# SYNTAX ANALYSIS

LECTURE 2

# CONTENT

- Context Free Grammar
- Derivation and Parse Tree
- Ambiguous and Unambiguous Grammar
- Removing Unambiguity
  - Removing Left Recursion
  - Left Factoring

# Context Free Grammar

$E \rightarrow E + E \mid E * E \mid ( E ) \mid - E \mid \textbf{id}$

To generate a valid string: - ( **id** + **id** * **id** )

Steps:

Rightmost Derivation

Here, rightmost Non-Terminal is replaced in every step.

E→**(E)**

E → (E+**E)**

E→ (E+E*E)

E→ (E+E***id**)

E→(E+**id*id**)

E→(**id**+**id*id**)

# Context Free Grammar

E → E + E | E * E | ( E ) | - E | **id**

To generate a valid string: - ( **id** + **id** * **id** )

Steps:

Leftmost Derivation

Here, leftmost Non-Terminal is replaced in every step.

E→**(E)**

E→(**E***E)

E→ (**E**+E*E)

E → (id+**E***E)

E → (id+id***E**)

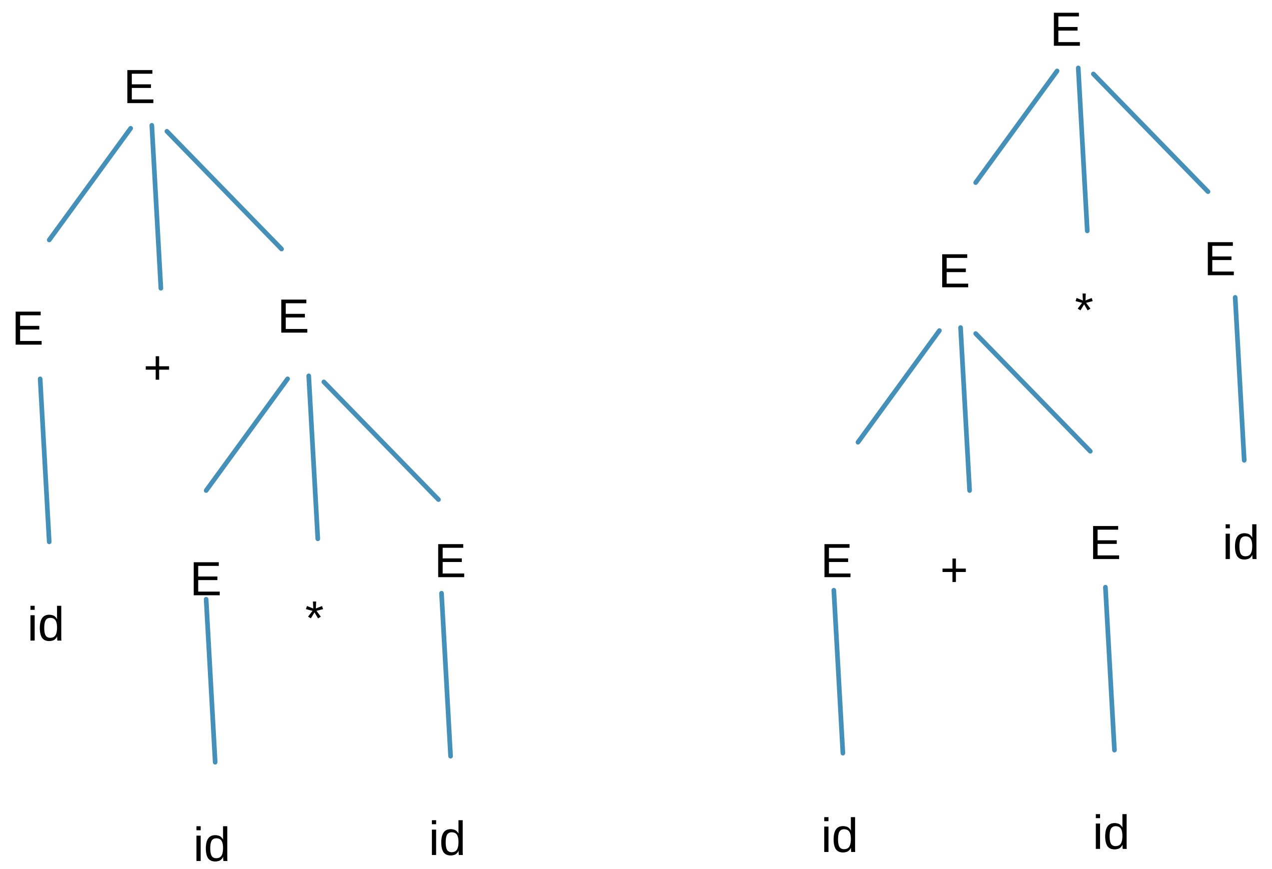
E→(**id**+**id*id**)

