Lecture 9

Ben Rosenberg

June 21, 2021

Definition: A context-free grammar is **regular** if every rule satisfies one of the following forms:

1. $A \to a$ (single terminal)
2. $A \to \lambda$ (empty string)
3. $A \to aB$ (terminal followed by variable)

where A, B are variables and a is a terminal.

$G_1 = (V, \Sigma, P, S)$

$V = \{S, A\}$

P:

- $S \to AaAaA$
- $A \to \lambda | AA | bc$

This is the language with exactly two a's. It is *not* regular because of the fact that there are multiple terminals in both rules; namely, the rule $A \to AA$ is a blatant offender.

We can write a regular expression for G_1 as follows:

$$(b \cup c)^* a (b \cup c)^* a (b \cup c)^*$$

Now let's write a regular grammar for this regex.

$G_2 = (V, \Sigma, P, S)$

$V = \{S, N, Z\}$

P:

- $S \to aN | bS | cS$
- $N \to aZ | bN | cN$
- $Z \to bZ | cZ | \lambda$

This leads us to our algorithm, which is that regular grammars have an algorithm for conversion to finite automata and back.

The language generated by a regular grammar is always regular.

Algorithm 1: Algorithm to convert between regular grammars and finite automata, where the finite automaton must satisfy:

- initial state with ~~zero in-degree~~
- single final state with zero out-degree
- no λ-transitions except into the final state

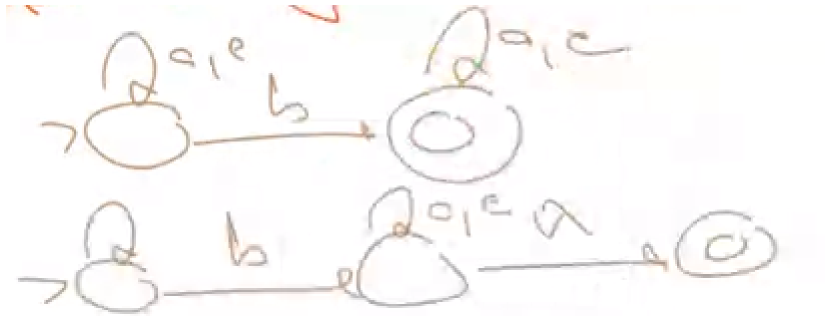

Figure 1: Noncompliant automaton (above) and its compliant version

To put the automaton into this form, we can simply make it deterministic, fix the final states, and make λ-arcs into the final states from all the other final states.

The conversion mapping is as follows:

grammar	automaton
$G = (V, \Sigma, P, S)$	$M = (Q, \Sigma, \delta, q_0, F)$
$V = Q \backslash \{Z\}$	$F = \{Z\}, Z \notin V, Q = V \cup \{Z\}$
$S = q_0$	$q_0 = S$
P	δ

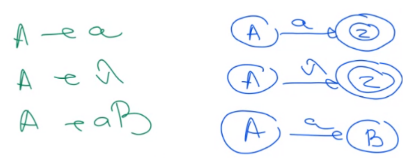

Figure 2: Example conversion

Problem 1: $\Sigma = \{a, b, c\}$; strings that contain exactly one b or exactly one c.

$\Sigma = \{a, b, c\}$

$G = (V, \Sigma, P, S)$

$V = \{S, \}$

P:

- $S \to aS|bB|cK$
- $B \to aB|cB|\lambda$
- $K \to aK|bK|\lambda$

We will have four states: one for each of the variables (S, B, K), and one for the final state (Z).

Problem 2: Strings that contain exactly 2 c's; going from automaton to grammar

$G = \{V, \Sigma, P, S\}$

$V = \{S, A, B\}$ (the states, without Z)

P:

- $S \to aS|bS|cA$
- $A \to aA|bA|cB$
- $B \to aB|bB|\lambda$

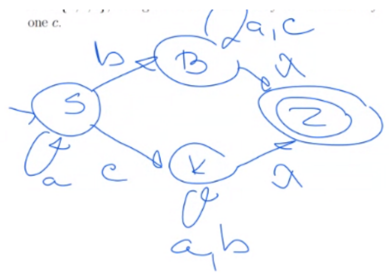

Figure 3: Automaton from grammar conversion

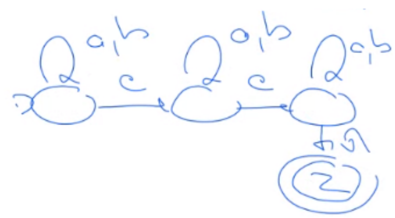

Figure 4: Strings that contain exactly 2 c's

This algorithm is separate from Kleene's Theorem, but is of course useful nonetheless.

