CHAPTER 8

Regular and Nonregular Languages

The language a*b* is regular. The language $A^nB^n = \{a^nb^n : n \geq 0\}$ is not regular (intuitively because it is not possible, given some finite number of states, to count an arbitrary number of a's and then compare that count to the number of b's). The language $\{w \in \{a, b\}^* :$ every a is immediately followed by a b$\}$ is regular. The similar sounding language $\{w \in \{a, b\}^* :$ every a has a matching b somewhere and no b matches more than one a$\}$ is not regular (again because it is now necessary to count the a's and make sure that the number of b's is at least as great as the number of a's.)

Given a new language L, how can we know whether or not it is regular? In this chapter, we present a collection of techniques that can be used to answer that question.

8.1 How Many Regular Languages Are There?

First, we observe that there are *many* more nonregular languages than there are regular ones:

THEOREM 8.1 The Regular Languages are Countably Infinite

Theorem: There is a countably infinite number of regular languages.

Proof: We can lexicographically enumerate all the syntactically legal DFSMs with input alphabet Σ. Every regular language is accepted by at least one of them. So there cannot be more regular languages than there are DFSMs. Thus there are at most a countably infinite number of regular languages. There is not a one-to-one relationship between regular languages and DFSMs since there is an infinite number of machines that accept any given language. But the number of regular languages is infinite because it includes the following infinite set of languages:

$$\{a\}, \{aa\}, \{aaa\}, \{aaaa\}, \{aaaaa\}, \{aaaaaa\}, \ldots$$

But, by Theorem 2.3, there is an uncountably infinite number of languages over any nonempty alphabet So there are many more nonregular languages than there are regular ones.

8.2 Showing That a Language Is Regular

But many languages *are* regular. How can we know which ones? We start with the simplest cases.

THEOREM 8.2 The Finite Languages

Theorem: Every finite language is regular.

Proof: If L is the empty set, then it is defined by the regular expression $\emptyset$ and so is regular. If it is any finite language composed of the strings $s_1, s_2, \ldots s_n$ for some positive integer n, then it is defined by the regular expression:

$$s_1 \cup s_2 \cup \cdots \cup s_n$$

So it too is regular.

EXAMPLE 8.1 The Intersection of Two Infinite Languages

Let $L = L_1 \cap L_2$, where $L_1 = \{a^n b^n : n \geq 0\}$ and $L_2 = \{b^n a^n : n \geq 0\}$. As we will soon be able to prove, neither L_1 nor L_2 is regular. But L is. $L = \{\varepsilon\}$, which is finite.

EXAMPLE 8.2 A Finite Language We May Not Be Able to Write Down

Let $L = \{w \in \{0 - 9\}^* : w$ is the social security number of a living US resident$\}$. L is regular because it is finite. It doesn't matter that no individual or organization happens, at any given instant, to know what strings are in L.

Note, however, that although the language in Example 8.2 is formally regular, the techniques that we have described for recognizing regular languages would not be very useful in building a program to check for a valid social security number. Regular expressions are most useful when the elements of L match one or more patterns. FSMs are most useful when the elements of L share some simple structural properties. Other techniques, like hash tables, are better suited to handling finite languages whose elements are chosen by our world, rather than by rule.

EXAMPLE 8.3 Santa Clause, God, and the History of the Americas

Let:

- $L_1 = \{w \in \{0 - 9\}^* : w$ is the social security number of the current US president$\}$.
- $L_2 = \{1$ if Santa Claus exists and 0 otherwise$\}$.
- $L_3 = \{1$ if God exists and 0 otherwise$\}$.
- $L_4 = \{1$ if there were people in North America more than 10,000 years ago and 0 otherwise$\}$.
- $L_5 = \{1$ if there were people in North America more than 15,000 years ago and 0 otherwise$\}$.
- $L_6 = \{w \in \{0 - 9\}^+ : w$ is the decimal representation, without leading 0's, of a prime Fermat number$\}$.

L_1 is clearly finite, and thus regular. There exists a simple FSM to accept it, even though none of us happens to know what that FSM is. L_2 and L_3 are perhaps a little less clear, but that is because the meanings of "Santa Claus" and "God" are less clear. Pick a definition for either of them. Then something that satisfies that definition either does or does not exist. So either the simple FSM that accepts $\{0\}$ and nothing else or the simple FSM that accepts $\{1\}$ and nothing else accepts L_2. And one of them (possibly the same one, possibly the other one) accepts L_3. L_4 is clear. It is the set $\{1\}$. L_5 is also finite, and thus regular. Either there were people in North America by 15,000 years ago or there were not, although the currently available fossil evidence 💻 is unclear as to which. So we (collectively) just don't know yet which machine to build. L_6 is similar, although this time what is lacking is mathematics, as opposed to fossils. Recall from Section 4.1 that the Fermat numbers are defined by

$$F_n = 2^{2^n} + 1, n \geq 0.$$

The first five elements of F_n are $\{3, 5, 17, 257, 65{,}537\}$. All of them are prime. It appears likely 💻 that no other Fermat numbers are prime. If that is true, then L_6 is finite and thus regular. If it turns out that the set of Fermat numbers is infinite, then it is almost surely not regular.

> Not every regular language is computationally tractable. Consider the Towers of Hanoi language. (P. 2)

But, of course, most interesting regular languages are infinite. So far, we've developed four techniques for showing that a (finite or infinite) language L is regular:

- Exhibit a regular expression for L.
- Exhibit an FSM for L.

- Show that the number of equivalence classes of $\approx_L$ is finite.
- Exhibit a regular grammar for L.

8.3 Some Important Closure Properties of Regular Languages

We now consider one final technique, which allows us, when analyzing complex languages, to exploit the other techniques as subroutines. The regular languages are closed under many common and useful operations. So, if we wish to show that some language L is regular and we can show that L can be constructed from other regular languages using those operations, then L must also be regular.

THEOREM 8.3 Closure under Union, Concatenation and Kleene Star

Theorem: The regular languages are closed under union, concatenation, and Kleene star.

Proof: By the same constructions that were used in the proof of Kleene's theorem.

THEOREM 8.4 Closure under Complement, Intersection, Difference, Reverse and Letter Substitution

Theorem: The regular languages are closed under complement, intersection, difference, reverse, and letter substitution.

Proof:

- The regular languages are closed under complement. If L_1 is regular, then there exists a DFSM $M_1 = (K, \Sigma, \delta, s, A)$ that accepts it. The DFSM $M_2 = (K, \Sigma, \delta, s, K - A)$, namely M_1 with accepting and nonaccepting states swapped, accepts $\neg(L(M_1))$ because it rejects all strings that M_1 accepts and rejects all strings that M_1 accepts.

