2.4 | THE PUMPING LEMMA

A finite automaton accepting a language operates in a very simple way. Not surprisingly, the languages that can be accepted in this way are "simple" languages, but it is not yet clear exactly what this means. In this section, we will see one property that every language accepted by an FA must have.

Suppose that $M = (Q, \Sigma, q_0, A, \delta)$ is an FA accepting $L \subseteq \Sigma^*$ and that Q has n elements. If x is a string in L with $|x| = n - 1$, so that x has n distinct prefixes, it is still conceivable that M is in a different state after processing every one. If $|x| \geq n$, however, then by the time M has read the symbols of x, it must have entered some state twice; there must be two different prefixes u and uv (saying they are different is the same as saying that $v \neq \Lambda$) such that

$$\delta^*(q_0, u) = \delta^*(q_0, uv)$$

This means that if $x \in L$ and w is the string satisfying $x = uvw$, then we have the situation illustrated by Figure 2.28. In the course of reading the symbols of $x = uvw$, M moves from the initial state to an accepting state by following a path that contains a loop, corresponding to the symbols of v. There may be more than one such loop, and more than one such way of breaking x into three pieces u, v, and w; but at least one of the loops must have been completed by the time M has read the first n symbols of x. In other words, for at least one of the choices of u, v, and w such that $x = uvw$ and v corresponds to a loop, $|uv| \leq n$.

The reason this is worth noticing is that it tells us there must be many more strings besides x that are also accepted by M and are therefore in L: strings that cause M to follow the same path but traverse the loop a different number of times. The string obtained from x by omitting the substring v is in L, because M doesn't have to traverse the loop at all. For each $i \geq 2$, the string uv^iw is in L, because M can take the loop i times before proceeding to the accepting state.

The statement we have now proved is known as the Pumping Lemma for Regular Languages. "Pumping" refers to the idea of pumping up the string x by inserting additional copies of the string v (but remember that we also get one of the new strings by leaving out v). "Regular" won't be defined until Chapter 3, but we will see after we define regular languages that they turn out to be precisely the ones that can be accepted by finite automata.

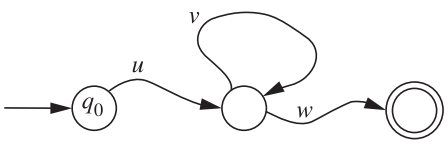

Figure 2.28 |
What the three strings u, v, and w in the pumping lemma might look like.

Theorem 2.29 The Pumping Lemma for Regular Languages
Suppose L is a language over the alphabet Σ. If L is accepted by a finite automaton $M = (Q, \Sigma, q_0, A, \delta)$, and if n is the number of states of M, then for every $x \in L$ satisfying $|x| \geq n$, there are three strings u, v, and w such that $x = uvw$ and the following three conditions are true:

1. $|uv| \leq n$.
2. $|v| > 0$ (i.e., $v \neq \Lambda$).
3. For every $i \geq 0$, the string uv^iw also belongs to L.

Later in this section we will find ways of applying this result for a language L that is accepted by an FA. But the most common application is to show that a language *cannot* be accepted by an FA, by showing that it doesn't have the property described in the pumping lemma.

A proof using the pumping lemma that L cannot be accepted by a finite automaton is a proof by contradiction. We assume, for the sake of contradiction, that L can be accepted by M, an FA with n states, and we try to select a string in L with length at least n so that statements 1–3 lead to a contradiction. There are a few places in the proof where it's easy to go wrong, so before looking at an example, we consider points at which we have to be particularly careful.

Before we can think about applying statements 1–3, we must have a string $x \in L$ with $|x| \geq n$. What is n? It's the number of states in M, but we don't know what M is—the whole point of the proof is to show that it can't exist! In other words, our choice of x must involve n. We can't say "let $x = aababaabbbab$", because there's no reason to expect that $11 \geq n$. Instead, we might say "let $x = a^nba^{2n}$", or "let $x = b^{n+1}a^nb$", or something comparable, depending on L.

The pumping lemma tells us some properties that *every* string in L satisfies, as long as its length is at least n. It is very possible that for some choices of x, the fact that x has these properties does not produce any contradiction. If we don't get a contradiction, we haven't proved anything, and so we look for a string x that *will* produce one. For example, if we are trying to show that the language of palindromes over $\{a, b\}$ cannot be accepted by an FA, there is no point in considering a string x containing only a's, because all the new strings that we will get by using the pumping lemma will also contain only a's, and they're all palindromes too. No contradiction!

Once we find a string x that looks promising, the pumping lemma says that there is *some* way of breaking x into shorter strings u, v, and w satisfying the three conditions. It doesn't tell us what these shorter strings are, only that they satisfy conditions 1–3. If $x = a^nb^na^n$, we can't say "let $u = a^{10}$, $v = a^{n-10}$, and $w = b^na^n$". It's not enough to show that *some* choices for u, v, and w produce a contradiction—we have to show that we *must* get a contradiction, no matter what u, v, and w are, as long as they satisfy conditions 1–3.

Let's try an example.

The Language *AnBn* — EXAMPLE 2.30

Let L be the language *AnBn* introduced in Example 1.18:

$$L = \{a^i b^i \mid i \geq 0\}$$

It would be surprising if *AnBn* could be accepted by an FA; if the beginning input symbols are a's, a computer accepting L surely needs to remember how many of them there are, because otherwise, once the input switches to b's, it won't be able to compare the two numbers.

Suppose for the sake of contradiction that there is an FA M having n states and accepting L. We choose $x = a^n b^n$. Then $x \in L$ and $|x| \geq n$. Therefore, by the pumping lemma, there are strings u, v, and w such that $x = uvw$ and the conditions 1–3 in the theorem are true.

