theorem 4.1

If L_1 and L_2 are regular languages, then so are $L_1 \cup L_2$, $L_1 \cap L_2$, L_1L_2, $\overline{L_1}$ and L_1^*. We say that the family of regular languages is closed under union, intersection, concatenation, complementation, and star-closure.

Proof: If L_1 and L_2 are regular, then there exist regular expressions r_1 and r_2 such that $L_1 = L(r_1)$ and $L_2 = L(r_2)$. By definition, $r_1 + r_2$, r_1r_2, and

r_1^* are regular expressions denoting the languages $L_1 \cup L_2$, L_1L_2, and L_1^*, respectively. Thus, closure under union, concatenation, and star-closure is immediate.

To show closure under complementation, let $M = (Q, \Sigma, \delta, q_0, F)$ be a dfa that accepts L_1. Then the dfa

$$\widehat{M} = (Q, \Sigma, \delta, q_0, Q - F)$$

accepts $\overline{L_1}$. This is rather straightforward; we have already suggested the result in Exercise 4 in Section 2.1. Note that in the definition of a dfa, we assumed δ^* to be a total function, so that $\delta^*(q_0, w)$ is defined for all $w \in \Sigma^*$. Consequently either $\delta^*(q_0, w)$ is a final state, in which case $w \in L$, or $\delta^*(q_0, w) \in Q - F$ and $w \in \overline{L}$.

Demonstrating closure under intersection takes a little more work. Let $L_1 = L(M_1)$ and $L_2 = L(M_2)$, where $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ and $M_2 = (P, \Sigma, \delta_2, p_0, F_2)$ are dfa's. We construct from M_1 and M_2 a combined automaton $\widehat{M} = \left(\widehat{Q}, \Sigma, \widehat{\delta}, (q_0, p_0), \widehat{F}\right)$, whose state set $\widehat{Q} = Q \times P$ consists of pairs (q_i, p_j), and whose transition function $\widehat{\delta}$ is such that $\widehat{M}$ is in state (q_i, p_j) whenever M_1 is in state q_i and M_2 is in state p_j. This is achieved by taking

$$\widehat{\delta}((q_i, p_j), a) = (q_k, p_l),$$

whenever

$$\delta_1(q_i, a) = q_k$$

and

$$\delta_2(p_j, a) = p_l.$$

$\widehat{F}$ is defined as the set of all (q_i, p_j), such that $q_i \in F_1$ and $p_j \in F_2$. Then it is a simple matter to show that $w \in L_1 \cap L_2$ if and only if it is accepted by $\widehat{M}$. Consequently, $L_1 \cap L_2$ is regular. ■

The proof of closure under intersection is a good example of a constructive proof. Not only does it establish the desired result, but it also shows explicitly how to construct a finite accepter for the intersection of two regular languages. Constructive proofs occur throughout this book; they are important because they give us insight into the results and often serve as the starting point for practical algorithms. Here, as in many cases, there are shorter but nonconstructive (or at least not so obviously constructive) arguments. For closure under intersection, we start with DeMorgan's law, Equation (1.3), taking the complement of both sides. Then

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

for any languages L_1 and L_2. Now, if L_1 and L_2 are regular, then by closure under complementation, so are $\overline{L_1}$ and $\overline{L_2}$. Using closure under union, we next get that $\overline{L_1} \cup \overline{L_2}$ is regular. Using closure under complementation once more, we see that

$$\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2$$

is regular.

The following example is a variation on the same idea.

Example 4.1

Show that the family of regular languages is closed under difference. In other words, we want to show that if L_1 and L_2 are regular, then $L_1 - L_2$ is necessarily regular also.

The needed set identity is immediately obvious from the definition of a set difference, namely

$$L_1 - L_2 = L_1 \cap \overline{L_2}.$$

The fact that L_2 is regular implies that $\overline{L_2}$ is also regular. Then, because of the closure of regular languages under intersection, we know that $L_1 \cap \overline{L_2}$ is regular, and the argument is complete. ■

A variety of other closure properties can be derived directly by elementary arguments.

Theorem 4.2

The family of regular languages is closed under reversal.

Proof: The proof of this theorem was suggested as an exercise in Section 2.3. Here are the details. Suppose that L is a regular language. We then construct an nfa with a single final state for it. By Exercise 7, Section 2.3, this is always possible. In the transition graph for this nfa we make the initial vertex a final vertex, the final vertex the initial vertex, and reverse the direction on all the edges. It is a fairly straightforward matter to show that the modified nfa accepts w^R if and only if the original nfa accepts w. Therefore, the modified nfa accepts L^R, proving closure under reversal. ■

Closure under Other Operations

In addition to the standard operations on languages, one can define other operations and investigate closure properties for them. There are many such results; we select only two typical ones. Others are explored in the exercises at the end of this section.

Definition 4.1

Suppose Σ and Γ are alphabets. Then a function

$$h : \Sigma \rightarrow \Gamma^*$$

is called a **homomorphism.** In words, a homomorphism is a substitution in which a single letter is replaced with a string. The domain of the function h is extended to strings in an obvious fashion; if

$$w = a_1 a_2 \cdots a_n,$$

then

$$h(w) = h(a_1) h(a_2) \cdots h(a_n).$$

If L is a language on Σ, then its **homomorphic image** is defined as

$$h(L) = \{h(w) : w \in L\}.$$

Example 4.2

Let $\Sigma = \{a, b\}$ and $\Gamma = \{a, b, c\}$ and define h by

$$h(a) = ab,$$
$$h(b) = bbc.$$

Then $h(aba) = abbbcab$. The homomorphic image of $L = \{aa, aba\}$ is the language $h(L) = \{abab, abbbcab\}$.

■

If we have a regular expression r for a language L, then a regular expression for $h(L)$ can be obtained by simply applying the homomorphism to each Σ symbol of r.

