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

Lecture 2 — 28/07/2025

Lecturer: Prof Minati De *Scribe: 2*

Scribed by:

1. Abhishek Singh (2022MT11934)
2. Adarsh Singh (2022MT11285)
3. Mehta Maalav Pranav (2022MT11265)
4. Tirth Golwala (2022MT11967)
5. Vagesh Mahajan (2022MT11260)

1 Overview

In the last lecture, we were introduced to the course and learned basic concepts such as alphabet (a set of symbols), symbol, string (a sequence of symbols), and the length of a string.

In this lecture, we build on those ideas and define what a language is : a set of strings formed using an alphabet. We cover important operations on strings, including concatenation, reverse, and length, and see how these behave. We also introduce the special sets Σ^*, Σ^+, and give examples of finite and infinite languages.

Later, we discuss how to describe languages using grammars. A grammar is a set of rules that define how strings can be generated. We learn the formal definition of a grammar, how derivations work, and see several examples of grammars generating specific types of languages, like $a^n b^n$ and $a^n b^{n+1}$.

2 Languages

We begin with a finite, nonempty set Σ of symbols, called the *alphabet*.

Definition 2.1 (String)**.** *A* string *(or* word*) on Σ is a finite sequence of symbols from Σ. The* empty string*, denoted λ, has length zero.*

If $\Sigma = \{a, b\}$, then for example $w = \texttt{abaaa}$ is a string on Σ.

Definition 2.2 (Concatenation)**.** *Given strings $w = a_1 a_2 \cdots a_n$ and $v = b_1 b_2 \cdots b_m$, their* concatenation *is*

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

Definition 2.3 (Reverse and Length)**.** *The* reverse *of w, denoted w^R, is the string with symbols of w in reverse order. The* length *of w, denoted $|w|$, is the number of symbols in w. We have $|\lambda| = 0$ and for all w,*

$$|\lambda w| = |w\lambda| = |w|.$$

A fundamental property is:

$$|uv| = |u| + |v|.$$

Example 2.1 (Proof of length additivity)**.** *Show that $|uv| = |u| + |v|$ for any strings u, v. Use the recursive definition:*

$$|a| = 1, \quad |wa| = |w| + 1.$$

The proof proceeds by induction on $|v|$.

Next, for any string w and integer $n \geq 0$, define

$$w^n = \underbrace{ww \cdots w}_{n \text{ times}}, \quad w^0 = \lambda.$$

Definition 2.4. *For alphabet Σ,*

$$\Sigma^* = \{\textit{all strings over } \Sigma\},$$

$$\Sigma^+ = \Sigma^* \setminus \{\lambda\}.$$

A language *L over Σ is any subset $L \subseteq \Sigma^*$.*

Example 2.2. *If $\Sigma = \{a, b\}$ then*

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, \ldots\}.$$

The finite language $\{a, aa, aab\}$ is an example, and also

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

is an infinite language.

Language operations: union, intersection, complement (wrt Σ^*), concatenation

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\},$$

reverse $L^R = \{w^R : w \in L\}$, and closures

$$L^* = \bigcup_{n \geq 0} L^n, \quad L^+ = \bigcup_{n \geq 1} L^n.$$

Example 2.3. *Let $L = \{a^n b^n : n \geq 0\}$. Then*

$$L^2 = \{a^n b^n a^m b^m : n, m \geq 0\}.$$

Reverse gives

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

3 Grammars

To describe languages, we use grammars.

Definition 3.1 (Grammar)**.** *A* grammar *is a quadruple $G = (V, T, S, P)$ where:*

- *V is a finite set of objects called variables,*
- *T is a finite set of terminal symbols,*
- *$S \in V$ is a special symbol called the start variable,*
- *P is a finite set of productions.*

It is assumed that the sets V and T are nonempty and disjoint.

The *production rules* define how the grammar transforms one string into another, thereby defining a language associated with the grammar.

Each production rule is of the form:

$$x \to y,$$

where $x \in (V \cup T)^+$ and $y \in (V \cup T)^*$.

Given a string $w = uxv$, the production $x \to y$ is applicable and produces the new string $z = uyv$. This is denoted as:

$$w \Rightarrow z.$$

We say that w *derives* z, or z is derived from w.

Successive derivations are written as:

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

and the notation:

$$w_1 \Rightarrow^* w_n$$

means that w_1 derives w_n in zero or more steps.

