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

Lecture 11 — 01/09/2025

Lecturer: Prof Minati De *Scribe: Team 11*

Scribed by:

1. Hemant Ramgaria (2022MT11854)
2. Sumit Sonowal (2022MT11296)
3. Lakshaya Jain (2022MT11933)
4. Ravi Kumawat (2022MT12027)
5. Praveen Lakhara (2022MT11280)

1 Overview

In the previous lecture, we learned about Context-Free Grammars and their basic definitions. We saw examples of CFGs and how they are more powerful than regular languages.

In this lecture, we will cover how to represent CFGs, derivation trees, leftmost and rightmost derivations, ambiguity in grammars, and Chomsky Normal Form with conversion procedures.

2 Representation of Context-Free Grammar

2.1 Standard Notation

A Context-Free Grammar is represented as $G = (V, \Sigma, S, P)$ where:

- V is a finite set of **variables** (non-terminals)
- Σ is a finite set of **terminals**
- $S \in V$ is the **start symbol**
- P is a finite set of **productions** (rules)
- $V \cap \Sigma = \emptyset$

2.2 Production Rules Format

Each production rule has the form: $A \to \alpha$ where

- $A \in V$ (left-hand side is a variable)
- $\alpha \in (V \cup \Sigma)^*$ (right-hand side is a string of variables and terminals)

2.3 Compact Representation

Multiple productions with the same left-hand side can be written compactly:

$$A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_k$$

Example: Grammar for balanced parentheses:

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

where P contains:

$$S \to (S) \mid SS \mid \lambda$$

3 Parse Trees (Derivation Trees)

3.1 Definition and Properties

Definition 1. *A* **parse tree** *for a CFG $G = (V, \Sigma, S, P)$ is a tree with the following properties:*

1. *The root is labeled with the start symbol S*
2. *Each internal node is labeled with a variable from V*
3. *Each leaf is labeled with a terminal from Σ or λ*
4. *If an internal node is labeled A and has children labeled $X_1, X_2, \ldots, X_k$ (left to right), then $A \to X_1 X_2 \cdots X_k \in P$*
5. *If a leaf is labeled λ, it must be the only child of its parent*

3.2 Parse Tree Example

Let us define a context-free grammar $G = (V, \Sigma, P, S)$ as follows:

- $V = \{S\}$ — the set of variables (non-terminals)
- $\Sigma = \{a, b\}$ — the set of terminals
- S — the start symbol
- P — the set of productions:

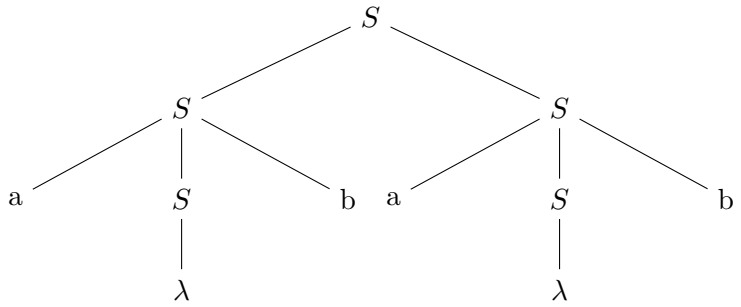
$$S \to SS \mid aSb \mid \lambda$$

Question: Can we generate the string *abab* using this grammar?

Answer: Yes. Here's the derivation sequence:

$$
\begin{aligned}
S &\Rightarrow SS \quad (\text{Using } S \to SS) \\
&\Rightarrow aSb\,aSb \quad (\text{Apply } S \to aSb \text{ to both } S\text{'s}) \\
&\Rightarrow ab\,ab \quad (\text{Apply } S \to \lambda \text{ to both inner } S\text{'s}) \\
&= abab
\end{aligned}
$$

Parse Tree for the string *abab***:**



Thus, the grammar successfully generates the string *abab*.