Example 4.3

Take $\Sigma = \{a, b\}$ and $\Gamma = \{b, c, d\}$. Define h by

$$h(a) = dbcc,$$
$$h(b) = bdc.$$

If L is the regular language denoted by

$$r = (a + b^*)(aa)^*,$$

then

$$r_1 = \left(dbcc + (bdc)^*\right)(dbccdbcc)^*$$

denotes the regular language $h(L)$. ■

The general result on the closure of regular languages under any homomorphism follows from this example in an obvious manner.

Theorem 4.3

Let h be a homomorphism. If L is a regular language, then its homomorphic image $h(L)$ is also regular. The family of regular languages is therefore closed under arbitrary homomorphisms.

Proof: Let L be a regular language denoted by some regular expression r. We find $h(r)$ by substituting $h(a)$ for each symbol $a \in \Sigma$ of r. It can be shown directly by an appeal to the definition of a regular expression that the result is a regular expression. It is equally easy to see that the resulting expression denotes $h(L)$. All we need to do is to show that for every $w \in L(r)$, the corresponding $h(w)$ is in $L(h(r))$ and conversely that for every v in $L(h(r))$ there is a w in L, such that $v = h(w)$. Leaving the details as an exercise, we claim that $h(L)$ is regular. ■

Definition 4.2

Let L_1 and L_2 be languages on the same alphabet. Then the **right quotient** of L_1 with L_2 is defined as

$$L_1/L_2 = \{x : xy \in L_1 \text{ for some } y \in L_2\}. \tag{4.1}$$

To form the right quotient of L_1 with L_2, we take all the strings in L_1 that have a suffix belonging to L_2. Every such string, after removal of this suffix, belongs to L_1/L_2.

Example 4.4 If

$$L_1 = \{a^n b^m : n \geq 1, m \geq 0\} \cup \{ba\}$$

and

$$L_2 = \{b^m : m \geq 1\},$$

then

$$L_1/L_2 = \{a^n b^m : n \geq 1, m \geq 0\}.$$

The strings in L_2 consist of one or more b's. Therefore, we arrive at the answer by removing one or more b's from those strings in L_1 that terminate with at least one b as a suffix.

Note that here L_1, L_2, and L_1/L_2 are all regular. This suggests that the right quotient of any two regular languages is also regular. We will prove this in the next theorem by a construction that takes the dfa's for L_1 and L_2 and constructs from them a dfa for L_1/L_2. Before we describe the construction in full, let us see how it applies to this example. We start with a dfa for L_1; say the automaton $M_1 = (Q, \Sigma, \delta, q_0, F)$ in Figure 4.1. Since an automaton for L_1/L_2 must accept any prefix of strings in L_1, we will try to modify M_1 so that it accepts x if there is any y satisfying (4.1).

Figure 4.1

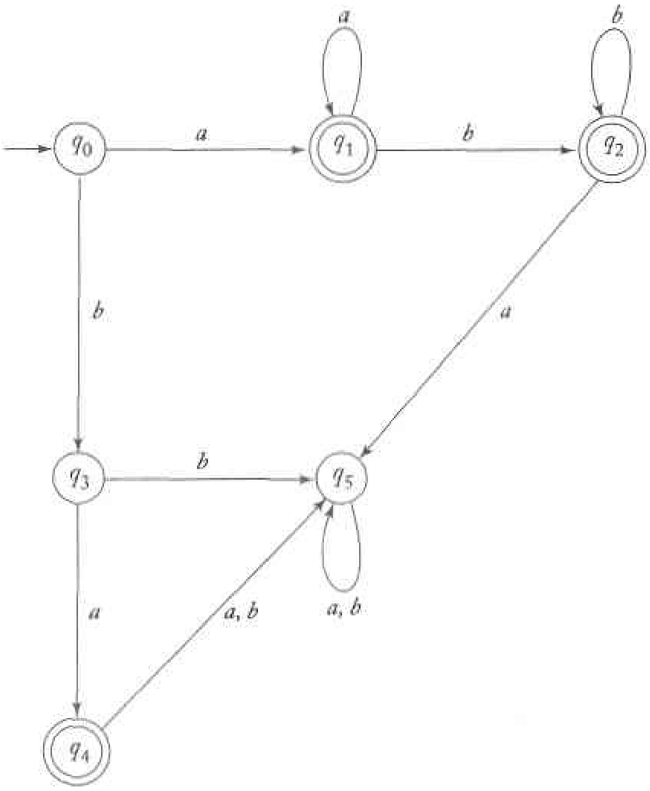
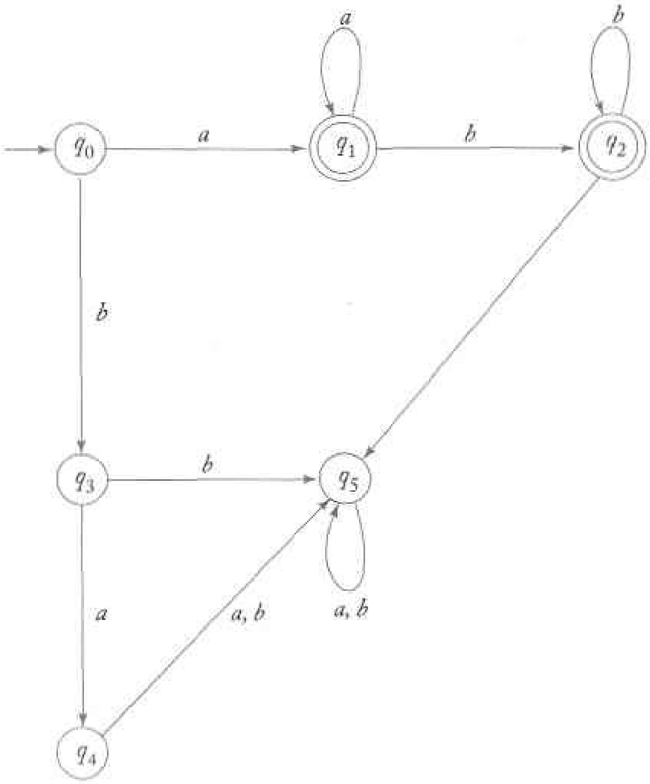


Figure 4.2



The difficulty comes in finding whether there is some y such that $xy \in L_1$ and $y \in L_2$. To solve it, we determine, for each $q \in Q$, whether there is a walk to a final state labeled v such that $v \in L_2$. If this is so, any x such that $\delta(q_0, x) = q$ will be in L_1/L_2. We modify the automaton accordingly to make q a final state.

