Theory of Computation: Non-regular Languages

Arjun Chandrasekhar

1 The pigeonhole principle

Theorem 1.1. The **pigeonhole principle** is a (seemingly obvious) principle which states that if there are m pigeons, and $n > m$ holes, then there must be a hole with more than one pigeon. More formally, if a function's co-domain is smaller than it's domain, the function cannot be injective (one-to-one).

Example 1.1. If I have three gloves, at least two of them must fit the same hand.

Example 1.2. If there is a room with 367 people, there must be a shared birthday.

Proposition 1.1. Let G be a directed graph in which every node has positive out-degree. G must have a directed cycle.

Proof. We will construct a random walk $u_1, u_2, \ldots$. For the first node, we pick a random node in the graph. At every step, if we are at a node u, we will walk to a random neighbor of u. Because every node has positive out-degree, there will always be at least one available node. Thus, we can extend our random walk ad infinitum.

Suppose G has n nodes. Consider nodes $u_1, u_2, \ldots, u_{n+1}$, i.e. the first $n+1$ nodes in the random walk. There are more nodes in our random walk than there are nodes in the graph. By the pigeonhole principle, our random walk has touched at least one node more than once, thus creating a cycle. □

2 The pumping lemma

A DFA can be thought of a directed graph. Because a DFA must have every transition defined for every state/symbol combination, every node in the DFA state graph must have positive out-degree. When the DFA reads a string, it essentially performs a random walk between its states. If we read a string whose length is longer than the number of states in the DFA, proposition 1.1 tells us that there will be a directed cycle.

If the DFA accepts a 'sufficiently long' string w, it will read some prefix of w to get to some state s, read the middle part of w to loop from s back to itself, and then read some suffix of w to go from s to an accept state. However, the DFA could have just as easily looped from s back to itself twice befroe going to the accept state. It could have also looped three times, four times, or even zero times! Thus, there are an infinite number of strings that the DFA can accept by simply 'pumping' as many times as desired. We formalize this below.

Lemma 2.1 (Regular language pumping lemma)**.** If A is a regular langugae, there is a number p (the pumping length) where if s is a string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following three conditions:

1. $|xy| \leq p$
2. $|y| > 0$
3. for each $i \geq 0$, $xy^iz \in A$

Recall the notation where $|s|$ represents the length of string s, y^i means that i copies of y are concatenated together, and $y^0 = \epsilon$.

When we divide s into xyz, condition 1 states that xy is contained within the first p characters. Condition 2 states that while either x or y (or both) may be ϵ, but y must have positive length (otherwise the lemma would be trivially true). Condition 3 states that we can add (or remove) any number of copies of y, and the resulting string will still be in the language.

Proof. Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA recognizing A and let p be the number of states of M, i.e. $|Q|$.

Let $s = s_1 s_2 \dots s_n \in A$ be a string of length $n \geq p$. Let $r_1, r_2, \dots, r_{n+1}$ be the sequence of states that M enters while processing s, so $r_{i+1} = \delta(r_i, s_i)$ for $1 \leq i \leq n$. This sequence has length $n + 1 \geq p + 1$. Among the first $p+1$ elements in the sequence, two must be the same state by the pigeonhole principle. We call the first of these r_j and the second r_l, with $l \leq p + 1$. Now let $x = s_1 \dots s_{j-1}$, $y = s_j \dots s_{l-1}$, and $z = s_l \dots s_n$.

As x takes M from r_1 to r_j, y takes M from r_j to r_j, and z takes M from r_j to r_{n+1}, which is an accept state. M must accept xy^iz for all $i \geq 0$. We now that $j \neq l$, so $|y| > 0$; and $l \leq p + 1$, so $|xy| \leq p$. Thus we have satisfied all conditions of the pumping lemma. □

Corollary 2.1. Suppose A is a language that does not have a valid pumping length. Then A is not regular.

Remark. Every finite language is regular. The pumping lemma claims that if a language is regular, then every sufficiently long string can be pumped infinitely many times. How do we reconcile this apparent contradiction?

