4.1 Proving Languages not to be Regular

We have established that the class of languages known as the regular languages has at least four different descriptions. They are the languages accepted by DFA's, by NFA's, and by ε-NFA's; they are also the languages defined by regular expressions.

Not every language is a regular language. In this section, we shall introduce a powerful technique, known as the “pumping lemma,” for showing certain languages not to be regular. We then give several examples of nonregular languages. In Section 4.2 we shall see how the pumping lemma can be used in tandem with closure properties of the regular languages to prove other languages not to be regular.

4.1.1 The Pumping Lemma for Regular Languages

Let us consider the language $L_{01} = \{0^n1^n \mid n \geq 1\}$. This language contains all strings 01, 0011, 000111, and so on, that consist of one or more 0's followed by an equal number of 1's. We claim that $L_{01}$ is not a regular language. The intuitive argument is that if $L_{01}$ were regular, then $L_{01}$ would be the language of some DFA A. This automaton has some particular number of states, say $k$ states. Imagine this automaton receiving $k$ 0's as input. It is in some state after consuming each of the $k+1$ prefixes of the input: $\epsilon, 0, 00, \ldots, 0^k$. Since there are only $k$ different states, the pigeonhole principle tells us that after reading two different prefixes, say 0$^i$ and 0$^j$, A must be in the same state, say state $q$.

However, suppose instead that after reading $i$ or $j$ 0's, the automaton A starts receiving 1's as input. After receiving $i$ 1's, it must accept if it previously received $i$ 0's, but not if it received $j$ 0's. Since it was in state $q$ when the 1's started, it cannot “remember” whether it received $i$ or $j$ 0's, so we can “fool” A and make it do the wrong thing -- accept if it should not, or fail to accept when it should.

The above argument is informal, but can be made precise. However, the same conclusion, that the language $L_{01}$ is not regular, can be reached using a general result, as follows.

**Theorem 4.1:** (The pumping lemma for regular languages) Let $L$ be a regular language. Then there exists a constant $n$ (which depends on $L$) such that for every string $w$ in $L$ such that $|w| \geq n$, we can break $w$ into three strings, $w = xyz$, such that:

1. $y \neq \epsilon$.
2. $|xy| \leq n$.
3. For all $k \geq 0$, the string $xy^kz$ is also in $L$.

That is, we can always find a nonempty string $y$ not too far from the beginning of $w$ that can be “pumped”; that is, repeating $y$ any number of times, or deleting it (the case $k = 0$), keeps the resulting string in the language $L$. 

4.1. PROVING LANGUAGES NOT TO BE REGULAR

**Proof:** Suppose $L$ is regular. Then $L = L(A)$ for some DFA $A$. Suppose $A$ has $n$ states. Now, consider any string $w$ of length $n$ or more, say $w = a_1a_2 \cdots a_m$, where $m \geq n$ and each $a_i$ is an input symbol. For $i = 0, 1, \ldots, n$ define state $p_i$ to be $\delta(q_0, a_1a_2 \cdots a_i)$, where $\delta$ is the transition function of $A$, and $q_0$ is the start state of $A$. That is, $p_i$ is the state $A$ is in after reading the first $i$ symbols of $w$. Note that $p_0 = q_0$.

By the pigeonhole principle, it is not possible for the $n + 1$ different $p_i$'s for $i = 0, 1, \ldots, n$ to be distinct, since there are only $n$ different states. Thus, we can find two different integers $i$ and $j$, with $0 \leq i < j \leq n$, such that $p_i = p_j$. Now, we can break $w = xyz$ as follows:

1. $x = a_1a_2 \cdots a_i$.
2. $y = a_{i+1}a_{i+2} \cdots a_j$.
3. $z = a_{j+1}a_{j+2} \cdots a_m$.

That is, $x$ takes us to $p_i$ once; $y$ takes us from $p_i$ back to $p_i$ (since $p_i$ is also $p_j$), and $z$ is the balance of $w$. The relationships among the strings and states are suggested by Fig. 4.1. Note that $x$ may be empty, in the case that $i = 0$. Also, $z$ may be empty if $j = n = m$. However, $y$ can not be empty, since $i$ is strictly less than $j$.

![Figure 4.1: Every string longer than the number of states must cause a state to repeat](image)

Now, consider what happens if the automaton $A$ receives the input $xy^kz$ for any $k \geq 0$. If $k = 0$, then the automaton goes from the start state $q_0$ (which is also $p_0$) to $p_1$ on input $x$. Since $p_i = p_j$, it must be that $A$ goes from $p_i$ to the accepting state shown in Fig. 4.1 on input $z$. Thus, $A$ accepts $xz$.