Definition 3.2. *Let $G = (V, T, S, P)$ be a grammar. Then the* ***language generated by*** *G is defined as:*

$$L(G) = \{w \in T^* : S \Rightarrow^* w\}.$$

That is, $L(G)$ consists of all terminal strings that can be derived from the start symbol S using the production rules in P.

Example 3.1. *Consider the grammar:*

$$G = (\{S\}, \{a, b\}, S, P),$$

with production rules P given by:

$$S \to aSb, \quad S \to \lambda.$$

We can derive strings using this grammar as follows:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb.$$

Thus,

$$S \Rightarrow^* aabb.$$

The grammar G completely defines the language $L(G)$, though it may not always be easy to explicitly describe. In this case, we can conjecture:

$$L(G) = \{a^n b^n : n \geq 0\},$$

and it is straightforward to prove this by induction.

We observe that the rule $S \to aSb$ is recursive. Let us prove by induction that all sentential forms (intermediate steps in derivation) are of the form:

$$w_i = a^i S b^i. \tag{1.7}$$

Assume that (1.7) holds for all sentential forms w_i of length $2i+1$ or less. Then applying the rule $S \to aSb$ yields:

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

which is of the same form as (1.7), now with $i+1$. Since the base case $i = 1$ is trivially true (aSb), by induction the form holds for all i.

To produce a sentence (a terminal string), we finally apply the rule $S \to \lambda$. This gives:

$$S \Rightarrow^* a^n S b^n \Rightarrow a^n b^n.$$

Thus, the grammar G derives only strings of the form $a^n b^n$.

We must also show that every string of the form $a^n b^n$ can be derived by G. This is evident, since we can apply $S \to aSb$ exactly n times, followed by $S \to \lambda$.

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow \cdots \Rightarrow a^n S b^n \Rightarrow a^n b^n.$$

Example 3.2. *We wish to construct a grammar that generates the language:*

$$L = \{a^n b^{n+1} : n \geq 0\}.$$

We build upon the idea from the earlier example that generated $a^n b^n$, and simply add one extra `b`*. This can be achieved by defining:*

$$G = (\{S, A\}, \{a, b\}, S, P)$$

with productions:

$$\begin{aligned} S &\to Ab \\ A &\to aAb \\ A &\to \lambda \end{aligned}$$

1. Any string derived by G is in L

The derivation always starts with $S \to Ab$, guaranteeing one final `b`*. Each application of $A \to aAb$ adds one* `a` *and one more* `b`*, so after n such steps followed by $A \to \lambda$, we obtain $a^n b^{n+1}$.*

2. Any string in L can be derived

Given any $n \geq 0$, we can:

- *Apply $A \to aAb$ n times to get $a^n A b^n$,*
- *Then apply $A \to \lambda$,*
- *The outermost rule $S \to Ab$ gives one more* `b`*,*

yielding:

$$S \Rightarrow Ab \Rightarrow a^n A b^n b \Rightarrow a^n b^{n+1}.$$

The grammar G generates exactly:

$$L(G) = \{a^n b^{n+1} : n \geq 0\}.$$

Example 3.3. *Let $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of occurrences of* `a` *and* `b` *in a string w, respectively. We define:*

$$L = \{w \in \Sigma^* : n_a(w) = n_b(w)\}.$$

The grammar $G = (\{S\}, \{a, b\}, S, P)$ has the following productions:

$$\begin{aligned} S &\to SS \\ S &\to aSb \\ S &\to bSa \\ S &\to \lambda \end{aligned}$$

We aim to show that:

$$L(G) = \{w \in \Sigma^* : n_a(w) = n_b(w)\}.$$

Observation: All strings generated by G are in L. This is straightforward. All production rules that introduce an `a` *also introduce a* `b`*. Hence, every sentential form — and therefore every terminal string derived — must contain equal numbers of* `a`*s and* `b`*s.*

Thus:

$$\forall w \in L(G), \quad n_a(w) = n_b(w).$$

Let us examine how to construct such a derivation based on the structure of $w \in L$:

Case 1: $w = aw_1b$ *If w begins with* `a` *and ends with* `b`*, then it can be derived via:*

$$S \Rightarrow aSb \Rightarrow aw_1b$$

provided $w_1 \in L$. The number of `a`*s and* `b`*s in w_1 must be equal.*

Case 2: $w = bw_1a$ *Similarly, we can derive:*

$$S \Rightarrow bSa \Rightarrow bw_1a$$

provided $w_1 \in L$.

