# CS 348: Module 02: Compilers

## Overview of Syntax Analysis or Parsing

March 22, 2025

## Module Objectives





- Different Type of Grammer
- Understand Parsing Fundamental
- Understand Top down Parsing
- Understand Bottom up parsing

# Grammar

# Grammar

$G =< T, N, S, P >$ is a (context-free) grammar where:

| | | |
|---|---|---|
| $T$ | : | Set of terminal symbols |
| $N$ | : | Set of non-terminal symbols |
| $S$ | : | $S \in N$ is the start symbol |
| $P$ | : | Set of production rules |

Every production rule is of the form: $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$.

# Recursively Enumerable Grammar (Type-0)

**Definition:** Unrestricted productions of the form:

$$\alpha \to \beta \tag{1}$$

where $\alpha, \beta \in (V \cup \Sigma)^*$ and $\alpha$ contains at least one non-terminal.
**Example:**

$$S \to aS \mid aSb \mid \varepsilon$$

**Language Generated:** Strings with equal numbers of *a*s and *b*s, allowing arbitrary order.

# Context-Sensitive Grammar (Type-1)

**Definition:** Productions are of the form:

$$\alpha A\beta \to \alpha\gamma\beta \tag{2}$$

where $|\gamma| \geq |A|$ to ensure non-decreasing length.

**Example:**

$$\begin{aligned} S &\to aSBC \mid abc \\ CB &\to BC \\ aB &\to ab \\ bB &\to bb \\ bC &\to bc \end{aligned}$$

**Language Generated:** $L = \{a^n b^n c^n \mid n \geq 1\}$.

# Context-Free Grammar (Type-2)





**Definition:** Each production rule is of the form:

$$A \to \alpha \tag{3}$$

where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$.

**Example:**

$$S \to aSb \mid \varepsilon$$

**Language Generated:** $L = \{a^n b^n \mid n \geq 0\}$.

# Regular Grammar (Type-3)





**Definition:** Productions follow one of the forms:

$$A \to aB \quad \text{or} \quad A \to a \tag{4}$$

where $A, B \in V$ and $a \in \Sigma$.
**Example:**

$$S \to aS \mid bS \mid \varepsilon$$

**Language Generated:** Strings over $\{a, b\}$ including the empty string.

# Example Grammar: Derivations



$G = <\{\mathbf{id}, +, *, (, )\}, \{E, T, F\}, E, P >$ where $P$ is:

$$\begin{array}{llll} 1: & E & \rightarrow & E + T \\ 2: & E & \rightarrow & T \\ 3: & T & \rightarrow & T * F \\ 4: & T & \rightarrow & F \\ 5: & F & \rightarrow & (E) \\ 6: & F & \rightarrow & \mathbf{id} \end{array}$$

Left-most Derivation of **id** * **id** + **id** \$:

$$\begin{array}{lllllll} E\ \$ & \Rightarrow & \underline{E} + T\ \$ & \Rightarrow & \underline{T} + T\ \$ & \Rightarrow & \underline{T} * F + T\ \$ \\ & \Rightarrow & \underline{F} * F + T\ \$ & \Rightarrow & \mathbf{id} * \underline{F} + T\ \$ & \Rightarrow & \mathbf{id} * \underline{\mathbf{id}} + T\ \$ \\ & \Rightarrow & \mathbf{id} * \mathbf{id} + \underline{F}\ \$ & \Rightarrow & \mathbf{id} * \mathbf{id} + \underline{\mathbf{id}}\ \$ & & \end{array}$$

Right-most Derivation of **id** * **id** + **id** \$:

$$\begin{array}{lllllll} E\ \$ & \Rightarrow & E + \underline{T}\ \$ & \Rightarrow & E + \underline{F}\ \$ & \Rightarrow & E + \underline{\mathbf{id}}\ \$ \\ & \Rightarrow & \underline{T} + \mathbf{id}\ \$ & \Rightarrow & T * \underline{F} + \mathbf{id}\ \$ & \Rightarrow & T * \underline{\mathbf{id}} + \mathbf{id}\ \$ \\ & \Rightarrow & \underline{F} * \mathbf{id} + \mathbf{id}\ \$ & \Rightarrow & \underline{\mathbf{id}} * \mathbf{id} + \mathbf{id}\ \$ & & \end{array}$$