  Given an arbitrary (possibly nondeterministic) FSM $M_1 = (K_1, \Sigma, \Delta_1, s_1, A_1)$, we can construct a DFSM $M_2 = (K_2, \Sigma, \delta_2, s_2, A_2)$ such that $L(M_2) = \neg(L(M_1))$. We do so as follows: From M_1, construct an equivalent deterministic FSM $M' = (K_{M'}, \Sigma, \delta_{M'}, s_{M'}, A_{M'})$, using the algorithm *ndfsmtodfsm*, presented in the proof of Theorem 5.3. (If M_1 is already deterministic, $M' = M_1$.) M' must be stated completely, so if it is described with an implied dead state, add the dead state and all required transitions to it. Begin building M_2 by setting it equal to M'. Then swap the accepting and the nonaccepting states. So $M_2 = (K_{M'}, \Sigma, \delta_{M'}, s_{M'}, K_{M'} - A_{M'})$.
- The regular languages are closed under intersection. We note that:

$$L(M_1) \cap L(M_2) = \neg(\neg L(M_1) \cup \neg L(M_2)).$$

  We have already shown that the regular languages are closed under both complement and union. Thus they are also closed under intersection.

It is also possible to prove this claim by construction of an FSM that accepts $L(M_1) \cap L(M_2)$. We leave that proof as an exercise.

- The regular languages are closed under set difference (subtraction). We note that:

$$L(M_1) - L(M_2) = L(M_1) \cap \neg L(M_2).$$

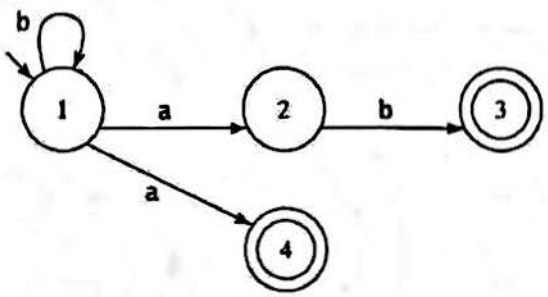
We have already shown that the regular languages are closed under both complement and intersection. Thus they are also closed under set difference.

This claim too can also be proved by construction, which we leave as an exercise.

- The regular languages are closed under reverse. Recall that $L^R = \{w \in \Sigma^* : w = x^R \text{ for some } x \in L\}$. We leave the proof of this as an exercise.
- The regular languages are closed under letter substitution, defined as follows: Consider any two alphabets, Σ_1 and Σ_2. Let *sub* be any function from Σ_1 to Σ_2^*. Then *letsub* is a letter substitution function from L_1 to L_2 iff $letsub(L_1) = \{w \in \Sigma_2^* : \exists y \in L_1 \ (w = y \text{ except that every character } c \text{ of } y \text{ has been replaced by } sub(c))\}$. For example, suppose that $\Sigma_1 = \{a, b\}$, $\Sigma_2 = \{0, 1\}$, $sub(a) = 0$, and $sub(b) = 11$. Then $letsub(\{a^n b^n : n \geq 0\}) = \{0^n 1^{2n} : n \geq 0\}$. We leave the proof that the regular languages are closed under letter substitution as an exercise.

EXAMPLE 8.4 Closure Under Complement

Consider the following NDFSM M =



If we use the algorithm that we just described to convert M to a new machine M' that accepts $\neg L(M)$, the last step is to swap the accepting and the nonaccepting states. A quick look at M makes it clear why it is necessary first to make M deterministic and then to complete it by adding the dead state. M accepts the input a in state 4. If we simply swapped accepting and nonaccepting states, without

making the other changes, M' would also accept a. It would do so in state 2. The problem is that M is nondeterministic, and has one path along which a is accepted and one along which it is rejected.

To see why it is necessary to add the dead state, consider the input string aba. M rejects it since the path from state 3 dies when M attempts to read the final a and the path from state 4 dies when it attempts to read the b. But, if we don't add the dead state, M' will also reject it since, in it too, both paths will die.

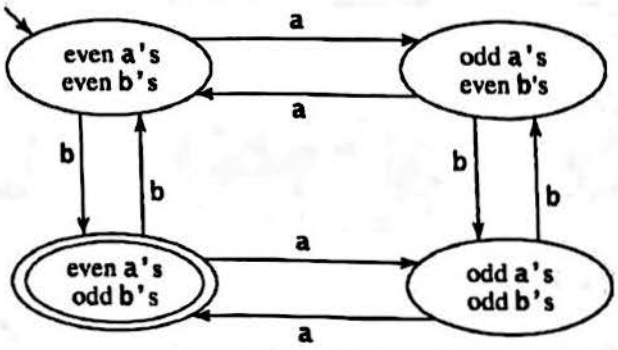
The closure theorems that we have now proved make it easy to take a divide-and-conquer approach to showing that a language is regular. They also let us reuse proofs and constructions that we've already done.

EXAMPLE 8.5 The Divide-and-Conquer Approach

Let $L = \{w \in \{a, b\}^* : w$ contains an even number of a's and an odd number of b's and all a's come in runs of three$\}$. L is regular because it is the intersection of two regular languages. $L = L_1 \cap L_2$, where:

- $L_1 = \{w \in \{a, b\}^* : w$ contains an even number of a's and an odd number of b's $\}$, and
- $L_2 = \{w \in \{a, b\}^* :$ all a's come in runs of three$\}$.

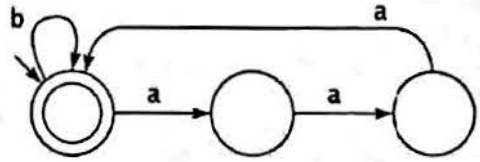
We already know that L_1 is regular, since we showed an FSM that accepts it in Example 5.9:



Of course, we could start with this machine and modify it so that it accepts L. But an easier way is exploit a divide-and-conquer approach. We'll just use the machine we have and then build a second simple machine, this one to accept L_2.

EXAMPLE 8.5 *(Continued)*

Then we can prove that L is regular by exploiting the fact that the regular languages are closed under intersection. The following machine accepts L_2:



The closure theorems are powerful, but they say only what they say. We have stated each of the closure theorems in as strong a form as possible. Any similar claims that are not implied by the theorems as we have stated them are almost certainly false, which can usually be shown easily by finding a simple counterexample.

EXAMPLE 8.6 What the Closure Theorem for Union Does Not Say

The closure theorem for union says that:

if L_1 and L_2 are regular *then* $L = L_1 \cup L_2$ is regular.

