MTL783: Theory of Computation 1st Semester, 2025-2026

Lecture 10 — 28/08/2025

Lecturer: Prof Minati De *Scribe: Team 10*

Scribed by:

1. Abhinav Nimkar (2022MT11943)
2. Chintada Srinivasa (2022MT11924)
3. Mukul Sahu (2022MT11939)
4. Piyush Abnave (2022MT11289)
5. Priyansh (2022MT11954)

Overview:

In the previous lecture, we learnt about Pumping lemma and looked at several examples of it's application.

In this lecture, we will look at some more examples and then learn about Context-Free Grammar for a more powerful model of computation.

Problem: Prove that

$$L = \{a^n b^n : n \neq l\}$$

is not regular.

Proof: We argue by contradiction using the Pumping Lemma.

Suppose L is regular. Then, by the Pumping Lemma, there exists a pumping length p such that every string $w \in L$ with $|w| \geq p$ can be decomposed as

$$w = xyz,$$

satisfying the following conditions:

- $|xy| \leq p$
- $|y| > 0$
- For all $i \geq 0$, $xy^i z \in L$

Now consider the string

$$w = a^p b^{p!+p} \in L.$$

Since $|xy| \leq p$, the substring y consists entirely of a's. Let

$$y = a^k, \quad \text{where } 1 \leq k \leq p.$$

On pumping y, we obtain

$$xy^i z = a^{p-k}(a^k)^i b^{p!+p} = a^{p-k+ki} b^{p!+p}.$$

Thus, the number of a's becomes

$$p - k + ki = p + (i-1)k,$$

while the number of b's remains fixed at $p! + p$.

To remain in L, the counts of a's and b's must always differ. However, if we choose

$$i = \frac{p!}{k} + 1,$$

then

$$p - k + k\left(\frac{p!}{k} + 1\right) = p! + p,$$

so the number of a's equals the number of b's.

This yields the string

$$xy^i z = a^{p!+p} b^{p!+p},$$

which does *not* belong to L (since here $n = l$).

This contradicts the Pumping Lemma condition.

$$\therefore \ L \text{ is not regular.}$$

Context Free Grammar:

Consider the language

$$L = \{a^n b^n : n \geq 0\}$$

The given language cannot be generated using a DFA or a NFA. As such, we have the need to generate a more powerful model of computation. To help us design it, we will be studying context free grammar.

Definition 1. *Context Free Grammar (CFG) is a quadruple $G = (V, \Sigma, S, R)$ such that*

1. *V is a finite set. Elements of V are called variables.*
2. *Σ is a finite set. Elements of Σ are called terminals*

3. *$S \in V$ is called the start variable*

4. *R is a finite set. Elements of R are called rules.*

5. $V \cap \Sigma = \emptyset$

In context free grammar every rule is of the form:

$$A \to x$$

where $A \in V$ and $x \in \{V \cup \Sigma\}^*$

Example

Consider grammar $G = (V, \Sigma, S, R)$ with

$$V = \{S\}, \Sigma = \{a, b\}, R = \{S \to aS|Sb|a|b|\lambda\}$$

We can see that G is a context free grammar. The language that is generated by G is of the form:

$$L(G) = \{w : S \Rightarrow^* w\} = \{a^m b^n : m, n \geq 0\}$$

Proof. :

Claim 1: Any string generated by G is in $L(G)$.
Since in all rules, a is generated on the left of S and b is generated on the right of S, it can be proven by induction that any intermediate string generated by the grammar is of the form

$$a^m S b^n : m, n \geq 0$$

and consequently, the final string generated is of the form $a^m b^n : m, n \geq 0$. Therefore, any string generated by grammar G is in $L(G)$

Claim 2: Any string in $L(G)$ can be generated by G.
Consider string $a^m b^n \in L(G)$. This can be generated from G using the following sequence of rules:

$$S \Rightarrow aS \Rightarrow ... \Rightarrow a^m S \Rightarrow a^m Sb \Rightarrow ... \Rightarrow a^m Sb^n \Rightarrow a^m b^n$$

Thus, the language generated by G is $L(G)$. □

Definition 2. *A language L is a context free language if $\exists$ context free grammar $G = (V, \Sigma, S, R)$ such that $L = L(G)$, that is, G generates the language L.*

Definition 3. *A context free grammar G is a left linear grammar if all rules are of the form $A \to Bx, A \to \lambda$ where $A, B \in V, x \in \Sigma^*$*

Definition 4. *A context free grammar G is a right linear grammar if all rules are of the form $A \to xB, A \to \lambda$ where $A, B \in V, x \in \Sigma^*$*

Examples of Context Free Grammars:

Example 1

Consider the language:

$$L = \{a^n b^n : n \geq 0\}.$$

We present two CFG styles: one that generates only strings with $n > 0$ and one that also allows $n = 0$.