# Parsing Fundamentals



| **Derivation** | **Parsing** | **Parser** | **Remarks** |
|---|---|---|---|
| Left-most | Top-Down | Predictive:<br>Recursive Descent,<br>LL(1) | No Ambiguity<br>No Left-recursion<br>Tool: Antlr |
| Right-most | Bottom-Up | Shift-Reduce:<br>SLR,<br>LALR(1), LR(1) | Ambiguity okay<br>Left-recursion okay<br>Tool: YACC, Bison |

# Curse or Boon 1: Left-Recursion



A grammar is left-recursive iff there exists a non-terminal $A$ that can derive to a sentential form with itself as the leftmost symbol. Symbolically,

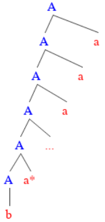
$$A \quad \Rightarrow^{+} \quad A\alpha$$

*We cannot have a recursive descent or predictive parser (with left-recursion in the grammar) because we do not know how long should we recur without consuming an input*

# Curse or Boon 1: Left-Recursion

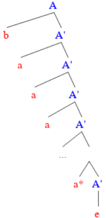
Note that, $\begin{array}{lcl} A & \rightarrow & A\alpha \\ A & \rightarrow & \beta \end{array}$ leads to:

$$\begin{array}{lclclcll} A\,\$ & \Rightarrow & A\alpha\,\$ & \Rightarrow & A\alpha\alpha\,\$ & \Rightarrow & A\alpha\alpha\alpha\,\$ & \cdots \\ & \Rightarrow & A\alpha^*\,\$ & \Rightarrow & \beta\alpha^*\,\$ & & & \end{array}$$




Removing left-recursion $\begin{array}{lclcl} A & \rightarrow & \beta A' & & \\ A' & \rightarrow & \alpha A' & | & \epsilon \end{array}$ leads to:

$$\begin{array}{lclclcll} A\,\$ & \Rightarrow & \beta A'\,\$ & \Rightarrow & \beta\alpha A'\,\$ & \Rightarrow & \beta\alpha\alpha A'\,\$ & \cdots \\ & \Rightarrow & \beta\alpha^* A'\,\$ & \Rightarrow & \beta\alpha^*\,\$ & & & \end{array}$$



# Left-Recursive Example

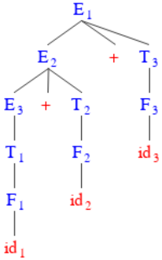
**Grammar $G_1$ before Left-Recursion Removal**

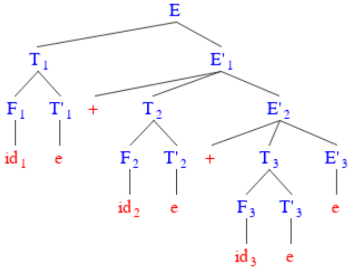
| | | | |
|---|---|---|---|
| 1: | $E$ | $\rightarrow$ | $E + T$ |
| 2: | $E$ | $\rightarrow$ | $T$ |
| 3: | $T$ | $\rightarrow$ | $T * F$ |
| 4: | $T$ | $\rightarrow$ | $F$ |
| 5: | $F$ | $\rightarrow$ | $(E)$ |
| 6: | $F$ | $\rightarrow$ | **id** |

**Grammar $G_2$ after Left-Recursion Removal**

| | | | | | |
|---|---|---|---|---|---|
| 1: | $E$ | $\rightarrow$ | $T\ E'$ | | |
| 2: | $E'$ | $\rightarrow$ | $+\ T\ E'$ | $\mid$ | $\epsilon$ |
| 3: | $T$ | $\rightarrow$ | $F\ T'$ | | |
| 4: | $T'$ | $\rightarrow$ | $*\ F\ T'$ | $\mid$ | $\epsilon$ |
| 5: | $F$ | $\rightarrow$ | $(E)$ | | |
| 6: | $F$ | $\rightarrow$ | **id** | | |

- These are syntactically equivalent. But what happens semantically?
- Can left recursion be effectively removed?
- What happens to Associativity?