The theorem says nothing, for example, about what happens if L is regular. Does that mean that L_1 and L_2 are also? The answer is maybe. We know that a^+ is regular. We will consider two cases for L_1 and L_2. First, let them be:

$$a^+ = \{a^p : p > 0 \text{ and } p \text{ is prime}\} \cup \{a^p : p > 0 \text{ and } p \text{ is not prime}\}.$$

$$a^+ = \qquad L_1 \qquad \cup \qquad L_2.$$

As we will see in the next section, neither L_1 nor L_2 is regular. But now consider:

$$a^+ = \{a^p : p > 0 \text{ and } p \text{ is even}\} \cup \{a^p : p > 0 \text{ and } p \text{ is odd}\}.$$

$$a^+ = \qquad L_1 \qquad \cup \qquad L_2.$$

In this case, both L_1 and L_2 are regular.

EXAMPLE 8.7 What the Closure Theorem for Concatenation Does Not Say

The closure theorem for concatenation says that:

if L_1 and L_2 are regular *then* $L = L_1L_2$ is regular.

But the theorem says nothing, for example, about what happens if L_2 is not regular. Does that mean that L isn't regular either? Again, the answer is maybe. We first consider the following example:

$$\{aba^nb^n : n \geq 0\} = \{ab\}\{a^nb^n : n \geq 0\}.$$

$$L \qquad = \quad L_1 \qquad L_2.$$

As we'll see in the next section, L_2 is not regular. And, in this case, neither is L. But now consider:

$$\{aaa^*\} = \{a^*\}\{a^p : p \text{ is prime}\}.$$

$$L = L_1 \quad L_2.$$

While again L_2 is not regular, now L is.

8.4 Showing That a Language is Not Regular

We can show that a language is regular by exhibiting a regular expression or an FSM or a finite list of the equivalence classes of $\approx_L$ or a regular grammar, or by using the closure properties that we have proved hold for the regular languages. But how shall we show that a language is not regular? In other words, how can we show that none of those descriptions exists for it? It is not sufficient to argue that we tried to find one of them and failed. Perhaps we didn't look in the right place. We need a technique that does not rely on our cleverness (or lack of it).

What we can do is to make use of the following observation about the regular languages: Every regular language L can be accepted by an FSM M with a finite number of states. If L is infinite, then there must be at least one loop in M. All sufficiently long strings in L must be characterized by one or more repeating patterns, corresponding to the substrings that drive M through its loops. It is also true that, if L is infinite, then any regular expression that describes L must contain at least one Kleene star, but we will focus here on FSMs.

To help us visualize the rest of this discussion, consider the FSM M_{LOOP}, shown in Figure 8.1 (a). M_{LOOP} has 5 states. It can accept an infinite number of strings. But the longest one that it can accept without going through any loops has length 4. Now consider the slightly different FSM M_ε, shown in Figure 8.1 (b). M_ε also has 5 states and one loop. But it accepts only one string, aab. The only string that can drive M_ε through its loop is ε. No matter how many times M_ε goes through the loop, it cannot accept any longer strings.

To simplify the following discussion, we will consider only DFSMs, which have no ε-transitions. Each transition step that a DFSM takes corresponds to exactly one character in its input. Since any language that can be accepted by an NDFSM can also be accepted by a DFSM, this restriction will not affect our conclusions.

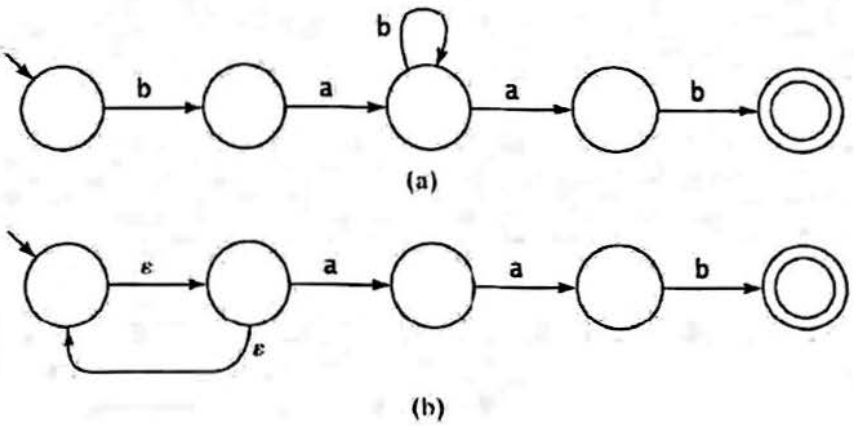


FIGURE 8.1 What is the longest string that a 5-state FSM can accept?

THEOREM 8.5 Long Strings Force Repeated States

Theorem: Let $M = (K, \Sigma, \delta, s, A)$ be any DFSM. If M accepts any string of length $|K|$ or greater, then that string will force M to visit some state more than once (thus traversing at least one loop).

Proof: M must start in one of its states. Each time it reads an input character, it visits some state. So, in processing a string of length n, M creates a total of $n + 1$ state visits (the initial one plus one for each character it reads). If $n + 1 > |K|$, then, by the pigeonhole principle, some state must get more than one visit. So, if $n \geq |K|$, then M must visit at least one state more than once.

Let $M = (K, \Sigma, \delta, s, A)$ be any DFSM. Suppose that there exists some "long" string w (i.e., $|w| \geq |K|$) such that $w \in L(M)$. Then M must go through at least one loop when it reads w. So there is some substring y of w that drove M through at least one loop. Suppose we excise y from w. The resulting string must also be in $L(M)$ since M can accept it just as it accepts w but skipping one pass through one loop. Further, suppose that we splice in one or more extra copies of y, immediately adjacent to the original one. All the resulting strings must also be in $L(M)$ since M can accept them by going through its loop one or more additional times. Using an analogy with a pump, we'll say that we can ***pump*** y out once or in an arbitrary number of times and the resulting string must still be in L.

To make this concrete, let's look again at M_{LOOP}, which accepts, for example, the string babbab. babbab is "long" since its length is 6 and $|K| = 5$. The second b drove M_{LOOP} through its loop. Call the string (in this case b) that drove M_{LOOP} through its loop y. We can pump it out, producing babab, which is also accepted by M_{LOOP}. Or we can pump in as many copies of b as we like, generating such strings as babbbab, babbbbbab, and so forth. M_{LOOP} also accepts all of them. Returning to the original string babbab, the third b also drove M_{LOOP} through its loop. We could also pump it (in or out) and get a similar result.

This property of FSMs, and the languages that they can accept, is the basis for a powerful tool for showing that a language is not regular. If a language contains even one long (to be defined precisely below) string that cannot be pumped in the fashion that we have just described, then it is not accepted by any FSM and so is not regular. We formalize this idea, as the Pumping Theorem, in the next section.