Pumping Lemma for Regular Languages

The idea now is to prove that a language is not regular.

The pumping lemma study plan follows:

- study
- read and understand, and remember
- prove
- apply
- practice

Theorem (pumping lemma):

Let L be a regular language. Then:

$$\begin{aligned}(\exists k > 0)(\forall w \in L)(|w| > k \implies \\ ((\exists x, y, z \in \Sigma^*)(w = xyz \wedge \\ |xy| \leq k \wedge \\ |y| > 0 \wedge \\ ((\forall i \geq 0)(xy^i z \in L)))))\end{aligned}$$

The "pumping" part here is the same string concatenated to itself multiple (i) times.

The pumping lemma says that all regular languages "pump": that is, every good string that is long enough will pump. By long enough, we mean that if a language is regular, there will be a positive constant such that every good string that is at least that long will pump – inside the string, but within the first k symbols, there must exist what is called a pumping window (which is nonempty). The "pumping" is repeating itself in place any number of times.

Proof: Let L be regular. Then, let M be a DFA that accepts L, and let M have k states.

Let $w \in L$ be a string that belongs to L and let $|w| \geq k$. (We call w a "good, long string".)

Observe M as it processes and accepts w:

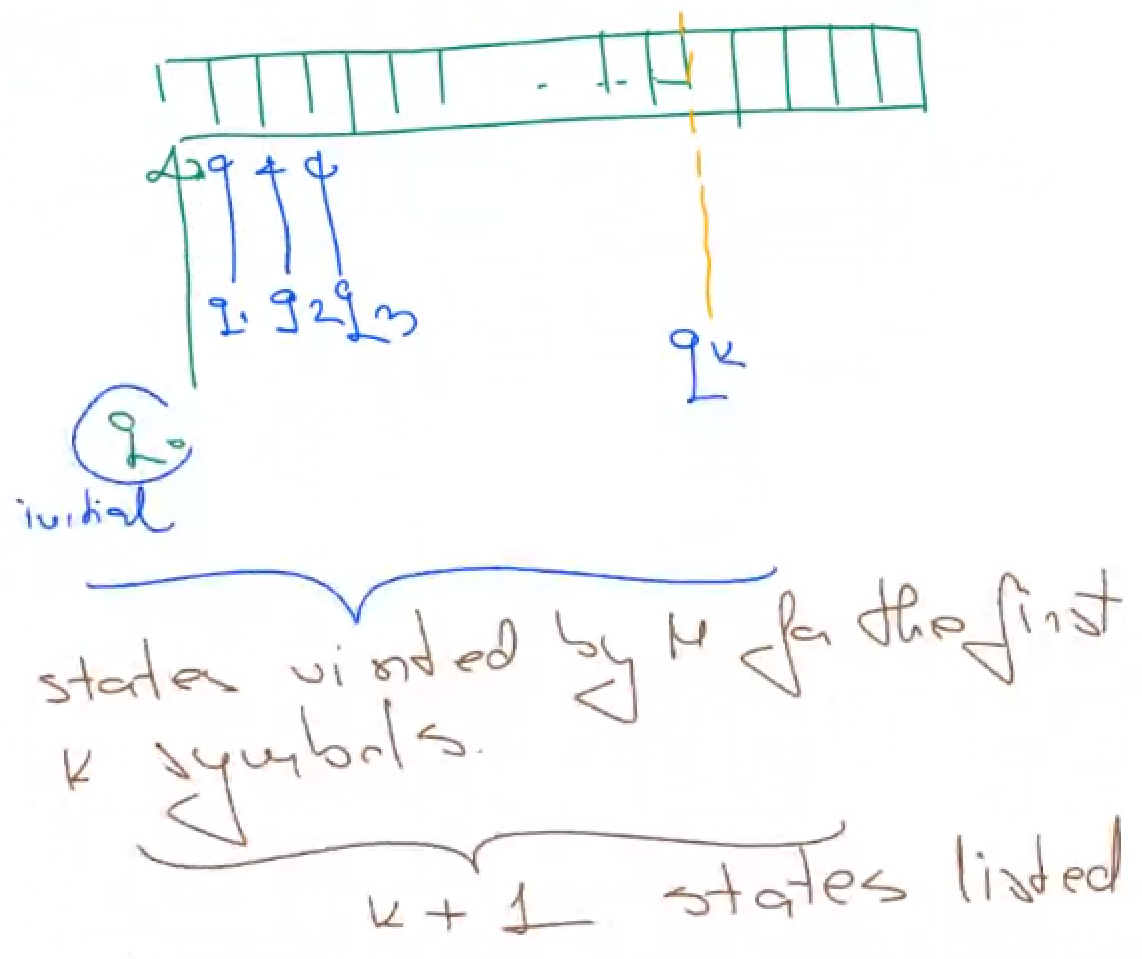

Figure 5: Diagram of pumping automaton

We can see that we have listed $k+1$ states (because there are k symbols), but the machine only has k states. Therefore, by the pigeonhole principle, we know that on the list, at least one state appears at least twice.

We will call this repeated state p. Let's call the substring before the first appearance of p by the name x; the substring between the two appearances of p, y; and the substring after the second appearance of p, z. Note that x and z can be empty, but y cannot as is needs to appear between two distinct occurrences of p.

Now, we need to write configurations of M. We begin with (q_0, xyz) (or, alternatively, (q_0, w)). Then, we go to (p, yz):

$$(q_0, xyz) \to (p, yz) \to (p, z) \to \text{accepts}$$