# Curse or Boon 2: Ambiguous Grammar





$$\begin{array}{llll} 1: & E & \rightarrow & E + E \\ 2: & E & \rightarrow & E * E \\ 3: & E & \rightarrow & (E) \\ 4: & E & \rightarrow & \textbf{id} \end{array}$$

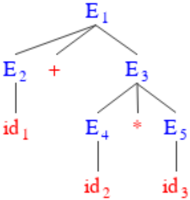
- Ambiguity simplifies. But, ...
  - Associativity is lost
  - Precedence is lost
- Can *Operator Precedence (infix → postfix)* give us a clue?

# Ambiguous Derivation of **id + id * id**



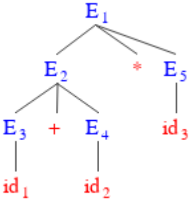
Correct derivation: * has precedence over +

$$
\begin{aligned}
E\ \$ &\Rightarrow \underline{E + E}\ \$ \\
&\Rightarrow E + \underline{E * E}\ \$ \\
&\Rightarrow E + E * \underline{\mathbf{id}}\ \$ \\
&\Rightarrow E + \underline{\mathbf{id}} * \mathbf{id}\ \$ \\
&\Rightarrow \underline{\mathbf{id}} + \mathbf{id} * \mathbf{id}\ \$
\end{aligned}
$$




Wrong derivation: + has precedence over *

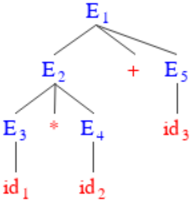
$$
\begin{aligned}
E\ \$ &\Rightarrow \underline{E * E}\ \$ \\
&\Rightarrow E * \underline{\mathbf{id}}\ \$ \\
&\Rightarrow \underline{E + E} * \mathbf{id}\ \$ \\
&\Rightarrow E + \underline{\mathbf{id}} * \mathbf{id}\ \$ \\
&\Rightarrow \underline{\mathbf{id}} + \mathbf{id} * \mathbf{id}\ \$
\end{aligned}
$$



# Ambiguous Derivation of **id * id + id**

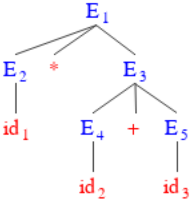
Correct derivation: * has precedence over +

$$
\begin{aligned}
E\ \$ &\Rightarrow \underline{E + E}\ \$ \\
&\Rightarrow E + \underline{\mathbf{id}}\ \$ \\
&\Rightarrow \underline{E * E} + \mathbf{id}\ \$ \\
&\Rightarrow E * \underline{\mathbf{id}} + \mathbf{id}\ \$ \\
&\Rightarrow \underline{\mathbf{id}} * \mathbf{id} + \mathbf{id}\ \$
\end{aligned}
$$




Wrong derivation: + has precedence over *

$$
\begin{aligned}
E\ \$ &\Rightarrow \underline{E * E}\ \$ \\
&\Rightarrow E * \underline{E + E}\ \$ \\
&\Rightarrow E * E + \underline{\mathbf{id}}\ \$ \\
&\Rightarrow E * \underline{\mathbf{id}} + \mathbf{id}\ \$ \\
&\Rightarrow \underline{\mathbf{id}} * \mathbf{id} + \mathbf{id}\ \$
\end{aligned}
$$



# Remove: Ambiguity and Left-Recursion

$$
\begin{array}{lccl}
1: & E & \rightarrow & E + E \\
2: & E & \rightarrow & E * E \\
3: & E & \rightarrow & (E) \\
4: & E & \rightarrow & \mathbf{id}
\end{array}
$$

Removing ambiguity:

$$
\begin{array}{lccl}
1: & E & \rightarrow & E + T \\
2: & E & \rightarrow & T \\
3: & T & \rightarrow & T * F \\
4: & T & \rightarrow & F \\
5: & F & \rightarrow & (E) \\
6: & F & \rightarrow & \mathbf{id}
\end{array}
$$

Removing left-recursion:

$$
\begin{array}{rccl}
1: & E & \rightarrow & T\ E' \\
2|3: & E' & \rightarrow & +\ T\ E' \mid \epsilon \\
4: & T & \rightarrow & F\ T' \\
5|6: & T' & \rightarrow & *\ F\ T' \mid \epsilon \\
7: & F & \rightarrow & (E) \\
8: & F & \rightarrow & \mathbf{id}
\end{array}
$$