8.4.1 The Pumping Theorem for Regular Languages

THEOREM 8.6 The Pumping Theorem for Regular Languages

Theorem: If L is a regular language, then:

$$\exists k \geq 1 \, (\forall \text{strings } w \in L, \text{where } |w| \geq k \, (\exists x, y, z \, (w = xyz, |xy| \leq k, y \neq \varepsilon, \text{and } \forall q \geq 0 \, (xy^q z \in L)))).$$

Proof: The proof is the argument that we gave above: If L is regular then it is accepted by some DFSM $M = (K, \Sigma, \delta, s, A)$. Let k be $|K|$. Let w be any string in L of length k or greater. By Theorem 8.5, to accept w, M must traverse some loop at least once. We can carve w up and assign the name y to the first substring to drive M through a loop. Then x is the part of w that precedes y and z is the part of w that follows y. We show that each of the last three conditions must then hold:

- $|xy| \leq k$: M must not only traverse a loop eventually when reading w, it must do so for the first time by at least the time it has read k characters. It can read $k - 1$ characters without revisiting any states. But the k^{th} character must, if no earlier character already has, take M to a state it has visited before. Whatever character does that is the last in one pass through some loop.
- $y \neq \varepsilon$: Since M is deterministic, there are no loops that can be traversed by ε.
- $\forall q \geq 0\ (xy^q z \in L)$: y can be pumped out once (which is what happens if $q = 0$) or in any number of times (which happens if q is greater than 1) and the resulting string must be in L since it will be accepted by M. It is possible that we could chop y out more than once and still generate a string in L, but without knowing how much longer w is than k, we don't know any more than that it can be pumped out once.

The Pumping Theorem tells us something that is true of every regular language. Generally, if we already know that a language is regular, we won't particularly care about what the Pumping Theorem tells us about it. But suppose that we are interested in some language L and we want to know whether or not it is regular. If we could show that the claims made in the Pumping Theorem are not true of L, then we would know that L is not regular. It is in arguments such as this that we will find the Pumping Theorem very useful. In particular, we will use it to construct ***proofs by contradiction***. We will say, "If L were regular, then it would possess certain properties. But it does not possess those properties. Therefore, it is not regular."

EXAMPLE 8.8 A^nB^n is not Regular

Let L be $A^nB^n = \{a^n b^n : n \geq 0\}$. We can use the Pumping Theorem to show that L is not regular. If it were, then there would exist some k such that any string w, where $|w| \geq k$, must satisfy the conditions of the theorem. We show one string w that does not. Let $w = a^k b^k$. Since $|w| = 2k$, w is long enough and it is in L, so it must satisfy the conditions of the Pumping Theorem. So there must exist x, y, and z, such that $w = xyz$, $|xy| \leq k$, $y \neq \varepsilon$, and $\forall q \geq 0\ (xy^q z \in L)$. But we show that no such x, y, and z exist. Since we must guarantee that $|xy| \leq k$, y must occur within the first k characters and so $y = a^p$ for some p. Since we must guarantee that $y \neq \varepsilon$, p must be greater than 0. Let $q = 2$. (In other words, we pump in one extra copy of y.) The resulting string is $a^{k+p}b^k$. The last condition of the Pumping Theorem states that this string must be in L, but it is not since it has more a's than b's. Thus there exists at least one long string in L that fails to satisfy the conditions of the Pumping Theorem. So $L = A^nB^n$ is not regular.

The Pumping Theorem is a powerful tool for showing that a language is not regular. But, as with any tool, using it effectively requires some skill. To see how the theorem can be used, let's state it again in its most general terms:

For any language L, if L is regular, then every "long" string in L is pumpable.

So, to show that L is not regular, it suffices to find a single long string w that is in L but is not pumpable. To show that a string is not pumpable, we must show that there is no way to carve it up into x, y, and z in such a way that all three of the conditions of the theorem are met. It is not sufficient to pick a particular y and show that it doesn't work. (We focus on y since, once it has been chosen, everything to the left of it is x and everything to the right of it is z). We must show that there is *no* value for y that works. To do that, we consider all the logically possible classes of values for y (sometimes there is only one such class, but sometimes several must be considered). Then we show that each of them fails to satisfy at least one of the three conditions of the theorem. Generally we do that by assuming that y does satisfy the first two conditions, namely that it occurs within the first k characters and is not ε. Then we consider the third requirement, namely that, for all values of q, xy^qz is in L. To show that it is not possible to satisfy that requirement, it is sufficient to find a single value of q such that the resulting string is not in L. Typically, this can be done by setting q to 0 (thus pumping out once) or to 2 (pumping in once), although sometimes some other value of q must be considered.

In a nutshell then, to use the Pumping Theorem to show that a language L is not regular, we must:

1. Choose a string w, where $w \in L$ and $|w| \geq k$. Note that we do not know what k is; we know only that it exists. So we must state w in terms of k.
2. Divide the possibilities for y into a set of equivalence classes so that all strings in a class can be considered together.
3. For each such class of possible y values, where $|xy| \leq k$ and $y \neq \varepsilon$:
   Choose a value for q such that xy^qz is not in L.

In Example 8.8, y had to fall in the initial a region of w, so that was the only case that needed to be considered. But, had we made a less judicious choice for w, our proof would not have been so simple. Let's look at another proof, with a different w:

EXAMPLE 8.9 A Less Judicious Choice for w

Again let L be $A^nB^n = \{a^nb^n : n \geq 0\}$. If A^nB^n were regular, then there would exist some k such that any string w, where $|w| \geq k$, must satisfy the conditions of the theorem. Let $w = a^{\lceil k/2 \rceil}b^{\lceil k/2 \rceil}$. (We must use $\lceil k/2 \rceil$, i.e., the smallest integer greater than $k/2$, rather than truncating the division, since k might be odd.) Since $|w| \geq k$ and w is in L, w must satisfy the conditions of the Pumping Theorem. So, there must exist x, y, and z, such that $w = xyz$, $|xy| \leq k$, $y \neq \varepsilon$, and $\forall q \geq 0\ (xy^qz \in L)$. We show that no such x, y, and z exist. This time, if they did, y

could be almost anywhere in w (since all the Pumping Theorem requires is that it occur in the first k characters and there are only at most $k + 1$ characters). So we must consider three cases and show that, in all three, there is no y that satisfies all conditions of the Pumping Theorem. A useful way to describe the cases is to imagine w divided into two regions:

aaaaa.....aaaaaa | bbbbb.....bbbbbb
1 | 2

Now we see that y can fall:

- Exclusively in region 1: In this case, the proof is identical to the proof we did for Example 8.8.
- Exclusively in region 2: then $y = b^p$ for some p. Since $y \neq \epsilon$, p must be greater than 0. Let $q = 2$. The resulting string is $a^k b^{k-p}$. But this string is not in L, since it has more b's than a's.
- Straddling the boundary between regions 1 and 2: Then $y = a^p b^r$ for some non-zero p and r. Let $q = 2$. The resulting string will have interleaved a's and b's, and so is not in L.