To apply this to our present case, we check each state q_0, q_1, q_2, q_3, q_4, q_5 to see whether there is a walk labeled bb^* to any of the q_1, q_2, or q_4. We see that only q_1 and q_2 qualify; q_0, q_3, q_4 do not. The resulting automaton for L_1/L_2 is shown in Figure 4.2. Check it to see that the construction works. The idea is generalized in the next theorem. ■

Theorem 4.4

If L_1 and L_2 are regular languages, then L_1/L_2 is also regular. We say that the family of regular languages is closed under right quotient with a regular language.

Proof: Let $L_1 = L(M)$, where $M = (Q, \Sigma, \delta, q_0, F)$ is a dfa. We construct another dfa $\widehat{M} = \left(Q, \Sigma, \delta, q_0, \widehat{F}\right)$ as follows. For each $q_i \in Q$, determine if there exists a $y \in L_2$ such that

$$\delta^*(q_i, y) = q_f \in F.$$

This can be done by looking at dfa's $M_i = (Q, \Sigma, \delta, q_i, F)$. The automaton M_i is M with the initial state q_0 replaced by q_i. We now determine whether there exists a y in $L(M_i)$ that is also in L_2. For this, we can use the construction for the intersection of two regular languages given in Theorem 4.1, finding the transition graph for $L_2 \cap L(M_i)$. If there is any path between its initial vertex and any final vertex, then $L_2 \cap L(M_i)$ is not empty. In that case, add q_i to $\widehat{F}$. Repeating this for every $q_i \in Q$, we determine $\widehat{F}$ and thereby construct $\widehat{M}$.

To prove that $L\left(\widehat{M}\right) = L_1/L_2$, let x be any element of L_1/L_2. Then there must be a $y \in L_2$ such that $xy \in L_1$. This implies that

$$\delta^*(q_0, xy) \in F,$$

so that there must be some $q \in Q$ such that

$$\delta^*(q_0, x) = q$$

and

$$\delta^*(q, y) \in F.$$

Therefore, by construction, $q \in \widehat{F}$, and $\widehat{M}$ accepts x because $\delta^*(q_0, x)$ is in $\widehat{F}$.

Conversely, for any x accepted by $\widehat{M}$, we have

$$\delta^*(q_0, x) = q \in \widehat{F}.$$

But again by construction, this implies that there exists a $y \in L_2$ such that $\delta^*(q, y) \in F$. Therefore xy is in L_1, and x is in L_1/L_2. We therefore conclude that

$$L\left(\widehat{M}\right) = L_1/L_2,$$

and from this that L_1/L_2 is regular. ■

Example 4.5 Find L_1/L_2 for

$$L_1 = L(a^*baa^*),$$
$$L_2 = L(ab^*).$$

We first find a dfa that accepts L_1. This is easy, and a solution is given in Figure 4.3. The example is simple enough so that we can skip the formalities of the construction. From the graph in Figure 4.3 it is quite evident that

$$L(M_0) \cap L_2 = \varnothing,$$
$$L(M_1) \cap L_2 = \{a\} \neq \varnothing,$$
$$L(M_2) \cap L_2 = \{a\} \neq \varnothing,$$
$$L(M_3) \cap L_2 = \varnothing.$$

Figure 4.3

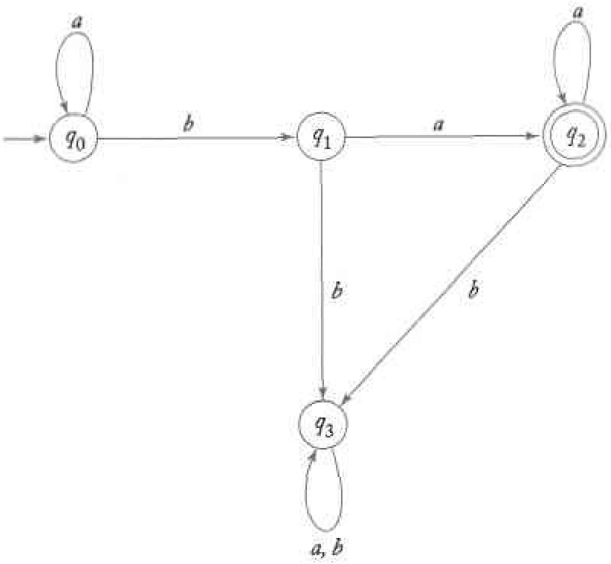
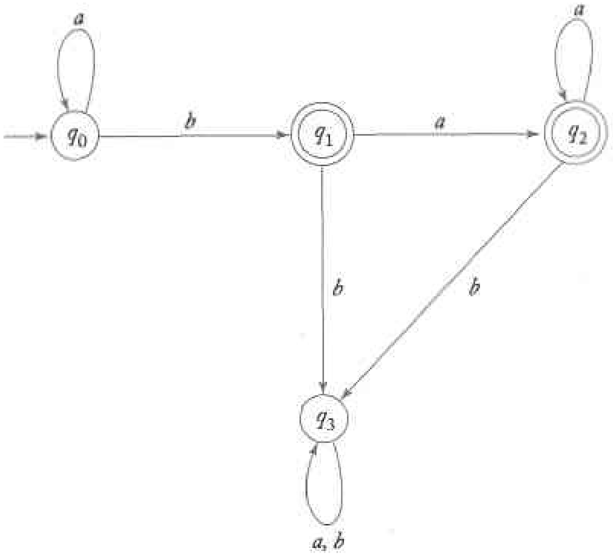


Figure 4.4



Therefore, the automaton accepting L_1/L_2 is determined. The result is shown in Figure 4.4. It accepts the language denoted by the regular expression of $a^*b + a^*baa^*$, which can be simplified to a^*ba^*. Thus $L_1/L_2 = L(a^*ba^*)$. ■

Exercises

1. Fill in the details of the constructive proof of closure under intersection in Theorem 4.1.

2. Use the construction in Theorem 4.1 to find nfa's that accept

   (a) $L((a+b)a^*) \cap L(baa^*)$, ●

   (b) $L(ab^*a^*) \cap L(a^*b^*a)$.