Grammar for $n > 0$:

$$G_1 = (\{S\}, \{a, b\}, S, R_1), \qquad R_1 = \{\, S \to aSb \mid S \to ab \,\}.$$

Using these rules, every derivation ends in the terminal production $S \to ab$, so for any $n > 0$ we have:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow \cdots \Rightarrow a^{n-1}Sb^{n-1} \Rightarrow a^{n-1}(ab)b^{n-1} = a^n b^n.$$

Thus G_1 generates all strings $a^n b^n$ with $n > 0$.

Grammar for $n \geq 0$:

$$G_0 = (\{S\}, \{a, b\}, S, R_0), \qquad R_0 = \{\, S \to aSb \mid S \to \lambda \,\}.$$

The rule $S \to \lambda$ additionally yields the empty string (the case $n = 0$).

Example 2

Consider the language:

$$L = \{0^n 1^n; n \geq 0\}^c,$$

the complement of $\{0^n 1^n; n \geq 0\}$. That is, L consists of all binary strings that are *not* of the form $0^n 1^n$.

This complement can be partitioned into three cases:

1. $0^m 1^n$ where $m > n$.
2. $0^m 1^n$ where $m < n$.
3. Any binary string that contains the substring "10".

Grammar rules:

- For case (1):

$$S \to 0S1 \quad | \quad 0S \quad | \quad 0$$

- For case (2):

$$S \to 0S1 \quad | \quad S1 \quad | \quad 1$$

- For case (3):

$$S \to T10T$$

$$T \to 0T \quad | \quad T0 \quad | \quad 1T \quad | \quad T1 \quad | \quad \lambda$$

Union of all cases:

$$S \ \to \ S_1 \ | \ S_2 \ | \ S_3$$

$$S_1 \ \to \ 0S_1 1 \ | \ 0S_1 \ | \ 0$$

$$S_2 \ \to \ 0S_2 1 \ | \ S_2 1 \ | \ 1$$

$$S_3 \ \to \ T10T$$

$$T \ \to \ 0T \ | \ T0 \ | \ 1T \ | \ T1 \ | \ \lambda$$

Result: This grammar generates exactly the complement of $\{0^n 1^n : n \geq 0\}$, i.e., all binary strings that are not of the form $0^n 1^n$.

Now we see how we can convert any DFA corresponding to a regular language to a Context Free Grammar, showing that Context Free Grammar and Context Free Language is more powerful model/ tool than DFA, NFA and Regular Language.

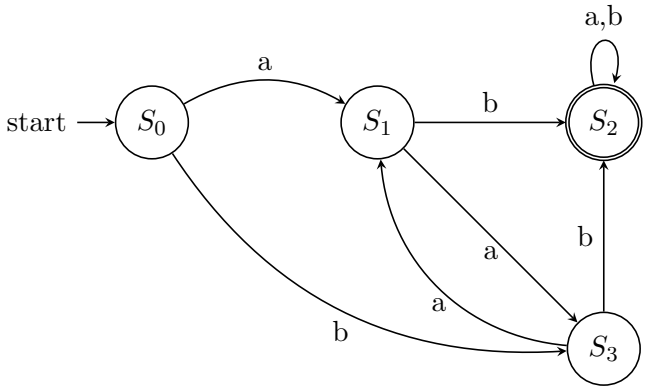
DFA to Context Free Grammar Conversion

Let the DFA be $M = (Q, \Sigma, \delta, q_0, F)$
We construct a grammar $G = (V, \Sigma, S, R)$ as follows:

1. Corresponding to each state q_i in DFA, we make an equivalent variable S_i of the CFG.
2. Make the variable S_0 corresponding to q_0 (starting state) as the start variable of the CFG.
3. For each variable S_j corresponding to final state q_j of the DFA, add the rule $S_j \to \lambda$.
4. If $\delta(q_i, a) = q_j$ in DFA, add the rule $S_i \to aS_j$ in the CFG.

Also this newly constructed Context Free Grammar, G generates the exact regular language, L as generated by the DFA, M.

Example:



$M = (Q, \Sigma, \delta, q_0, F)$ with, $Q = (q_0, q_1, q_2, q_3)$, $\Sigma = (a, b)$, $F = (q_2)$

The Corresponding CFG $G = (V, \Sigma, S, R)$ with, $V = (S_0, S_1, S_2, S_3)$, $\Sigma = (a, b)$, $S = S_0$,
And the corresponding rules R of the CFG as follows

$$S_0 \to aS_1 \mid bS_3$$
$$S_1 \to bS_2 \mid aS_3$$
$$S_2 \to aS_2 \mid bS_2 \mid \lambda$$
$$S_3 \to bS_2 \mid aS_1$$

It can be easily seen that this grammar generates exactly the same language as the DFA.

Observation

Thus, **any DFA can be converted into an equivalent CFG.** This means that every regular language can also be described by a CFG.

Conclusion

- Context-Free Grammars can generate **all regular languages**.
- In addition, they can generate many **non-regular languages** such as $\{a^n b^n \mid n \geq 0\}$.
- Therefore, CFGs are strictly more powerful than DFAs/NFAs.

Formally:

$$\text{Regular Languages} \subset \text{Context-Free Languages}$$