There exists at least one long string in L that fails to satisfy the conditions of the Pumping Theorem. So $L = A^nB^n$ is not regular.

To make maximum use of the Pumping Theorem's requirement that y fall in the first k characters, it is often a good idea to choose a string w that is substantially longer than the k characters required by the theorem. In particular, if w can be chosen so that there is a uniform first region of length at least k, it may be possible to consider just a single case for where y can fall.

> The Pumping Theorem inspires poets 🖳, as we'll see in Chapter 10.

A^nB^n is a simple language that illustrates the kind of property that characterizes languages that aren't regular. It isn't of much practical importance, but it is typical of a family of languages, many of which are of more practical significance. In the next example, we consider Bal, the language of balanced parentheses. The structure of Bal is very similar to that of A^nB^n. Bal is important because most languages for describing arithmetic expressions, Boolean queries, and markup systems require balanced delimiters.

EXAMPLE 8.10 The Balanced Parenthesis Language is Not Regular

Let L be Bal $= \{w \in \{), (\}^* :$ the parentheses are balanced$\}$. If L were regular, then there would exist some k such that any string w, where $|w| \geq k$, must satisfy the conditions of the theorem. Bal contains complex strings like (())(()()). But it is

EXAMPLE 8.10 *(Continued)*

almost always easier to use the Pumping Theorem if we pick as simple a string as possible. So, let $w = (^k)^k$. Since $|w| = 2k$ and w is in L, w must satisfy the conditions of the Pumping Theorem. So there must exist x, y, and z, such that $w = xyz$, $|xy| \leq k$, $y \neq \varepsilon$, and $\forall q \geq 0\ (xy^q z \in L)$. But we show that no x, y, and z exist. Since $|xy| \leq k$, y must occur within the first k characters and so $y = (^p$ for some p). Since $y \neq \varepsilon$, p must be greater than 0. Let $q = 2$. (In other words, we pump in one extra copy of y.) The resulting string is $(^{k+p})^k$. The last condition of the Pumping Theorem states that this string must be in L, but it is not since it has more ('s than)'s. There exists at least one long string in L that fails to satisfy the conditions of the Pumping Theorem. So L = Bal is not regular.

EXAMPLE 8.11 The Even Palindrome Language is Not Regular

Let L be PalEven $= \{ww^R : w \in \{a, b\}^*\}$. PalEven is the language of even-length palindromes of a's and b's. We can use the Pumping Theorem to show that PalEven is not regular. If it were, then there would exist some k such that any string w, where $|w| \geq k$, must satisfy the conditions of the theorem. We show one string w that does not. (Note here that the variable w used in the definition of L is different from the variable w mentioned in the Pumping Theorem.) We will choose w so that we only have to consider one case for where y could fall. Let $w = a^k b^k b^k a^k$. Since $|w| = 4k$ and w is in L, w must satisfy the conditions of the Pumping Theorem. So there must exist x, y, and z, such that $w = xyz$, $|xy| \leq k$, $y \neq \varepsilon$, and $\forall q \geq 0\ (xy^q z \in L)$. Since $|xy| \leq k$, y must occur within the first k characters and so $y = a^p$ for some p. Since $y \neq \varepsilon$, p must be greater than 0. Let $q = 2$. The resulting string is $a^{k+p} b^k b^k a^k$. If p is odd, then this string is not in PalEven because all strings in PalEven have even length. If p is even then it is at least 2, so the first half of the string has more a's than the second half does, so it is not in PalEven. So L = PalEven is not regular.

The Pumping Theorem says that, for any language L, if L is regular, then all long strings in L must be pumpable. Our strategy in using it to show that a language L is not regular is to find *one* string that fails to meet that requirement. Often, there are many long strings that *are* pumpable. If we try to work with them, we will fail to derive the contradiction that we seek. In that case, we will know nothing about whether or not L is regular. To find a w that is not pumpable, think about what property of L is not checkable by an FSM and choose a w that exhibits that property. Consider again our last example. The thing that an FSM cannot do is to remember an arbitrarily long first half and check it against the second half. So we chose a w that would have forced it to do that. Suppose instead that we had let $w = a^k a^k$. It is in L and long enough. But y could be aa and we could pump it out or in and all the resulting strings would be in L.

So far, all of our Pumping Theorem proofs have set q to 2. But that is not always the thing to do. Sometimes it will be necessary to set it to 0. (In other words, we will pump y out).

EXAMPLE 8.12 The Language with More a's Than b's is Not Regular

Let $L = \{a^n b^m : n > m\}$. We can use the Pumping Theorem to show that L is not regular. If it were, then there would exist some k such that any string w, where $|w| \geq k$, must satisfy the conditions of the theorem. We show one string w that does not. Let $w = a^{k+1}b^k$. Since $|w| = 2k + 1$ and w is in L, w must satisfy the conditions of the Pumping Theorem. So there must exist x, y, and z, such that $w = xyz$, $|xy| \leq k$, $y \neq \varepsilon$, and $\forall q \geq 0\ (xy^q z \in L)$. Since $|xy| \leq k$, y must occur within the first k characters and so $y = a^p$ for some p. Since $y \neq \varepsilon$, p must be greater than 0. There are already more a's than b's, as required by the definition of L. If we pump in, there will be even more a's and the resulting string will still be in L. But we can set q to 0 (and so pump out). The resulting string is then $a^{k+1-p}b^k$. Since $p > 0$, $k + 1 - p \leq k$, so the resulting string no longer has more a's than b's and so is not in L. There exists at least one long string in L that fails to satisfy the conditions of the Pumping Theorem. So L is not regular.

Notice that the proof that we just did depended on our having chosen a w that is just barely in L. It had exactly one more a than b. So y could be any string of up to k a's. If we pumped in extra copies of y, we would have gotten strings that were still in L. But if we pumped out even a single a, we got a string that was not in L, and so we were able to complete the proof.. Suppose, though, that we had chosen $w = a^{2k}b^k$. Again, pumping in results in strings in L. And now, if y were simply a, we could pump out and get a string that was still in L. So that proof attempt fails. In general, it is a good idea to choose a w that barely meets the requirements for L. That makes it more likely that pumping will create a string that is not in L.

Sometimes values of q other than 0 or 2 may also be required.