3. In Example 4.1 we showed closure under difference for regular languages, but the proof was nonconstructive. Provide a constructive argument for this result.

4. In the proof of Theorem 4.3, show that $h(r)$ is a regular expression. Then show that $h(r)$ denotes $h(L)$.

5. Show that the family of regular languages is closed under finite union and intersection, that is, if $L_1, L_2, ..., L_n$ are regular, then

$$L_U = \bigcup_{i=\{1,2,...,n\}} L_i$$

and

$$L_I = \bigcap_{i=\{1,2,...,n\}} L_i$$

are also regular.

6. The **symmetric difference** of two sets S_1 and S_2 is defined as

$$S_1 \ominus S_2 = \{x : x \in S_1 \text{ or } x \in S_2, \text{ but } x \text{ is not in both } S_1 \text{ and } S_2\}.$$

Show that the family of regular languages is closed under symmetric difference.

7. The *nor* of two languages is

$$nor(L_1, L_2) = \{w : w \notin L_1 \text{ and } w \notin L_2\}.$$

Show that the family of regular languages is closed under the *nor* operation. ●

8. Define the complementary or (*cor*) of two languages by

$$cor(L_1, L_2) = \{w : w \in \overline{L_1} \quad \text{or} \quad w \in \overline{L_2}\}.$$

Show that the family of regular languages is closed under the *cor* operation.

8 9. Which of the following are true for all regular languages and all homomorphisms?

   (a) $h(L_1 \cup L_2) = h(L_1) \cup h(L_2)$ True

   (b) $h(L_1 \cap L_2) = h(L_1) \cap h(L_2)$ False

   (c) $h(L_1 L_2) = h(L_1) h(L_2)$ True

9 **10.** Let $L_1 = L(a^*baa^*)$ and $L_2 = L(aba^*)$. Find L_1/L_2.

10 & **11.** Show that $L_1 = L_1L_2/L_2$ is not true for all languages L_1 and L_2.

11 ★**12.** Suppose we know that $L_1 \cup L_2$ is regular and that L_1 is finite. Can we conclude from this that L_2 is regular? ●

13. If L is a regular language, prove that $L_1 = \{uv : u \in L, |v| = 2\}$ is also regular.

14. If L is a regular language, prove that the language $\{uv : u \in L, v \in L^R\}$ is also regular. ●

15. If L_1 and L_1L_2 are regular, so is L_2? True or false

14 **15.** The left quotient of a language L_1 with respect to L_2 is defined as

$$L_2/L_1 = \{y : x \in L_2, xy \in L_1\}.$$

Show that the family of regular languages is closed under the left quotient with a regular language.

16. Show that, if the statement "If L_1 is regular and $L_1 \cup L_2$ is also regular, then L_2 must be regular" were true for all L_1 and L_2, then all languages would be regular. ●

17. The *tail* of a language is defined as the set of all suffixes of its strings, that is

$$tail(L) = \{y : xy \in L \text{ for some } x \in \Sigma^*\}.$$

Show that if L is regular, so is $tail(L)$.

18. The *head* of a language is the set of all prefixes of its strings, that is,

$$head(L) = \{x : xy \in L \text{ for some } y \in \Sigma^*\}.$$

Show that the family of regular languages is closed under this operation. ●

19. Define an operation *third* on strings and languages as

$$third(a_1a_2a_3a_4a_5a_6\cdots) = a_3a_6\cdots$$

with the appropriate extension of this definition to languages. Prove the closure of the family of regular languages under this operation.

20. For a string $a_1a_2\cdots a_n$ define the operation *shift* as

$$shift(a_1a_2\cdots a_n) = a_2\cdots a_na_1.$$

From this, we can define the operation on a language as

$$shift(L) = \{v : v = shift(w) \text{ for some } w \in L\}.$$

Show that regularity is preserved under the *shift* operation.

19 **21.** Define

$$exchange(a_1a_2\cdots a_{n-1}a_n) = a_na_2\cdots a_{n-1}a_1,$$

and

$$exchange(L) = \{v : v = exchange(w) \text{ for some } w \in L\}.$$

Show that the family of regular languages is closed under *exchange*.

★**22.** The shuffle of two languages L_1 and L_2 is defined as

$$\mathit{shuffle}(L_1, L_2) = \{w_1 v_1 w_2 v_2 \cdots w_m v_m : w_1 w_2 ... w_m \in L_1,$$
$$v_1 v_2 ... v_m \in L_2, \text{ for all } w_i, v_i \in \Sigma^*\}$$

Show that the family of regular languages is closed under the shuffle operation.

★**23.** Define an operation *minus5* on a language L as the set of all strings of L with the fifth symbol from the left removed (strings of length less than five are left unchanged). Show that the family of regular languages is closed under the *minus5* operation.

★**24.** Define the operation *leftside* on L by

$$\mathit{leftside}(L) = \left\{w : ww^R \in L\right\}.$$

Is the family of regular languages closed under this operation?