Case 3: $w = w_1w_2$ *If w starts and ends with the same letter (e.g.,* `a...a` *or* `b...b`*), we use a counting argument: define a running sum where we add $+1$ for* `a` *and -1 for* `b`*. Since $n_a(w) = n_b(w)$, the total sum is 0, and the prefix-sum must cross 0 at some point before the end. This gives us:*

$$w = w_1w_2 \quad \textit{with} \quad n_a(w_1) = n_b(w_1),\ n_a(w_2) = n_b(w_2)$$

Thus, we can derive:

$$S \Rightarrow SS \Rightarrow w_1S \Rightarrow w_1w_2$$

Let $P(n)$ be the statement: every string $w \in L$ of length $\leq 2n$ can be derived from S.

Base Case: *$n = 0$ The only string of length 0 is λ, which satisfies $n_a = n_b = 0$, and $S \Rightarrow \lambda$.*

Inductive Step: *Assume $P(n)$ holds. Let $w \in L$ with $|w| = 2n + 2$.*

- *If $w = aw_1b$ or $w = bw_1a$, then $w_1 \in L$, $|w_1| = 2n$, so by the inductive hypothesis $S \Rightarrow^* w_1$. Then:*

$$S \Rightarrow aSb \Rightarrow^* aw_1b = w \quad \textit{or} \quad S \Rightarrow bSa \Rightarrow^* bw_1a = w$$

- *If w starts and ends with the same letter, use the prefix-sum method to split $w = w_1w_2$, where both $w_1, w_2 \in L$, and $|w_1|, |w_2| \leq 2n$. By the inductive hypothesis:*

$$S \Rightarrow SS \Rightarrow^* w_1S \Rightarrow^* w_1w_2 = w$$

Thus, $P(n + 1)$ holds, and by induction, every $w \in L$ can be derived.

$$L(G) = \{w \in \{a, b\}^* : n_a(w) = n_b(w)\}$$

Grammar Equivalence

Normally, a given language has many grammars that generate it. Even though these grammars are different, they are equivalent in some sense. We say that two grammars G_1 and G_2 are **equivalent** if they generate the same language, that is, if

$$L(G_1) = L(G_2).$$

As we will see later, it is not always easy to determine whether two grammars are equivalent.

Example 3.4. *Consider the grammar* $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$*, with* P_1 *consisting of the productions:*

$$S \to aAb \mid \lambda$$
$$A \to aAb \mid \lambda$$

Here we introduce a convenient shorthand notation in which several production rules with the same left-hand side are written on the same line, with alternative right-hand sides separated by $|$*. For example:*

$$S \to aAb \mid \lambda$$

is equivalent to the two separate productions:

$$S \to aAb \quad \textit{and} \quad S \to \lambda$$

This grammar is ***equivalent*** *to the grammar in Example 2.1*

The equivalence is shown by proving that:

$$L(G_1) = \{a^n b^n : n \geq 0\}$$

Recall that the grammar in Example 2.1 is:

$$G = (\{S\}, \{a, b\}, S, P), \quad \textit{where}$$
$$S \to aSb \quad \textit{and} \quad S \to \lambda$$

which generates the language:

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

We now show that the grammar G_1 *also generates the same language.*

$$L(G_1) = \{a^n b^n : n \geq 0\}$$

(1) Every string derived from G_1 ***is in*** L***:***

Each use of aAb *in* S *and in* A *adds one* a *to the left and one* b *to the right. Eventually,* $A \to \lambda$*, so the number of* a*'s and* b*'s will always match. Thus:*

$$\forall w \in L(G_1), \quad w = a^n b^n$$

(2) Every string in L ***can be derived by*** G_1***:***

We use induction on n*:*

Base Case: $n = 0$*:* $S \to \lambda$

Inductive Step: *Assume true for* $n = k$*. For* $n = k + 1$*, do:*

$$S \Rightarrow aAb$$

and apply the inductive hypothesis inside A *to get* $a^k b^k$*, so total string becomes:*

$$aa^k b^k b = a^{k+1} b^{k+1}$$

Hence, G_1 *can derive all* $a^n b^n$*, and:*

$$L(G_1) = \{a^n b^n : n \geq 0\} = L(G)$$

References

[1] Peter Linz, *An Introduction to Formal Languages and Automata*, 6th Edition, Jones & Bartlett Learning, 2016.