EXAMPLE 8.13 The Prime Number of a's Language is Not Regular

Let L be $\text{Prime}_a = \{a^n : n \text{ is prime}\}$. We can use the Pumping Theorem to show that L is not regular. If it were, then there would exist some k such that any string w, where $|w| \geq k$, must satisfy the conditions of the theorem. We show one string w that does not. Let $w = a^j$, where j is the smallest prime number greater than $k + 1$. Since $|w| > k$, w must satisfy the conditions of the Pumping Theorem. So there must exist x, y, and z, such that $w = xyz$, $|xy| \leq k$ and $y \neq \varepsilon$. $y = a^p$ for some p. The Pumping Theorem further requires that $\forall q \geq 0\ (xy^q z \in L)$. So, $\forall q \geq 0$ ($a^{|x|+|z|+q|y|}$ must be in L). That means that $|x| + |z| + q \cdot |y|$ must be prime.

EXAMPLE 8.13 *(Continued)*

But suppose that $q = |x| + |z|$. Then:

$$|x| + |z| + q \cdot |y| = |x| + |z| + (|x| + |z|) \cdot y$$
$$= (|x| + |z|) \cdot (1 + |y|),$$

which is composite (non-prime) if both factors are greater than 1. $(|x| + |z|) > 1$ because $|w| > k + 1$, and $|y| \leq k$. $(1 + |y|) > 1$ because $|y| > 0$. So, for at least that one value of q, the resulting string is not in L. So L is not regular.

When we do a Pumping Theorem proof that a language L is not regular, we have two choices to make: a value for w and a value for q. As we have just seen, there are some useful heuristics that can guide our choices:

- To choose w:
  - Choose a w that is in the part of L that makes it not regular.
  - Choose a w that is only barely in L.
  - Choose a w with as homogeneous as possible an initial region of length at least k.
- To choose q:
  - Try letting q be either 0 or 2.
  - If that doesn't work, analyze L to see if there is some other specific value that will work.

8.4.2 Using Closure Properties

Sometimes the easiest way to prove that a language L is not regular is to use the closure theorems for regular languages, either alone or in conjunction with the Pumping Theorem. The fact that the regular languages are closed under intersection is particularly useful.

EXAMPLE 8.14 Using Intersection to Force Order Constraints

Let $L = \{w \in \{a, b\}^* : \#_a(w) = \#_b(w)\}$. If L were regular, then $L' = L \cap a^*b^*$ would also be regular. But $L' = \{a^n b^n : n \geq 0\}$, which we have already shown is not regular. So L isn't either.

EXAMPLE 8.15 Using Closure Under Complement

Let $L = \{a^i b^j : i, j \geq 0 \text{ and } i \neq j\}$. It seems unlikely that L is regular since any machine to accept it would have to count the a's. It is possible to use the Pumping

Theorem to prove that L is not regular but it is not easy to see how. Suppose, for example, that we let $w = a^{k+1}b^k$. But then y could be aa and it would pump since $a^{k-1}b^k$ is in L, and so is $a^{k+1+2(q-1)}b^k$, for all nonnegative values of q.

Instead, let $w = a^k b^{k+k!}$. Then $y = a^p$ for some nonzero p. Let $q = (k!/p) + 1$ (in other words, pump in $(k!/p)$ times). Note that $(k!/p)$ must be an integer because $p < k$. The number of a's in the resulting string is $k + (k!/p)p = k + k!$. So the resulting string is $a^{k+k!}b^{k+k!}$, which has equal numbers of a's and b's and so is not in L.

The closure theorems provide an easier way. We observe that if L were regular, then $\neg L$ would also be regular, since the regular languages are closed under complement. $\neg L = \{a^n b^n : n \geq 0\} \cup \{$ strings of a's and b's that do not have all a's in front of all b's$\}$. If $\neg L$ is regular, then $\neg L \cap a^*b^*$ must also be regular. But $\neg L \cap a^*b^* = \{a^n b^n : n \geq 0\}$, which we have already shown is not regular. So neither is $\neg L$ or L.

Sometimes, using the closure theorems is more than a convenience. There are languages that are not regular but that do meet all the conditions of the Pumping Theorem. The Pumping Theorem alone is insufficient to prove that those languages are not regular, but it may be possible to complete a proof by exploiting the closure properties of the regular languages.

EXAMPLE 8.16 Sometimes We Must Use the Closure Theorems

Let $L = \{a^i b^j c^k : i, j, k \geq 0$ and (if $i = 1$ then $j = k)\}$. Every string of length at least 1 that is in L is pumpable. It is easier to see this if we rewrite the final condition as $(i \neq 1)$ or $(j = k)$. Then we observe:

- If $i = 0$ then: If $j \neq 0$, let y be b; otherwise, let y be c. Pump in or out. Then i will still be 0 and thus not equal to 1, so the resulting string is in L.
- If $i = 1$ then: Let y be a. Pump in or out. Then i will no longer equal 1, so the resulting string is in L.
- If $i = 2$ then: Let y be aa. Pump in or out. Then i cannot equal 1, so the resulting string is in L.
- If $i > 2$ then: Let y be a. Pump out once or in any number of times. Then i cannot equal 1, so the resulting string is in L.

But L is not regular. One way to prove this is to use the fact that the regular languages are closed under intersection. So, if L were regular, then $L' = L \cap ab^*c^* = \{ab^j c^k : j, k \geq 0$ and $j = k\}$ would also be regular. But it is not, which we can show using the Pumping Theorem. Let $w = ab^k c^k$. Then y must occur in the first k characters of w. If y includes the initial a, pump in once. The resulting string is not in L' because it contains more than one a. If y does not include the initial a, then it must be b^p, where $0 < p < k$. Pump in once. The resulting string is not in L' because it contains more b's than c's. Since L' is not regular, neither is L.

EXAMPLE 8.16 *(Continued)*

Another way to show that L is not regular is to use the fact that the regular languages are closed under reverse. $L^R = \{c^k\ b^j\ a^i : i, j, k \geq 0$ and (if $i = 1$ then $j = k)\}$. If L were regular then L^R would also be regular. But it is not, which we can show using the Pumping Theorem. Let $w =$ $c^k\ b^k\ a$. y must occur in the first k characters of w, so $y = c^p$, where $0 < p \leq k$. Set q to 0. The resulting string contains a single a, so the number of b's and c's must be equal for it to be in L^R. But there are fewer c's than b's. So the resulting string is not in L^R. L^R is not regular. Since L^R is not regular, neither is L.

8.5 Exploiting Problem-Specific Knowledge

Given some new language L, the theory that we have been describing provides the skeleton for an analysis of L. If L is simple, that may be enough. But if L is based on a real problem, any analysis of it will also depend on knowledge of the task domain. We got a hint of this in Example 8.13, where we had to use some knowledge about numbers and algebra. Other problems also require mathematical facts.