25. The *min* of a language L is defined as

$$\mathit{min}(L) = \left\{w \in L : \text{ there is no } u \in L, v \in \Sigma^+, \text{ such that } w = uv\right\}.$$

Show that the family of regular languages is closed under the *min* operation.

24 **26.** Let G_1 and G_2 be two regular grammars. Show how one can derive regular grammars for the languages

(a) $L(G_1) \cup L(G_2)$ ●

(b) $L(G_1) L(G_2)$ ●

(c) $L(G_1)^*$ ●

4.2 Elementary Questions about Regular Languages

We now come to a very fundamental issue: Given a language L and a string w, can we determine whether or not w is an element of L? This is the **membership** question and a method for answering it is called a membership algorithm. Very little can be done with languages for which we cannot find efficient membership algorithms. The question of the existence and nature of membership algorithms will be of great concern in later discussions; it is an issue that is often difficult. For regular languages, though, it is an easy matter.

We first consider what exactly we mean when we say "given a language...." In many arguments, it is important that this be unambiguous. We have used several ways of describing regular languages: informal verbal descriptions, set notation, finite automata, regular expressions, and regular grammars. Only the last three are sufficiently well defined for use in theorems. We therefore say that a regular language is given in a **standard representation** if and only if it is described by a finite automaton, a regular expression, or a regular grammar.

Theorem 4.5

Given a standard representation of any regular language L on Σ and any $w \in \Sigma^*$, there exists an algorithm for determining whether or not w is in L.

Proof: We represent the language by some dfa, then test w to see if it is accepted by this automaton. ■

Other important questions are whether a language is finite or infinite, whether two languages are the same, and whether one language is a subset of another. For regular languages at least, these questions are easily answered.

Theorem 4.6

There exists an algorithm for determining whether a regular language, given in standard representation, is empty, finite, or infinite.

Proof: The answer is apparent if we represent the language as a transition graph of a dfa. If there is a simple path from the initial vertex to any final vertex, then the language is not empty.

To determine whether or not a language is infinite, find all the vertices that are the base of some cycle. If any of these are on a path from an initial to a final vertex, the language is infinite. Otherwise, it is finite. ■

The question of the equality of two languages is also an important practical issue. Often several definitions of a programming language exist, and we need to know whether, in spite of their different appearances, they specify the same language. This is generally a difficult problem; even for regular languages the argument is not obvious. It is not possible to argue on a sentence-by-sentence comparison, since this works only for finite languages. Nor is it easy to see the answer by looking at the regular expressions, grammars, or dfa's. An elegant solution uses the already established closure properties.

Theorem 4.7

Given standard representations of two regular languages L_1 and L_2, there exists an algorithm to determine whether or not $L_1 = L_2$.

Proof: Using L_1 and L_2 we define the language

$$L_3 = \left(L_1 \cap \overline{L_2}\right) \cup \left(\overline{L_1} \cap L_2\right).$$

By closure, L_3 is regular, and we can find a dfa M that accepts L_3. Once we have M we can then use the algorithm in Theorem 4.6 to determine if L_3 is empty. But from Exercise 8, Section 1.1 we see that $L_3 = \varnothing$ if and only if $L_1 = L_2$. ■

These results are fundamental, in spite of being obvious and unsurprising. For regular languages, the questions raised by Theorems 4.5 to 4.7 can be answered easily, but this is not always the case when we deal with larger families of languages. We will encounter questions like these on several occasions later on. Anticipating a little, we will see that the answers become increasingly more difficult, and eventually impossible to find.

Exercises

For all the exercises in this section, assume that regular languages are given in standard representation.

1. Show that there exists an algorithm to determine whether or not $w \in L_1 - L_2$, for any given w and any regular languages L_1 and L_2. ●

2. Show that there exists an algorithm for determining if $L_1 \subseteq L_2$, for any regular languages L_1 and L_2. ●

3. Show that there exists an algorithm for determining if $\lambda \in L$, for any regular language L.

4. Show that for any regular L_1 and L_2, there is an algorithm to determine whether or not $L_1 = L_1/L_2$.

5. A language is said to be a *palindrome* language if $L = L^R$. Find an algorithm for determining if a given regular language is a palindrome language. ●

6. Exhibit an algorithm for determining whether or not a regular language L contains any string w such that $w^R \in L$.

7. Exhibit an algorithm that, given any three regular languages, L, L_1, L_2, determines whether or not $L = L_1 L_2$.

8. Exhibit an algorithm that, given any regular language L, determines whether or not $L = L^*$.

9. Let L be a regular language on Σ and $\widehat{w}$ be any string in Σ^*. Find an algorithm to determine if L contains any w such that $\widehat{w}$ is a substring of it, that is, such that $w = u\widehat{w}v$, with $u, v \in \Sigma^*$.

10. Show that there is an algorithm to determine if $L = shuffle\,(L, L)$ for any regular L.

11. The operation *tail* (L) is defined as

$$tail\,(L) = \{v : uv \in L, u, v \in \Sigma^*\}.$$

Show that there is an algorithm for determining whether or not $L = tail\,(L)$ for any regular L.

12. Let L be any regular language on $\Sigma = \{a, b\}$. Show that an algorithm exists for determining if L contains any strings of even length.

13. Find an algorithm for determining whether a regular language L contains an infinite number of even-length strings.

14. Describe an algorithm which, when given a regular grammar G, can tell us whether or not $L\,(G) = \Sigma^*$.

4.3 Identifying Nonregular Languages

Regular languages can be infinite, as most of our examples have demonstrated. The fact that regular languages are associated with automata that have finite memory, however, imposes some limits on the structure of a regular language. Some narrow restrictions must be obeyed if regularity is to hold. Intuition tells us that a language is regular only if, in processing any string, the information that has to be remembered at any stage is strictly limited. This is true, but has to be shown precisely to be used in any meaningful way. There are several ways in which this precision can be achieved.

