Pumping Lemma For Regular Languages

Manasseh Ahmed

1 Preface

Late last year(2021), I audited an introductory course on theoretical computer science. One of the most used results was the pumping lemma. In writing this handout, I'm essentially going through my memory and trying to derive the pumping lemma. So a big reason why I'm writing this is to test my own understanding. However, the work here is somewhat rigorous, and the proofs provided here might help your intuition, in addition, there are exercises. Lastly, if you find any errors, you should contact me.

2 Prerequisites

You should have some idea about what a language, a string, and an alphabet are. Previous experience with regular languages and DFAs would help, however I will cover these concepts briefly.

3 Regular languages and DFAs

A language is a set of strings, infinite or finite. In order to understand regular languages, you must first understand a few operations/sets which we define here.

Definition 3.1. The empty language, denoted as $\emptyset$. It is just an empty set.

Definition 3.2. The empty string language, denoted as $\{\epsilon\}$, is the language consisting of only the empty string, notably it is different from the empty language.

Definition 3.3. The union operation is an operation on two languages, say A and B, that produces another language whose words are either from A, B, or both. The union of A and B would be denoted as $A \cup B$.

Definition 3.4. The concatenation of two strings X and Y, denoted as $X \parallel Y$, is a string whose first half(from the left) is X and whose second half is Y. The concatenation of two languages, A and B, is the set which consists of the concatenation of all possible string pairs, with the first from A and second from B. This is denoted as $A \parallel B$.

For example, $aaa \parallel bbb = aaabbb$ and $\{a, b\} \parallel \{a, aa\} = \{aa, aaa, ba, baa\}$

Definition 3.5. The Kleene star of a language L is defined as $L^* = \{\epsilon\} \cup \bigcup_{i=1}^{\infty} L^i$, where $L^i = L \parallel L ... \parallel L$, where there are i copies of L.

Exercise 3.1. Find $\{1\}^*$.

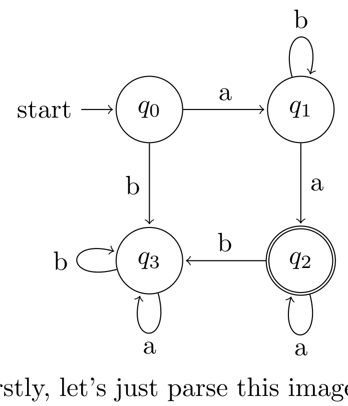
We can now define regular languages

Definition 3.6. A language is regular over some alphabet Σ if it can be expressed using the operations $\cup, *$, or $\parallel$(where parentheses are permitted) over other regular languages, or if it is either $\emptyset, \{\epsilon\}$, or $\{a\}, a \in \Sigma$.

There is another way to define regular languages, and it's by using DFAs, we will now define these, although we won't prove the equivalency between the two.

Definition 3.7. DFA stands for deterministic finite automaton. Put simply, it's a mathematical object, which takes a string as an input and either accepts or rejects it depending on certain computations that it makes. It's a very simplistic model of computation. We present an informal definition.
A DFA takes in strings, all of which are over the same alphabet Σ. The DFA also has a finite number of states, we denote the state set as S, one of which is the start state, which we call s_0. There is also a set of accept states, which we call S'. Here's how the DFA does computation, after receiving a string, $a_1a_2...a_n$, it starts by looking at a_1, a_1 can be anything in Σ, and for each state of the DFA, there is another state which the DFA goes to for each letter in Σ. So it reads a_1 while in the state s_0, depending on what it reads it goes to another state, and at that state it will read a_2, and continues as before. Once all letters have been read the DFA will see if it is at an accept state or not, depending on this, it decides whether or not it accepts the string.

This is sort of complicated, so here's an example: Let's say we have a DFA over $\{a, b\}$. In order to graphically repreresent the states and how the DFA acts, graphs like the one below are often used:



Firstly, let's just parse this image. Clearly q_0 is the start state, what about the accept state(s)? Well, in this case it would be q_2, since its border is different from normal. Now, let's do some example computations, say we input *abab*, here's what would happen:

- We start at q_0, and read an a, as shown on the diagram, this makes us move to q_1.
- We now read a b at q_1, which keeps us at q_1.
- We read an a at q_1, which takes us to q_2.
- We read a b, and end at q_3, which is not an accept state, so clearly our DFA rejects the string.

The set of all strings accepted by a DFA is its language.

Exercise 3.2. Our definition of a DFA was informal, try and find a formal definition.

Exercise 3.3. Find the regular expression for the language recognized by the DFA we showed.

4 The Pumping Lemma

We now look at the pumping lemma, a result which essentially lets us determine if a language is non regular.

Theorem 1. *Every regular language has a pumping length p, for all strings of a length greater than or equal to this pumping length, the following can be done: Denote the string as s, then we can rewrite s as xyz, with the length of xy less than or equal to p, the length of y itself being non zero, and every string xy^nz is also in the language, where $0 \leq n$, and $a^n = aa...aa$, with n a's.*

Proof. We already know that DFAs and regular languages are equivalent, thus to prove this lemma, we utilize the notion of a DFA. So for a regular language L, let A be its DFA, let's say that A has n states, we claim that the pumping length $p = n$. Assume that we have a string of length n, call it s, now when our DFA operates on s, it starts at the start state, and then visits n states, since s has length n. So, A visits $n + 1$ states, while having only n. By the pigeonhole principle, this means that a given state is visited twice. If you don't know what the pigeonhole principle is, you can think about this intuitively, A visits more states than there are, so at least one must be visited twice. Let's call the first state which is repeated r. We claim that the portion of s which got the DFA to r is x, the portion which causes it to go back to r is y, and the rest is z. Clearly $|xy| \leq p$, due to the length of the string itself, and also $|y| > 0$, as you need to read some characters to get back to r from r, remember, we visit it at least twice. Lastly, we need to show xy^nz is in L for all non negative n. However, since y takes A from r to r, adding copies of it in the middle of the string will keep s in the language, as z still gets read from r, removing y works for the same reason. Thus the lemma has been proven(note that for strings longer than this the proof is analogous). $\square$

Note that this lemma really says nothing about non regular language directly, meaning that a language could satisfy these criteria and still be non regular. However, if a language doesn't satisfy them, we have confirmation that it is not a regular language. We conclude with an example.

Example 4.1. The language $L = \{a^n b^n | n \in \mathbb{N}\}$ is non regular.

Proof. This is the classic non regular language, so I had to include it here. Here's how you'd prove it though. Assume L is regular, then it must satisfy the pumping lemma. Assume it has pumping length p, clearly $a^p b^p \in L$. We now partition the string as according to the pumping lemma. Clearly xy would have to be included within the sequence of a's, as it has length p. However, no matter where it is located, pumping will cause problems(pumping refers to looking at the strings $xz, xyz, xy^2z, ...$). As, since the y is located within the sequence of a's, pumping would cause there to be more a's than b's. Meaning the strings produced wouldn't be in L. Thus, L is non-regular, since it doesn't satisfy the pumping lemma. $\square$

Here's one exercise to try out:

Exercise 4.1. Show that $L = \{a^p | p \text{ is prime }\}$ is non regular.

5 Conclusion

I hope this handout has given you newfound insight into the pumping lemma, and I hope that the exercises were fun! Make sure to contact me if you find any errors, and thanks for reading.

6 Works Cited

Introduction to the Theory of Computation by Michael Sipser
Theory of Automata by Arto Salomaa