EXAMPLE 8.17 The Octal Representation of a Number Divisible by 7

Let $L = \{w \in \{0, 1, 2, 3, 4, 5, 6, 7\}^* : w$ is the octal representation of a nonnegative integer that is divisible by $7\}$. The first several strings in L are: 0, 7, 16, 25, 34, 43, 52, and 61. Is L regular? Yes, because there is a simple, 7-state DFSM M that accepts L. The structure of M takes advantage of the fact that w is in L iff the sum of its digits, viewed as numbers, is divisible by 7. So the states of M correspond to the modulo 7 sum of the digits so far. We omit the details.

Sometimes L corresponds to a problem from a domain other than mathematics, in which case facts from that domain will be important.

EXAMPLE 8.18 A Music Language

Let $\Sigma = \{$𝅝, 𝅗𝅥, ♩, ♪, 𝅘𝅥𝅯, 𝅘𝅥𝅰$\}$. Let $L = \{w : w$ represents a song written in 4/4 time$\}$. L is regular. It can be accepted by an FSM that checks for 4 beats between measure bars, where 𝅝 counts as 4, 𝅗𝅥 counts as 2, ♩ counts as 1, ♪ counts as $^1/_2$, 𝅘𝅥𝅯 counts as $^1/_4$, and 𝅘𝅥𝅰 counts as 1/8.

> Other techniques described in this book can also be applied to the language of music. (N.1)

EXAMPLE 8.23 The Function *mix*

Define $mix(L) = \{w : \exists x, y, z (x \in L, x = yz, |y| = |z|, w = yz^R)\}$. In other words, $mix(L)$ contains exactly those strings that can be formed by taking some even length string in L and reversing its second half. Let's look at *mix* applied to some languages:

L	$mix(L)$
$\emptyset$	$\emptyset$
$(a \cup b)^*$	$((a \cup b)(a \cup b))^*$
$(ab)^*$	$\{(ab)^{2n+1} : n \geq 0\} \cup \{(ab)^n(ba)^n : n \geq 0\}$
$(ab)^*a(ab)^*$	$\emptyset$

The regular languages are closed under *maxstring*. They are not closed under *mix*. We leave the proof of these claims as an exercise.

Exercises

1. For each of the following languages L, state whether L is regular or not and prove your answer:
   a. $\{a^i b^j : i, j \geq 0 \text{ and } i + j = 5\}$.
   b. $\{a^i b^j : i, j \geq 0 \text{ and } i - j = 5\}$.
   c. $\{a^i b^j : i, j \geq 0 \text{ and } |i - j| \equiv_5 0\}$.
   d. $\{w \in \{0, 1, \#\}^* : w = x \# y, \text{ where } x, y \in \{0, 1\}^* \text{ and } |x| \cdot |y| \equiv_5 0\}$.
   e. $\{a^i b^j : 0 \leq i < j < 2000\}$.
   f. $\{w \in \{Y, N\}^* : w \text{ contains at least two Y's and at most two N's}\}$.
   g. $\{w = xy : x, y \in \{a, b\}^* \text{ and } |x| = |y| \text{ and } \#_a(x) \geq \#_a(y)\}$.
   h. $\{w = xyzy^R x : x, y, z \in \{a, b\}^*\}$.
   i. $\{w = xyzy : x, y, z \in \{0, 1\}^+\}$.
   j. $\{w \in \{0, 1\}^* : \#_0(w) \neq \#_1(w)\}$.
   k. $\{w \in \{a, b\}^* : w = w^R\}$.
   l. $\{w \in \{a, b\}^* : \exists x \in \{a, b\}^+ (w = x x^R x)\}$.
   m. $\{w \in \{a, b\}^* : \text{the number of occurrences of the substring ab equals the number of occurrences of the substring ba}\}$.
   n. $\{w \in \{a, b\}^* : w \text{ contains exactly two more b's than a's}\}$.
   o. $\{w \in \{a, b\}^* : w = xyz, |x| = |y| = |z|, \text{ and } z = x \text{ with every a replaced by b and every b replaced by a}\}$. Example: abbbabbaa $\in L$, with $x =$ abb, $y =$ bab, and $z =$ baa.
   p. $\{w : w \in \{a - z\}^* \text{ and the letters of } w \text{ appear in reverse alphabetical order}\}$. For example, spoonfeed $\in L$.

q. $\{w: w \in \{a - z\}^*$ every letter in w appears at least twice$\}$. For example, unprosperousness $\in L$.

r. $\{w : w$ is the decimal encoding of a natural number in which the digits appear in a non-decreasing order without leading zeros$\}$.

s. $\{w$ of the form: $<integer_1> + <integer_2> = <integer_3>$, where each of the substrings $<integer_1>$, $<integer_2>$, and $<integer_3>$ is an element of $\{0 - 9\}^*$ and $integer_3$ is the sum of $integer_1$ and $integer_2\}$. For example, 124+5=129 $\in L$.

t. L_0^*, where $L_0 = \{ba^i b^j a^k, j \geq 0, 0 \leq i \leq k\}$.

u. $\{w : w$ is the encoding of a date that occurs in a year that is a prime number$\}$. A date will be encoded as a string of the form *mm/dd/yyyy*, where each m, d, and y is drawn from {0-9}.

v. $\{w \in \{1\}^* : w$ is, for some $n \geq 1$, the unary encoding of $10^n\}$. (So $L = \{1111111111, 1^{100}, 1^{1000}, \ldots\}$.)

2. For each of the following languages L, state whether L is regular or not and prove your answer:

a. $\{w \in \{a, b, c\}^* :$ in each prefix x of w, $\#_a(x) = \#_b(x) = \#_c(x))\}$.

b. $\{w \in \{a, b, c\}^* : \exists$ some prefix x of w $(\#_a(x) = \#_b(x) = \#_c(x))\}$.

c. $\{w \in \{a, b, c\}^* : \exists$ some prefix x of w $(x \neq \varepsilon$ and $\#_a(x) = \#_b(x) = \#_c(x))\}$.

3. Define the following two languages:

$L_a = \{w \in \{a, b\}^* :$ in each prefix x of w, $\#_a(x) \geq \#_b(x)\}$.

$L_b = \{w \in \{a, b\}^* :$ in each prefix x of w, $\#_b(x) \geq \#_a(x)\}$.

a. Let $L_1 = L_a \cap L_b$. Is L_1 regular? Prove your answer.

b. Let $L_2 = L_a \cup L_b$. Is L_2 regular? Prove your answer.

4. For each of the following languages L, state whether L is regular or not and prove your answer:

a. $\{uww^R v : u, v, w \in \{a, b\}^+\}$.

b. $\{xyzy^R x : x, y, z \in \{a, b\}^+\}$.