4 Leftmost and Rightmost Derivations

4.1 Derivation Process

Definition 2. *A* **derivation** *is a sequence of applications of production rules:*

$$S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n = w$$

where each $\alpha_i \Rightarrow \alpha_{i+1}$ *follows from applying some production rule.*

4.2 Leftmost Derivation

Definition 3. *A* **leftmost derivation** *is a derivation where at each step, we replace the leftmost variable in the current sentential form.*

Example: Leftmost derivation of $\mathrm{id} + \mathrm{id} \times \mathrm{id}$:

$$
\begin{aligned}
E &\Rightarrow_{\mathrm{lm}} E + E && (1)\\
&\Rightarrow_{\mathrm{lm}} \mathrm{id} + E && (2)\\
&\Rightarrow_{\mathrm{lm}} \mathrm{id} + E \times E && (3)\\
&\Rightarrow_{\mathrm{lm}} \mathrm{id} + \mathrm{id} \times E && (4)\\
&\Rightarrow_{\mathrm{lm}} \mathrm{id} + \mathrm{id} \times \mathrm{id} && (5)
\end{aligned}
$$

4.3 Rightmost Derivation

Definition 4. *A* **rightmost derivation** *is a derivation where at each step, we replace the rightmost variable in the current sentential form.*

Example: Rightmost derivation of $\mathrm{id} + \mathrm{id} \times \mathrm{id}$:

$$\begin{aligned} E &\Rightarrow_{\mathrm{rm}} E + E && (6)\\ &\Rightarrow_{\mathrm{rm}} E + E \times E && (7)\\ &\Rightarrow_{\mathrm{rm}} E + E \times \mathrm{id} && (8)\\ &\Rightarrow_{\mathrm{rm}} E + \mathrm{id} \times \mathrm{id} && (9)\\ &\Rightarrow_{\mathrm{rm}} \mathrm{id} + \mathrm{id} \times \mathrm{id} && (10) \end{aligned}$$

5 Ambiguity in Context-Free Grammars

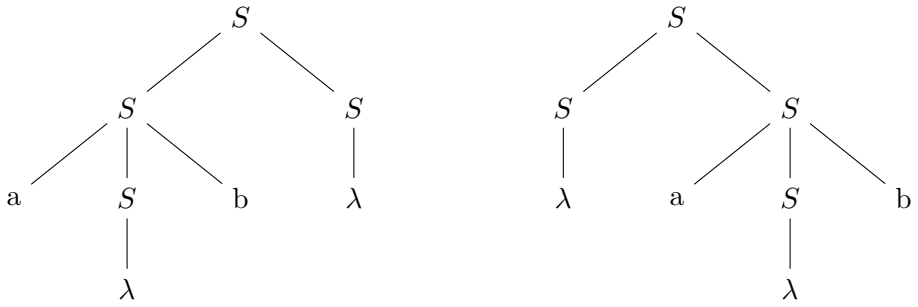
5.1 Definition of Ambiguity

Definition 5. *A context-free grammar G is said to be* ***ambiguous*** *if there exists a string $w \in L(G)$ such that there are two or more distinct parse trees corresponding to w.*

Example 1: Consider the grammar:

$$S \to SS \mid aSb \mid \lambda$$

For the string ab, we can construct two distinct parse trees:



Thus, the grammar is ambiguous.

5.2 Ambiguity in Arithmetic Expressions

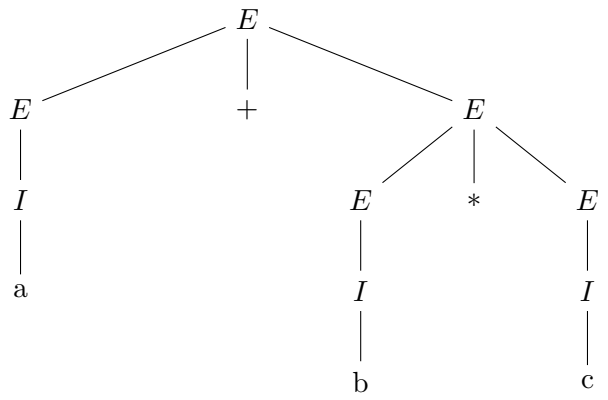
Consider the grammar:

$$E \to I \mid E + E \mid E * E \mid (E), \qquad I \to a \mid b \mid c$$

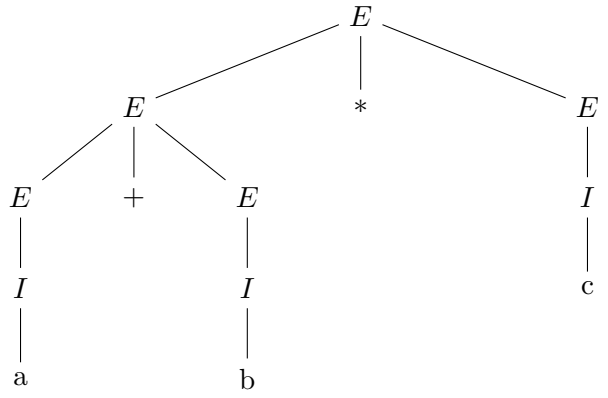
Check the string $a + b * c$.

There are two possible parse trees:

Parse Tree 1:



Parse Tree 2:



Therefore, the grammar is ambiguous.

5.3 Removing Ambiguity

To remove ambiguity in expressions involving operators like '+' and '*', we can introduce new variables to enforce **precedence** and **associativity**.

We define the grammar $G = (V, \Sigma, P, S)$ as follows:

- $V = \{E, T, M, I\}$ — set of variables (non-terminals)
- $\Sigma = \{a, b, c, +, *, (,)\}$ — set of terminals
- $S = E$ — the start symbol

- P — the set of production rules:

$$E \to E + T \mid T$$
$$T \to T * M \mid M$$
$$M \to I \mid (E)$$
$$I \to a \mid b \mid c$$

This grammar ensures:

- **Operator precedence:** `*` has higher precedence than `+`.
- **Left associativity:** Both `+` and `*` are left-associative.

Note: This grammar is unambiguous because each string has a unique parse tree according to operator precedence and associativity.

Definition: A context-free language L is said to be **unambiguous** if there exists a context-free grammar for L that is not ambiguous.

5.4 Inherent Ambiguity

Definition 6. *A language L is said to be **inherently ambiguous** if every context-free grammar that generates L is ambiguous.*

Important Note:

- Not all ambiguous grammars define inherently ambiguous languages.
- A language is inherently ambiguous if it is **impossible** to construct an unambiguous grammar for it.

Is it possible to make all ambiguous grammars unambiguous? **No.** Some languages are inherently ambiguous.

Example: Define the following two context-free languages:

$$L_1 = \{a^m b^n c^n \mid m \geq 0, n \geq 0\}$$
$$L_2 = \{a^m b^m c^n \mid m \geq 0, n \geq 0\}$$

These can be generated using unambiguous grammars:

- For L_1:

$$S \to aS \mid T$$
$$T \to bTc \mid \lambda$$

- For L_2:

$$S \to Sc \mid T$$
$$T \to aTb \mid \lambda$$

Both grammars are unambiguous individually.

Now consider the union:

$$L = L_1 \cup L_2 = \{a^m b^n c^n \mid m, n \geq 0\} \cup \{a^m b^m c^n \mid m, n \geq 0\}$$

Define a grammar for L:

$$U \to S_1 \mid S_2$$

Where:

- S_1 generates L_1 using its own rules.
- S_2 generates L_2 using its own rules.

Now, consider the string $a^m b^m c^m$. This string belongs to both L_1 and L_2:

- In L_1, with $n = m$.
- In L_2, with $n = m$.

Thus, there are two distinct derivations for this string:

- One from S_1.
- Another from S_2.

Hence, the string has two parse trees — the grammar is ambiguous.

This ambiguity cannot be removed because any grammar that generates L must include both L_1 and L_2, and strings like $a^m b^m c^m$ will always belong to both.