Using the Pigeonhole Principle

The term "pigeonhole principle" is used by mathematicians to refer to the following simple observation. If we put n objects into m boxes (pigeonholes), and if $n > m$, then at least one box must have more than one item in it. This is such an obvious fact that it is surprising how many deep results can be obtained from it.

Example 4.6 Is the language $L = \{a^n b^n : n \geq 0\}$ regular? The answer is no, as we show using a proof by contradiction.

Suppose L is regular. Then some dfa $M = (Q, \{a, b\}, \delta, q_0, F)$ exists for it. Now look at $\delta^*\left(q_0, a^i\right)$ for $i = 1, 2, 3, ...$. Since there are an unlimited number of i's, but only a finite number of states in M, the pigeonhole principle tells us that there must be some state, say q, such that

$$\delta^*\left(q_0, a^n\right) = q$$

and

$$\delta^*\left(q_0, a^m\right) = q,$$

with $n \neq m$. But since M accepts $a^n b^n$ we must have

$$\delta^* (q, b^n) = q_f \in F.$$

From this we can conclude that

$$\begin{aligned}\delta^* (q_0, a^m b^n) &= \delta^* (\delta^* (q_0, a^m), b^n) \\ &= \delta^* (q, b^n) \\ &= q_f.\end{aligned}$$

This contradicts the original assumption that M accepts $a^m b^n$ only if $n = m$, and leads us to conclude that L cannot be regular. ■

In this argument, the pigeonhole principle is just a way of stating precisely what we mean when we say that a finite automaton has a limited memory. To accept all $a^n b^n$, an automaton would have to differentiate between all prefixes a^n and a^m. But since there are only a finite number of internal states with which to do this, there are some n and m for which the distinction cannot be made.

In order to use this type of argument in a variety of situations, it is convenient to codify it as a general theorem. There are several ways to do this; the one we give here is perhaps the most famous one.

A Pumping Lemma

The following result, known as the **pumping lemma** for regular languages, uses the pigeonhole principle in another form. The proof is based on the observation that in a transition graph with n vertices, any walk of length n or longer must repeat some vertex, that is, contain a cycle.

Theorem 4.8

Let L be an infinite regular language. Then there exists some positive integer m such that any $w \in L$ with $|w| \geq m$ can be decomposed as

$$w = xyz,$$

with

$$|xy| \leq m,$$

and

$$|y| \geq 1,$$

such that

$$w_i = xy^i z, \tag{4.2}$$

is also in L for all $i = 0, 1, 2, \ldots$.

To paraphrase this, every sufficiently long string in L can be broken into three parts in such a way that an arbitrary number of repetitions of the middle part yields another string in L. We say that the middle string is "pumped," hence the term pumping lemma for this result.

Proof: If L is regular, there exists a dfa that recognizes it. Let such a dfa have states labeled $q_0, q_1, q_2, ..., q_n$. Now take a string w in L such that $|w| \boxed{\geq m = n+1}$. Since L is assumed to be infinite, this can always be done. Consider the set of states the automaton goes through as it processes w, say

$$q_0, q_i, q_j, ..., q_f.$$

Since this sequence has exactly $|w| + 1$ entries, at least one state must be repeated, and such a repetition must start no later than the nth move. Thus the sequence must look like

$$q_0, q_i, q_j, ..., q_r, ..., q_r, ..., q_f,$$

indicating there must be substrings x, y, z of w such that

$$\begin{aligned} \delta^* (q_0, x) &= q_r, \\ \delta^* (q_r, y) &= q_r, \\ \delta^* (q_r, z) &= q_f, \end{aligned}$$

with $|xy| \leq n + 1 = m$ and $|y| \geq 1$. From this it immediately follows that

$$\delta^* (q_0, xz) = q_f,$$

as well as

$$\begin{aligned} \delta^* \left(q_0, xy^2z\right) &= q_f, \\ \delta^* \left(q_0, xy^3z\right) &= q_f, \end{aligned}$$

and so on, completing the proof of the theorem. ■

We have given the pumping lemma only for infinite languages. Finite languages, although always regular, cannot be pumped since pumping automatically creates an infinite set. The theorem does hold for finite languages, but it is vacuous empty. The m in the pumping lemma is to be taken larger than the longest string, so that no string can be pumped.

The pumping lemma, like the pigeonhole argument in Example 4.6, is used to show that certain languages are not regular. The demonstration is always by contradiction. There is nothing in the pumping lemma, as we have stated it here, which can be used for proving that a language is regular.

Even if we could show (and this is normally quite difficult) that any pumped string must be in the original language, there is nothing in the statement of Theorem 4.8 that allows us to conclude from this that the language is regular.

Example 4.7 Using the pumping lemma to show that $L = \{a^n b^n : n \geq 0\}$ is not regular. Assume that L is regular, so that the pumping lemma must hold. We do not know the value of m, but whatever it is, we can always choose $\boxed{n = m.}$ Therefore, the substring y must consist entirely of a's. Suppose $|y| = k$. ≥ 1 Then the string obtained by using $i = 0$ in Equation (4.2) is

$$w_0 = a^{m-k} b^m$$

and is clearly not in L. This contradicts the pumping lemma and thereby indicates that the assumption that L is regular must be false. ■

In applying the pumping lemma, we must keep in mind what the theorem says. We are guaranteed the existence of an m as well as the decomposition xyz, but we do not know what they are. We cannot claim that we have reached a contradiction just because the pumping lemma is violated for some specific values of m or xyz. On the other hand, the pumping lemma holds for every $w \in L$ and every i. Therefore, if the pumping lemma is violated even for one w or i, then the language cannot be regular.

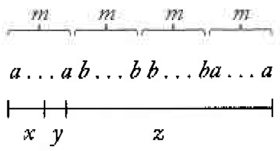
The correct argument can be visualized as a game we play against an opponent. Our goal is to win the game by establishing a contradiction of the pumping lemma, while the opponent tries to foil us. There are four moves in the game.