5. Use the Pumping Theorem to complete the proof, given in L.3.1, that English isn't regular.

6. Prove *by construction* that the regular languages are closed under:

a. intersection.

b. set difference.

7. Prove that the regular languages are closed under each of the following operations:

a. $pref(L) = \{w : \exists x \in \Sigma^*(wx \in L)\}$.

b. $suff(L) = \{w : \exists x \in \Sigma^*(xw \in L)\}$.

c. $reverse(L) = \{x \in \Sigma^* : x = w^R$ for some $w \in L\}$.

d. letter substitution (as defined in Section 8.3).

8. Using the defintions of *maxstring* and *mix* given in Section 8.6, give a precise definition of each of the following languages:

a. *maxstring*($A^n B^n$).

b. *maxstring*($a^i b^j c^k, 1 \le k \le j \le i$).

c. *maxstring*(L_1L_2), where $L_1 = \{w \in \{a, b\}^* : w$ contains exactly one a$\}$ and $L_2 = \{a\}$.

d. *mix*((aba)*).

e. *mix*(a*b*).

9. Prove that the regular languages are not closed under *mix*.

10. Recall that $maxstring(L) = \{w : w \in L$ and $\forall z \in \Sigma^*(z \neq \varepsilon \rightarrow wz \notin L)\}$.

a. Prove that the regular languages are closed under *maxstring*.

b. If *maxstring*(L) is regular, must L also be regular? Prove your answer.

11. Define the function $midchar(L) = \{c : \exists w \in L(w = ycz, c \in \Sigma_L, y \in \Sigma_L^*, z \in \Sigma_L^*, |y| = |z|)\}$. Answer each of the following questions and prove your answer:

a. Are the regular languages closed under *midchar*?

b. Are the nonregular languages closed under *midchar*?

12. Define the function $twice(L) = \{w : \exists x \in L$ (x can be written as $c_1c_2 \ldots c_n$, for some $n \ge 1$, where each $c_i \in \Sigma_L$, and $w = c_1c_1c_2c_2 \ldots c_nc_n)\}$.

a. Let $L =$ (1 ∪ 0)*1. Write a regular expression for *twice*(L).

b. Are the regular languages closed under *twice*? Prove your answer.

13. Define the function $shuffle(L) = \{w : \exists x \in L$ (w is some permutation of x)$\}$. For example, if $L =$ {ab, abc}, then *shuffle*(L) = {ab, abc, ba, acb, bac, bca, cab, cba}. Are the regular languages closed under *shuffle*? Prove your answer.

14. Define the function $copyandreverse(L) = \{w : \exists x \in L(w = xx^R)\}$. Are the regular languages closed under *copyandreverse*? Prove your answer.

15. Let L_1 and L_2 be regular languages. Let L be the language consisting of strings that are contained in exactly one of L_1 and L_2. Prove that L is regular.

16. Define two integers i and j to be ***twin primes*** ☐ iff both i and j are prime and $|j - i| = 2$.

a. Let $L = \{w \in \{1\}^* : w$ is the unary notation for a natural number n such that there exists a pair p and q of twin primes, both $> n.\}$ Is L regular?

b. Let $L = \{x, y : x$ is the decimal encoding of a positive integer i, y is the decimal encoding of a positive integer j, and i and j are twin primes$\}$. Is L regular?

17. Consider any function $f(L_1) = L_2$, where L_1 and L_2 are both languages over the alphabet $\Sigma = \{0, 1\}$. A function f is ***nice*** iff whenever L_2 is regular, L_1 is regular. For each of the following functions, f, state whether or not it is nice and prove your answer.

a. $f(L) = L^R$.

b. $f(L) = \{w : w$ is formed by taking a string in L and replacing all 1's with 0's and leaving the 0's unchanged$\}$.

c. $f(L) = L \cup 0^*$.

d. $f(L) = \{w : w$ is formed by taking a string in L and replacing all 1's with 0's and all 0's with 1's (simultaneously)$\}$.

e. $f(L) = \{w : \exists x \in L\ (w = x00)\}$.

f. $f(L) = \{w : w$ is formed by taking a string in L and removing the last character$\}$.

18. We'll say that a language L over an alphabet Σ is ***splitable*** iff the following property holds: Let w be any string in L that can be written as $c_1c_2 \ldots c_{2n}$, for some $n \geq 1$, where each $c_i \in \Sigma$. Then $x = c_1c_3 \ldots c_{2n-1}$ is also in L.

a. Give an example of a splitable regular language.

b. Is every regular language splitable?

c. Does there exist a nonregular language that is splitable?

19. Define the class IR to be the class of languages that are both infinite and regular. Tell whether the class IR closed under:

a. union.

b. intersection.

c. Kleene star.

20. Consider the language $L = \{x0^n y\ 1^n z : n \geq 0, x \in P, y \in Q, z \in R$, where P, Q, and R are nonempty sets over the alphabet $\{0, 1\}\}$. Can you find regular sets P, Q, and R such that L is not regular? Can you find regular sets P, Q, and R such that L is regular?

21. For each of the following claims, state whether it is *True* or *False*. Prove your answer.

a. There are uncountably many non-regular languages over $\Sigma = \{a, b\}$.

b. The union of an infinite number of regular languages must be regular.

c. The union of an infinite number of regular languages is never regular.

d. If L_1 and L_2 are not regular languages, then $L_1 \cup L_2$ is not regular.

e. If L_1 and L_2 are regular languages, then $L_1 \otimes L_2 = \{w : w \in (L_1 - L_2)$ or $w \in (L_2 - L_1)\}$ is regular.

f. If L_1 and L_2 are regular languages and $L_1 \subseteq L \subseteq L_2$, then L must be regular.

g. The intersection of a regular language and a nonregular language must be regular.

h. The intersection of a regular language and a nonregular language must not be regular.

i. The intersection of two nonregular languages must not be regular.

j. The intersection of a finite number of nonregular languages must not be regular.

k. The intersection of an infinite number of regular languages must be regular.

l. It is possible that the concatenation of two nonregular languages is regular.

m. It is possible that the union of a regular language and a nonregular language is regular.

n. Every nonregular language can be described as the intersection of an infinite number of regular languages.

o. If L is a language that is not regular, then L^* is not regular.

p. If L^* is regular, then L is regular.

q. The nonregular languages are closed under intersection.

r. Every subset of a regular language is regular.

s. Let $L_4 = L_1L_2L_3$. If L_1 and L_2 are regular and L_3 is not regular, it is possible that L_4 is regular.

t. If L is regular, then so is $\{xy : x \in L \text{ and } y \notin L\}$.

u. Every infinite regular language properly contains another infinite regular language.