Let L be a finite language, and let N be the length of the longest string in L. We set the pumping length to be $p = L + 1$. The pumping lemma states that every string *whose length is at least* p can be split into xyz and pumped. We are not on the hook for any strings whose length is shorter than p. Thus, we don't have to worry abouy pumping any of the strings in L. Technically, every string in L whose length is at least p can be pumped - this statement is vacuously true.

3 Pumping lemma arguments

The pumping lemma gives us a tool to prove that certain languages are not regular. If we show that every possible pumping length p fails for language A, we conclude that A is not actually regular because it can't be recognized by any DFA.

Pumping lemma arguments typically follow the following formula:

1. Assume for sake contradiction that A is a regular language. Then it must have a pumping length p
2. Pick some string w such that $w \in A$ and $|w| \geq p$
3. Show that w is not 'pumpable'; That is, show that no matter how we split up $w = xyz$, there exists some i such that $xy^iz \notin A$. Note that we may assume w is split up such that $|xy| \leq p$ and $|y| > 0$.
4. Because A contains a string that is not pumpable, this contradicts the assumption that p is a valid pumping length for A. Thus, we have arrived at a contradiction and reject the claim that A is regular.

Example 3.1. Let B be the language $\{0^n1^n | n \geq 0\}$. We use the pumping lemma to prove that B is not regular.

Assume for sake of contradiction that B is regular. Let p be the pumping length given by the pumping lemma. Choose s to be the string 0^p1^p. Because s is a member of B and s has length $\geq p$, the pumping lemma guarantees that s can be split into three pieces, $s = xyz$, such that:

1. $|xy| \leq p$
2. $|y| > 0$
3. $xy^iz \in B$ for all i

Because $|xy| \leq p$, the xy part of s must be contained in 0^p. The second condition ensures that y contains a positive number of 0's. When we 'pump up', i.e. add more copies of y, we will have more 0's than 1's. If we 'pump down', we will have fewer 0's than 1's. Either way, after pumping we will produce a string that is not in B. Thus, 0^p1^p is not pumpable, a contradiction of the pumping lemma. We conclude that B is not regular.

4 The pumping lemma as a two player game

Sometimes it can be confusing to remember the order of the quantifiers. In particular, it can be difficult to remember which things can and cannot be assumed about the language and the string. One helpful way to think about the pumping lemma is to think of it as a two player game.

1. Player 1 declares that L is a regular language, and declares a pumping length p
2. Player 2 picks a string s such that $s \in L$ and $|s| \geq p$. If player 2 cannot do this, player 1 wins.
3. Player 1 splits up s into three parts xyz. Player 1 must ensure that $|xy| \leq p$ and $|y| > p$.
4. Player 2 tries to pump y up or down in order to produce a string that is not in L

If L is regular, the pumping lemma tells us there exists a valid pumping length p; Player 1 can declare this pumping length p, and player 1 will have a way to win no matter what string player 2 picks.

If L is not regular, no matter what pumping length player 1 picks, there will always be a valid string that player 2 can pick in order to guarantee a win. In particular, there is some string s that player 1 can pick, and no matter how player 1 splits s into three parts, player 2 can pump up or down to generate a string that is not in the language.

Example 4.1. Let's prove that 0^n1^n is not regular, using the two-player game interpretation.

1. Player 1 declares L is regular, and picks some pumping length p.
2. Depending on what value of p player 1 picks, player 2 picks $s = 0^p1^p$.
3. Player 1 splits s into $s = xyz$. Note that player 1 is obligated to make sure that xy is contained within 0^p, and y must be non-empty.
4. Player 2 pumps y (either up or down) and produces a string that is not in the language.

Player 2 is guaranteed to win for *every possible choice of* p. Thus, L is not regular.

4.1 Exercises

1. Let $\Sigma = \{0, 1\}$. Prove that the language L = $\{ww^R \mid w^R$ is the reverse of $w\}$ is not regular.

2. Let $\Sigma = \{0, 1, +, =\}$ and let ADD = {x + y = z | x, y, z are binary numbers}. Prove that ADD is not regular.