# Context Free Grammar

E → E + E | E * E | ( E ) | - E | **id**

To generate a valid string: - ( **id** + **id** * **id** )

Parse Trees



# Ambiguous Grammar

- Ambiguous Grammar

For a given grammar, we can generate at least one string which can be presented using more than one parse tree then such grammar is called ambiguous grammar

- Unambiguous Grammar

For a given grammar, all possible strings which can be generated using it have only one representation of Parse Tree, such grammar is called Unambiguous Grammar

# Removing Unambiguity

- Two Techniques:

1. Removing Left Recursion
2. Left Factoring

# Removing Left Recursion

- A grammar is left recursive if it has a nonterminal such that there is a derivation $A \rightarrow A\alpha$ for some string $\alpha$.
- Top-down parsing methods cannot handle left recursive grammars.
- Left recursion can be eliminated as follows:

Given Left Recursion: $\mathbf{A \rightarrow A\ \alpha \mid \beta}$

Then,

$\mathbf{A \rightarrow \beta\ A'}$

$\mathbf{A' \rightarrow \alpha A' \mid \varepsilon}$

# Examples on Removing Left Recursion

Given Grammar:

E→E+T|T

T→T*F|F

F→(E)|**id**

**Solution:**

First eliminate the left recursion for E as E→E+T|T

E→TE'

E'→+TE'|ε

Then eliminate for T as T→T*F|F

T→FT'

T'→*FT'|ε

# Examples on Removing Left Recursion

Given Grammar:

E→E+T|T

T→T*F|F

F→(E)|**id**

**Solution:**

E→TE'

E'→+TE'|ε

T→FT'

T'→*FT'|ε

F→(E)|**id**

# Examples on Removing Left Recursion

Example 2:

$A \rightarrow ABd \mid Aa \mid a$

$B \rightarrow Be \mid b$

Solution-

The grammar after eliminating left recursion is-

$A \rightarrow aA'$

$A' \rightarrow BdA' \mid aA' \mid \epsilon$

$B \rightarrow bB'$

$B' \rightarrow eB' \mid \epsilon$

# Examples on Removing Left Recursion

Example 3:

$S \rightarrow A$

$A \rightarrow Ad \mid Ae \mid aB \mid ac$

$B \rightarrow bBc \mid f$

Solution:

The grammar after eliminating left recursion is-

$S \rightarrow A$

$A \rightarrow aBA' \mid acA'$

$A' \rightarrow dA' \mid eA' \mid \epsilon$

$B \rightarrow bBc \mid f$

# Left Factoring

- It is a grammar transformation that is suitable for producing grammar for predictive parser
- Basic Idea: When it is not clear which of two alternative productions to use to expand non Terminal

**If $A \rightarrow \alpha \beta 1 | \alpha \beta 2$**

The left factor of above grammar is

**$A \rightarrow \alpha A'$**

**$A' \rightarrow \beta 1 \mid \beta 2$**

# Example on Left Factoring

S → a | ab | abc | abcd

Solution.-

Step-01:

S → aS’

S’ → b | bc | bcd | ∈

Again, this is a grammar with common prefixes

Step-02:

S → aS’

S’→ bA’ | ∈

A’ → c | cd | ∈

Again, this is a grammar with common

# Example on Left Factoring

$S \rightarrow a \mid ab \mid abc \mid abcd$

Solution.-

Step-03:

$S \rightarrow aS'$

$S' \rightarrow bA' \mid \epsilon$

$A' \rightarrow cB' \mid \epsilon$

$B' \rightarrow d \mid \epsilon$

# Example on Left Factoring

Example 2:

$S \rightarrow aAd \mid aB$

$A \rightarrow a \mid ab$

$B \rightarrow ccd \mid ddc$

Solution-

The left factored grammar is-

$S \rightarrow aS'$

$S' \rightarrow Ad \mid B$

$A \rightarrow aA'$

$A' \rightarrow b \mid \in$

$B \rightarrow ccd \mid ddc$

# Example on Left Factoring

Example 3:

S → aSSbS | aSaSb | abb | b

Solution:

Step-01:

S → aS’ | b

S’ → SSbS | SaSb | bb

Again, this is a grammar with common prefixes

Step-02:

S → aS’ | b

S’ → SA’ | bb A’ → SbS | aSb