1. ✱ The opponent picks m.
2. Given m, we pick a string w in L of length equal or greater than m. We are free to choose any w, subject to $w \in L$ and $|w| \geq m$.
3. The opponent chooses the decomposition xyz, subject to $|xy| \leq m$, $|y| \geq 1$. We have to assume that the opponent makes the choice that will make it hardest for us to win the game.
4. We try to pick i in such a way that the pumped string w_i, defined in Equation (4.2), is not in L. If we can do so, we win the game.

quivalent in value

A strategy that allows us to win whatever the opponent's choices is tantamount to a proof that the language is not regular. In this, Step 2 is crucial. While we cannot force the opponent to pick a particular decomposition of w, we may be able to choose w so that the opponent is very

Figure 4.5



restricted in Step 3, forcing a choice of x, y, and z that allows us to produce a violation of pumping lemma on our next move.

Example 4.8

Let $\Sigma = \{a, b\}$. Show that

$$L = \{ww^R : w \in \Sigma^*\}$$

is not regular.

Note that w_i should be of the form ww^R so it cannot force to have equal a's or b's i.e w_i not force to be of the form w_i but has belong to L.

Whatever m the opponent picks on Step 1, we can always choose a w as shown in Figure 4.5. Because of this choice, and the requirement that $\boxed{|xy| \leq m}$, the opponent is restricted in Step 3 to choosing a y that consists entirely of a's. In Step 4, we use $i = 0$. The string obtained in this fashion has fewer a's on the left than on the right and so cannot be of the form ww^R. Therefore L is not regular.

Note that if we had chosen w too short, then the opponent could have chosen a y with an even number of b's. In that case, we could not have reached a violation of the pumping lemma on the last step. We would also fail if we were to choose a string consisting of all a's, say,

$$w = a^{2m},$$

which is in L. To defeat us, the opponent need only pick

$$y = aa.$$

Now w_i is in L for all i, and we lose.

To apply the pumping lemma we cannot assume that the opponent will make a wrong move. If, in the case where we pick $w = a^{2m}$, the opponent were to pick

$$y = a,$$

then w_0 is a string of odd length and therefore not in L. But any argument that assumes that the opponent is so accommodating is automatically incorrect. ■

Example 4.9 Let $\Sigma = \{a, b\}$. The language

$$L = \{w \in \Sigma^* : n_a(w) < n_b(w)\}$$

is not regular.

Suppose we are given m. Since we have complete freedom in choosing w, we pick $w = a^m b^{m+1}$. Now, because $|xy|$ cannot be greater than m, the opponent cannot do anything but pick a y with all a's, that is

$$y = a^k, \qquad 1 \leq k \leq m.$$

We now pump up, using $i = 2$. The resulting string

$$w_2 = a^{m+k} b^{m+1}$$

is not in L. Therefore, the pumping lemma is violated, and L is not regular. ■

Example 4.10 The language

$$L = \left\{(ab)^n a^k : n > k, k \geq 0\right\}$$

is not regular.

Given m, we pick as our string

$$w = (ab)^{m+1} a^m$$

which is in L. Because of the constraint $|xy| \leq m$, both x and y must be in the part of the string made up of ab's. The choice of x does not affect the argument, so let us see what can be done with y. If our opponent picks $y = a$, we choose $i = 0$ and get a string not in $L\left((ab)^* a^*\right)$. If the opponent picks $y = ab$, we can choose $i = 0$ again. Now we get the string $(ab)^m a^m$, which is not in L. In the same way, we can deal with any possible choice by the opponent, thereby proving our claim. ■

Example 4.11 Show that

$$L = \left\{a^{n!} : n \geq 0\right\}$$

is not regular.

Given the opponent's choice for m, we pick as w the string $a^{m!}$ (unless the opponent picks $m < 3$, in which case we can use $a^{3!}$ as w). The various decompositions of w obviously differ only in the lengths of the substrings. Suppose the opponent picks y such that

$$|y| = k \leq m.$$

We then look at xz which has length $m! - k$. This string is in L only if there exists a j such that

$$m! - k = j!$$

But this is impossible, since for $m > 2$ and $k \leq m$ we have

$$m! > m! - k > (m-1)!$$

Therefore, the language is not regular. ■

In some cases, closure properties can be used to relate a given problem to one we have already classified. This may be much simpler than a direct application of the pumping lemma.

Example 4.12 Show that the language

$$L = \{a^n b^k c^{n+k} : n \geq 0, k \geq 0\}$$

is not regular.

It is not difficult to apply the pumping lemma directly, but it is even easier to use closure under homomorphism. Take

$$h(a) = a, h(b) = a, h(c) = c$$

then

$$\begin{aligned} h(L) &= \{a^{n+k} c^{n+k} : n + k \geq 0\} \\ &= \{a^i c^i : i \geq 0\}, \end{aligned}$$

but we know this language is not regular; therefore L cannot be regular either. ■

Example 4.13 Show that the language

$$L = \{a^n b^l : n \neq l\}$$

is not regular.

Here we need a bit of ingenuity to apply the pumping lemma directly. Choosing a string with $n = l+1$ or $n = l+2$ will not do, since our opponent can always choose a decomposition that will make it impossible to pump the string out of the language (that is, pump it so that it has an equal number of a's and b's). We must be more inventive. Let us take $n = m!$ and $l = (m+1)!$. If the opponent now chooses a y (by necessity consisting of all a's) of length $k < n$, we pump i times to generate a string with $m! + (i-1)\,k$ a's. We can get a contradiction of the pumping lemma if we can pick i such that

$$m! + (i-1)\,k = (m+1)!$$

This is always possible since

$$i = 1 + \frac{m\;m!}{k}$$

and $k \leq m$. The right side is therefore an integer, and we have succeeded in violating the conditions of the pumping lemma.