If $k > 0$, then $A$ goes from $q_0$ to $p_i$ on input $x$, circles from $p_i$ to $p_i$ $k$ times on input $y^k$, and then goes to the accepting state on input $z$. Thus, for any $k \geq 0$, $xy^kz$ is also accepted by $A$; that is, $xy^kz$ is in $L$. \( \Box \)

4.1.2 Applications of the Pumping Lemma

Let us see some examples of how the pumping lemma is used. In each case, we shall propose a language and use the pumping lemma to prove that the language is not regular.
The Pumping Lemma as an Adversarial Game

Recall our discussion from Section 1.2.3 where we pointed out that a theorem whose statement involves several alternations of "for all" and "there exists" quantifiers can be thought of as a game between two players. The pumping lemma is an important example of this type of theorem, since it in effect involves four different quantifiers: "for all regular languages $L$, there exists $n$ such that for all $w$ in $L$ with $|w| \geq n$ there exists $xyz$ equal to $w$ such that $\cdots$." We can see the application of the pumping lemma as a game, in which:

1. Player 1 picks the language $L$ to be proved nonregular.
2. Player 2 picks $n$, but doesn't reveal to player 1 what $n$ is; player 1 must devise a play for all possible $n$'s.
3. Player 1 picks $w$, which may depend on $n$ and which must be of length at least $n$.
4. Player 2 divides $w$ into $x$, $y$, and $z$, obeying the constraints that are stipulated in the pumping lemma; $y \neq \varepsilon$ and $|xy| \leq n$. Again, player 2 does not have to tell player 1 what $x$, $y$, and $z$ are, although they must obey the constraints.
5. Player 1 "wins" by picking $k$, which may be a function of $n$, $x$, $y$, and $z$, such that $xy^kz$ is not in $L$.

Example 4.2: Let us show that the language $L_{eq}$ consisting of all strings with an equal number of 0's and 1's (not in any particular order) is not a regular language. In terms of the "two-player game" described in the box on "The Pumping Lemma as an Adversarial Game," we shall be player 1 and we must deal with whatever choices player 2 makes. Suppose $n$ is the constant that must exist if $L_{eq}$ is regular, according to the pumping lemma; i.e., "player 2" picks $n$. We shall pick $w = 0^n1^n$, that is, $n$ 0's followed by $n$ 1's, a string that surely is in $L_{eq}$.

Now, "player 2" breaks our $w$ up into $xyz$. All we know is that $y \neq \varepsilon$, and $|xy| \leq n$. However, that information is very useful, and we "win" as follows. Since $|xy| \leq n$, and $xy$ comes at the front of $w$, we know that $x$ and $y$ consist only of 0's. The pumping lemma tells us that $xz$ is in $L_{eq}$, if $L_{eq}$ is regular. This conclusion is the case $k = 0$ in the pumping lemma.\(^1\) However, $xz$ has $n$ 1's, since all the 1's of $w$ are in $z$. But $xz$ also has fewer than $n$ 0's, because we

\(^1\)Observe in what follows that we could have also succeeded by picking $k = 2$, or indeed any value of $k$ other than 1.
4.1. PROVING LANGUAGES NOT TO BE REGULAR

lost the 0’s of y. Since y ≠ ε we know that there can be no more than n − 1 0’s among x and z. Thus, after assuming L_{eq} is a regular language, we have proved a fact known to be false, that xz is in L_{eq}. We have a proof by contradiction of the fact that L_{eq} is not regular. □

Example 4.3: Let us show that the language L_{pr} consisting of all strings of 1’s whose length is a prime is not a regular language. Suppose it were. Then there would be a constant n satisfying the conditions of the pumping lemma. Consider some prime p ≥ n + 2; there must be such a p, since there are an infinity of primes. Let w = 1^p.

By the pumping lemma, we can break w = xyz such that y ≠ ε and |xy| ≤ n. Let |y| = m. Then |xz| = p − m. Now consider the string y^{p−m}z, which must be in L_{pr} by the pumping lemma, if L_{pr} really is regular. However,

|y^{p−m}z| = |xz| + (p−m)|y| = p − m + (p−m)m = (m+1)(p−m)

It looks like |y^{p−m}z| is not a prime, since it has two factors m + 1 and p − m. However, we must check that neither of these factors are 1, since then (m + 1)(p − m) might be a prime after all. But m + 1 > 1, since y ≠ ε tells us m ≥ 1. Also, p − m > 1, since p ≥ n + 2 was chosen, and m ≤ n since

m = |y| ≤ |xy| ≤ n

Thus, p − m ≥ 2.

Again we have started by assuming the language in question was regular, and we derived a contradiction by showing that some string not in the language was required by the pumping lemma to be in the language. Thus, we conclude that L_{pr} is not a regular language. □

4.1.3 Exercises for Section 4.1

Exercise 4.1.1: Prove that the following are not regular languages.

a) \{0^n1^n \mid n \geq 1\}. This language, consisting of a string of 0’s followed by an equal-length string of 1’s, is the language L_{01} we considered informally at the beginning of the section. Here, you should apply the pumping lemma in the proof.

b) The set of strings of balanced parentheses. These are the strings of characters "(" and ")" that can appear in a well-formed arithmetic expression.

* c) \{0^n1^n \mid n \geq 1\}.

d) \{0^n1^n2^n \mid n and m are arbitrary integers\}.

e) \{0^n1^m \mid n \leq m\}.

f) \{0^n1^{2n} \mid n \geq 1\}.