Conclusion: The language

$$L = \{a^m b^n c^n \mid m, n \geq 0\} \cup \{a^m b^m c^n \mid m, n \geq 0\}$$

is **inherently ambiguous**.

This result was **proven in 1961**.

6 Chomsky Normal Form

6.1 Definition

Definition 7. *A context-free grammar is in* **Chomsky Normal Form (CNF)** *if every production is of one of the following forms:*

1. $A \to BC$ *where* $A, B, C \in V$ *and* $B \neq S$, $C \neq S$
2. $A \to a$ *where* $A \in V$ *and* $a \in \Sigma$ *(terminal production)*
3. $S \to \lambda$ *(only if* $\lambda \in L(G)$ *and* S *is the start symbol)*

6.2 Importance of CNF

- Every context-free language can be generated by a CFG in CNF
- CNF is useful for proving theorems about CFLs
- Parsing algorithms often work with CNF grammars
- CNF grammars have a **binary tree structure** for parse trees

6.3 Theorem and Conversion Procedure (Worked Example)

Theorem 8. *Any context-free language is generated by a CFG in Chomsky Normal Form.*

6.4 Conversion Procedure (Example from Class Notes)

Given grammar:

$$G = (V, \Sigma, P, S), \quad \Sigma = \{0, 1\}, \quad V = \{A, B\}, \quad S = A$$
$$A \to BAB \mid B \mid \lambda, \quad B \to 00 \mid \lambda$$

We now convert this grammar into Chomsky Normal Form.

Step 1: Introduce new start symbol
Ensure that the start symbol does not appear on the right-hand side of any production. Introduce S_1 such that:

$$S_1 \to A \mid \lambda, \quad V = \{S_1, A, B\}$$

Step 2: Eliminate λ-productions
Remove all productions of the form $A \to \lambda$ where A is not the start symbol.

Given:

$$A \to BAB \mid B \mid \lambda, \quad B \to 00 \mid \lambda$$

Identify nullable variables: A, B.

Modify productions to account for nullable variables. For $A \to BAB$:

$$A \to BAB \mid AB \mid BA \mid BB$$

And for $B \to 00 \mid \lambda$, we keep $B \to 00$ and drop λ since it's not needed anymore.

Final updated productions:

$$A \to BAB \mid AB \mid BA \mid BB \mid B$$
$$B \to 00$$
$$S_1 \to A \mid \lambda$$

Step 3: Eliminate unit productions
Eliminate rules of the form $A \to B$ where B is a single variable. For example:

$$A \to B \quad \text{(unit production)}$$

Since $B \to 00$, replace $A \to B$ with $A \to 00$.

$$A \to BAB \mid AB \mid BA \mid BB \mid 00$$

Step 4: Convert long productions to binary form
For any rule of the form $A \to X_1X_2X_3 \cdots$ where length ≥ 3, break it down.

Example:

$$A \to BAB \quad \text{(length} = 3\text{)}$$

Introduce new variables:

$$A \to BA_1, \quad A_1 \to AB$$

Update the productions:

$$A \to BA_1 \mid AB \mid BA \mid BB \mid 00$$
$$A_1 \to AB$$

Now all productions are of the form:

- $A \to BC$ (two variables)
- $A \to a$ (a terminal)
- $S \to \lambda$ (if allowed)

Final CNF Grammar:

$$S_1 \to A \mid \lambda, \quad A \to BA_1 \mid AB \mid BA \mid BB \mid 00, \quad B \to 00, \quad A_1 \to AB$$

7 Summary

In this lecture, we covered:

- **CFG Representation:** Standard notation and production rule formats
- **Parse Trees:** Definition, properties, and examples
- **Derivations:** Leftmost and rightmost derivations with examples
- **Ambiguity:** Definition, examples, and resolution techniques
- **Chomsky Normal Form:** Definition, conversion algorithm, and applications

These concepts form the foundation for understanding the structure and parsing of context-free languages, which are essential in compiler design and formal language theory.

References

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