However, there is a much more elegant way of solving this problem. Suppose L were regular. Then, by Theorem 4.1, $\overline{L}$ and the language

$$L_1 = \overline{L} \cap L\,(a^*b^*)$$

would also be regular. But $L_1 = \{a^n b^n : n \geq 0\}$, which we have already classified as nonregular. Consequently, L cannot be regular.

The pumping lemma is difficult for several reasons. Its statement is complicated, and it is easy to go astray in applying it. But even if we master the technique, it may still be hard to see exactly how to use it. The pumping lemma is like a game with complicated rules. Knowledge of the rules is essential, but that alone is not enough to play a good game. You also need a good strategy to win. If you can apply the pumping lemma correctly to some of the more difficult cases in this book, you are to be congratulated.

Exercises

1. Prove the following version of the pumping lemma. If L is regular, then there is an m such that, every $w \in L$ of length greater than m can be decomposed as

$$w = xyz,$$

with

$$|yz| \leq m,$$
$$|y| \geq 1,$$

such that xy^iz is in L for all i.

2. Prove the following generalization of the pumping lemma, which includes Theorem 4.8 as well as Exercise 1 as special cases.

   If L is regular, then there exists an m, such that the following holds for every sufficiently long $w \in L$ and every one of its decompositions $w = u_1vu_2$, with $u_1, u_2 \in \Sigma^*$, $|v| \geq m$. The middle string v can be written as $v = xyz$, with $|xy| \leq m$, $|y| \geq 1$, such that $u_1xy^izu_2 \in L$ for all $i = 0, 1, 2, ...$ ●

3. Show that the language $L = \{w : n_a(w) = n_b(w)\}$ is not regular. Is L^* regular?

4. Prove that the following languages are not regular.

   (a) $L = \{a^nb^la^k : k \geq n + l\}$ ●

   (b) $L = \{a^nb^la^k : k \neq n + l\}$

   (c) $L = \{a^nb^la^k : n = l \text{ or } l \neq k\}$

   (d) $L = \{a^nb^l : n \leq l\}$

   (e) $L = \{w : n_a(w) \neq n_b(w)\}$ ●

   (f) $L = \{ww : w \in \{a,b\}^*\}$

   (g) $L = \{wwww^R : w \in \{a,b\}^*\}$

5. Determine if the following languages on $\Sigma = \{a\}$ are regular.

   (a) $L = \{a^n : n \geq 2, n \text{ is a prime number}\}$ ●

   [illegible] Fact [illegible] (b) $L = \{a^n : n \text{ is not a prime number}\}$

   [illegible]

   (c) $L = \{a^n : n = k^2 \text{ for some } k \geq 0\}$ $\longrightarrow a^{k^2}$ L a^{n^2}

   (d) $L = \{a^n : n = 2^k \text{ for some } k \geq 0\}$

   (e) $L = \{a^n : n \text{ is the product of two prime numbers}\}$

   (f) $L = \{a^n : n \text{ is either prime or the product of two or more prime numbers}\}$

6. Apply the pumping lemma directly to show the result in Example 4.12.

7. Show that the following language is not regular.

   $L = \{a^nb^k \; n > k-1\} \cup \{a^nb^k \; n < k-1\}$

   $n > k-1$
   $n < k-1$

$$L = \left\{a^nb^k : n > k\right\} \cup \left\{a^nb^k : n \neq k - 1\right\}$$

[illegible]

8. Prove or disprove the following statement. If L_1 and L_2 are nonregular languages, then $L_1 \cup L_2$ is also nonregular. ●

9. Consider the languages below. For each, make a conjecture whether or not it is regular. Then prove your conjecture.

(a) $L = \{a^n b^l a^k : n + l + k > 5\}$ ●

(b) $L = \{a^n b^l a^k : n > 5, l > 3, k \leq l\}$ ●

(c) $L = \{a^n b^l : n/l \text{ is an integer}\}$

(d) $L = \{a^n b^l : n + l \text{ is a prime number}\}$

(e) $L = \{a^n b^l : n \leq l \leq 2n\}$

(f) $L = \{a^n b^l : n \geq 100, l \leq 100\}$

(g) $L = \{a^n b^l : |n - l| = 2\}$.

10. Is the following language regular?

$$L = \{w_1 c w_2 : w_1, w_2 \in \{a,b\}^*, w_1 \neq w_2\}$$

11. Let L_1 and L_2 be regular languages. Is the language $L = \{w : w \in L_1, w^R \in L_2\}$ necessarily regular? ●

12. Apply the pigeonhole argument directly to the language in Example 4.8.

13. Are the following languages regular?

(a) $L = \{uww^R v : u, v, w \in \{a,b\}^+\}$ ●

★ (b) $L = \{uww^R v : u, v, w \in \{a,b\}^+, |u| \geq |v|\}$ ●

14. Is the following language regular?

$$L = \left\{ww^R v : v, w \in \{a,b\}^+\right\}$$

★★ **15.** Let P be an infinite but countable set, and associate with each $p \in P$ a language L_p. The smallest set containing every L_p is the union over the infinite set P; it will be denoted by $\cup_{p \in P} L_P$. Show by example that the family of regular languages is not closed under infinite union. ●

★ **16.** Consider the argument in Section 3.2 that the language associated with any generalized transition graph is regular. The language associated with such a graph is

$$L = \bigcup_{p \in P} L(r_p),$$

where P is the set of all walks through the graph and r_p is the expression associated with a walk p. The set of walks is generally infinite, so that in light of Exercise 15, it does not immediately follow that L is regular. Show that in this case, because of the special nature of P, the infinite union is regular.

* * ★17. Is the family of regular languages closed under infinite intersection? ●

* * 18. Suppose that we know that $L_1 \cup L_2$ and L_1 are regular. Can we conclude from this that L_2 is regular?

19. In the chain code language in Exercise 22, Section 3.1, let L be the set of all $w \in \{u, r, l, d\}^*$ that describe rectangles. Show that L is not a regular language.