Because $|uv| \leq n$ (by condition 1) and the first n symbols of x are a's (because of the way we chose x), all the symbols in u and v must be a's. Therefore, $v = a^k$ for some $k > 0$ (by condition 2). We can get a contradiction from statement 3 by using any number i other than 1, because $uv^i w$ will still have exactly n b's but will no longer have exactly n a's. The string uv^2w, for example, is $a^{n+k}b^n$, obtained by inserting k additional a's into the first part of x. This is a contradiction, because the pumping lemma says $uv^2w \in L$, but $n + k \neq n$.

Not only does the string uv^2w fail to be in L, but it also fails to be in the bigger language *AEqB* containing all strings in $\{a, b\}^*$ with the same number of a's as b's. Our proof, therefore, is also a proof that *AEqB* cannot be accepted by an FA.

The Language $\{x \in \{a, b\}^* \mid n_a(x) > n_b(x)\}$ — EXAMPLE 2.31

Let L be the language

$$L = \{x \in \{a, b\}^* \mid n_a(x) > n_b(x)\}$$

The first sentence of a proof using the pumping lemma is always the same: Suppose for the sake of contradiction that there is an FA M that accepts L and has n states. There are more possibilities for x than in the previous example; we will suggest several choices, all of which satisfy $|x| \geq n$ but some of which work better than others in the proof.

First we try $x = b^n a^{2n}$. Then certainly $x \in L$ and $|x| \geq n$. By the pumping lemma, $x = uvw$ for some strings u, v, and w satisfying conditions 1–3. Just as in Example 2.30, it follows from conditions 1 and 2 that $v = b^k$ for some $k > 0$. We can get a contradiction from condition 3 by considering $uv^i w$, where i is large enough that $n_b(uv^i w) \geq n_a(uv^i w)$. Since $|v| \geq 1$, $i = n + 1$ is guaranteed to be large enough. The string $uv^{n+1}w$ has at least n more b's than x does, and therefore at least $2n$ b's, but it still has exactly $2n$ a's.

Suppose that instead of $b^n a^{2n}$ we choose $x = a^{2n} b^n$. This time $x = uvw$, where v is a string of one or more a's and $uv^i w \in L$ for every $i \geq 0$. The way to get a contradiction now is to consider $uv^0 w$, which has fewer a's than x does. Unfortunately, this produces a contradiction only if $|v| = n$. Since we don't know what $|v|$ is, the proof will not work for this choice of x.

The problem is not that x contains a's before b's; rather, it is that the original numbers of a's and b's are too far apart to guarantee a contradiction. Getting a contradiction in this case means making an inequality fail; if we start with a string in which the inequality is

just barely satisfied, then ideally any change in the right direction will cause it to fail. A better choice, for example, is $x = a^{n+1}b^n$. (If we had used $x = b^n a^{n+1}$ instead of $b^n a^{2n}$ for our first choice, we could have used $i = 2$ instead of $i = n$ to get a contradiction.)

Letting $x = (ab)^n a$ is also a bad choice, but for a different reason. We know that $x = uvw$ for some strings u, v, and w satisfying conditions 1–3, but now we don't have enough information about the string v. It might be $(ab)^k a$ for some k, so that uv^0w produces a contradiction; it might be $(ba)^k b$, so that uv^2w produces a contradiction; or it might be either $(ab)^k$ or $(ba)^k$, so that changing the number of copies of v doesn't change the relationship between n_a and n_b and doesn't give us a contradiction.

EXAMPLE 2.32 The Language $L = \{a^{i^2} \mid i \geq 0\}$

Whether a string of a's is an element of L depends only on its length; in this sense, our proof will be more about numbers than about strings.

Suppose L can be accepted by an FA M with n states. Let us choose x to be the string a^{n^2}. Then according to the pumping lemma, $x = uvw$ for some strings u, v, and w satisfying conditions 1–3. Conditions 1 and 2 tell us that $0 < |v| \leq n$. Therefore,

$$n^2 = |uvw| < |uv^2w| = n^2 + |v| \leq n^2 + n < n^2 + 2n + 1 = (n+1)^2$$

This is a contradiction, because condition 3 says that $|uv^2w|$ must be i^2 for some integer i, but there is no integer i whose square is strictly between n^2 and $(n+1)^2$.

EXAMPLE 2.33 Languages Related to Programming Languages

Almost exactly the same pumping-lemma proof that we used in Example 2.30 to show *AnBn* cannot be accepted by a finite automaton also works for several other languages, including several that a compiler for a high-level programming language must be able to accept. These include both the languages *Balanced* and *Expr* introduced in Example 1.19, because $(^m)^n$ is a balanced string, and $(^m a)^n$ is a legal expression, if and only if $m = n$.

Another example is the set L of legal C programs. We don't need to know much about the syntax of C to show that this language can't be accepted by an FA—only that the string

```
main(){{{ ... }}}
```

with m occurrences of "{" and n occurrences of "}", is a legal C program precisely if $m = n$. As usual, we start our proof by assuming that L is accepted by some FA with n states and letting x be the string `main()` $\{^n\}^n$. If $x = uvw$, and these three strings satisfy conditions 1–3, then the easiest way to obtain a contradiction is to use $i = 0$ in condition 3. The string v cannot contain any right brackets because of condition 1; if the shorter string uw is missing any of the symbols in "`main()`", then it doesn't have the legal header necessary for a C program, and if it is missing any of the left brackets, then the two numbers don't match.

As the pumping lemma demonstrates, one way to answer questions about a language is to examine a finite automaton that accepts it. In particular, for languages that can be accepted by FAs, there are several *decision problems* (questions with