Example (exponent of i is 2):

- $(q_0, xyyz) \to (p, yyz)$
- $(p, yyz) \to (p, yz)$
- $(p, yz) \to (p, z)$

- $(p, z) \to$ accept

This pattern should be clear.

Another example:

- $(q_0, xz) \to (p, z) \to$ accept (exponent of i is 0)

Note that if we can't find a good long string w as indicated by the pumping lemma, then the regular expression must be finite.

Recall: PL (pumping lemma): if L is regular, then L pumps. Note that this is an implication, not an iff – meaning, that languages could still pump and not be regular. However, if we find a language that does not pump, then it must not be regular.

To prove that L is not regular, we prove that it cannot pump (it violates the pumping lemma). In other words, we want to show that the negation of the PL holds for L.

We can write the negation of the pumping lemma as follows:

$$
\begin{aligned}
&(\forall k > 0)(\exists w \in L)(|w| > k \wedge \\
&\quad ((\forall x, y, z \in \Sigma^*)(w = xyz \wedge \\
&\qquad |xy| \leq k \wedge \\
&\qquad |y| > 0 \implies \\
&\qquad\quad ((\exists i \geq 0)(xy^i z \notin L)))))
\end{aligned}
$$

If the above holds for some language L then L is not regular.

To prove that a language L is not regular:

1. Recognize a property that must be satisfied by **all** elements of L;
2. Assume tht L is regular, name a positive constant of the Pumpiung Lemma;
3. Select an element $w \in L$ that is long enough to pump;
4. For every admissible pumping decomposition $w = xyz$
    - Select i such that $xy^i z \notin L$

Example: $L_2 = \{a^n b^n | n \geq 0\}$

Assume for the sake of contradiction that L_2 is regular. Let $k > 0$ be the constant of the PL for L.

Select $n > k$, and $w = a^n b^n$. The property we select is that the number of a's is equal to the number of b's.

Where is the pumping window? That is, where are the admissible pumping decompositions?

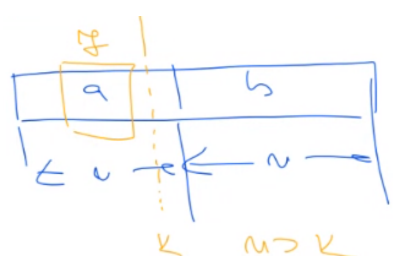

Figure 6: Pumping window diagram

So, $y = a^j | j > 0$. Now, we pump up once to obtain $w_1 = a^{n+j} b^n$. But since $n + j \neq n$, $w_i \notin L$ and so L is not regular.