Theory of Computing (CSE 221)

LECTURE 05 - Context Free Grammar

Book

Chapter 03 : Syntax Analysis

Lexical Analysis vs. Syntax Analysis

Lexical Analysis	Syntax Analysis
Splits the input into tokens	Recombine those tokens

To find the structure of the input text, the syntax analysis must also reject invalid texts by reporting syntax errors.

Here we use:

context-free grammars.

CFG: Context Free Grammar

Like regular expressions, context-free grammars describe sets of strings, i.e., languages.

Additionally, a context-free grammar also defines structure on the strings in the language it defines.

CFG: Context Free Grammar

- □ A context-free grammar has four components: G = (V, Σ, P, S)

1. A set of **nonterminals** (V). Non-terminals are syntactic variables that denote sets of strings. The non-terminals define sets of strings that help define the language generated by the grammar.

Ex. **A** → **aA, A** is the **Nonterminal.**

2. A set of tokens, known as **terminals** (Σ). Terminals are the basic symbols from which strings are formed. Such as **Alphabets**.

Ex. **A** → **aA, a** is the **Terminal.**

CFG: Context Free Grammar

- A context-free grammar has four components: G = (V, Σ, P, S)

3. A set of **productions** (P). The productions of a grammar specify the manner in which the terminals and nonterminals can be combined to form strings.

Ex. **N** → **X1 …Xn** is called the **production.**

4. One of the non-terminals is designated as the **start symbol** (S); from where the production begins.

Derivation

- A derivation is basically a sequence of production rules, in order to get the input string.
- During parsing, we take two decisions for some sentential form of input:
  - Deciding the non-terminal which is to be replaced.
  - Deciding the production rule, by which, the non-terminal will be replaced.
- To decide which non-terminal to be replaced with production rule, we can have two options.

Derivation

Ex. Consider the Grammar:
G1 = ({S,A}, {a,b},S,{S→aAb, aA→aaAb, A→Empty})
Generate language from this grammar.

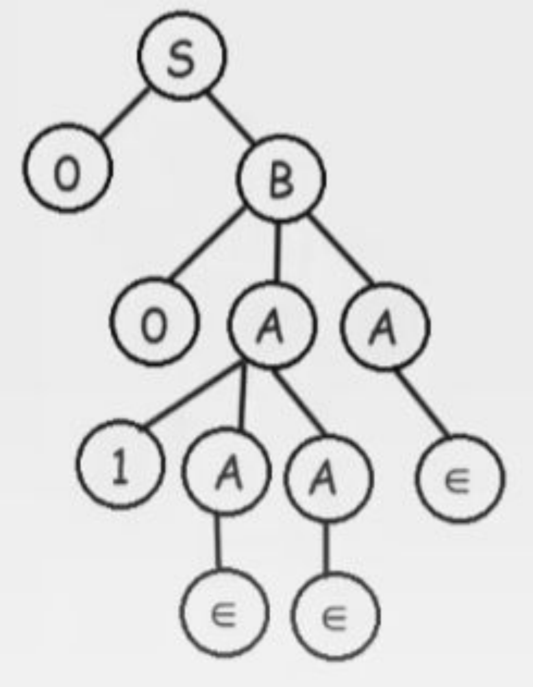
S→aAb	~by S→aAb
S→aaAbb	~by aA→aaAb
S→aaaAbbb	~by aA→aaAb
S→aaabbb	~ by A→Empty

Practice: G2 = ({S,A,B},{a,b},S,{S→AB, A→a, B→b})

Derivation Tree

A Derivation Tree represents the information of strings from a CFG.

Ex. Derivation Tree for G = {V,T,P,S} where S→OB, A→1AA|Empty, B→OAA is shown below.



Root Vertex:	**Start (S)**
Vertex:	**Nonterminal**
Leaves:	**Terminal or Empty**

Derivation Tree Types

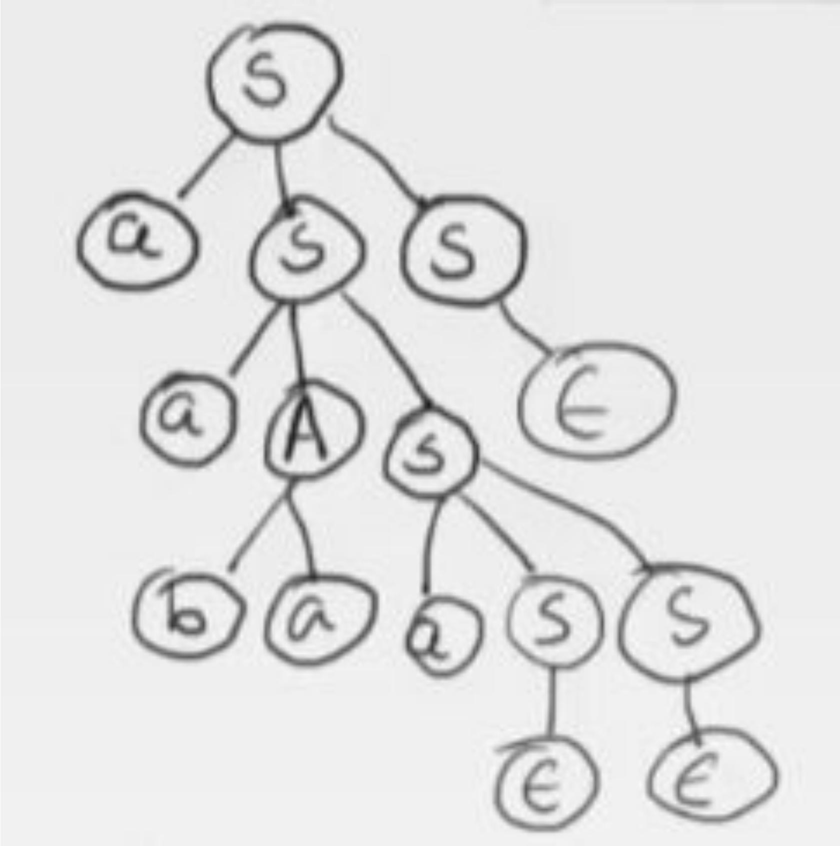
- **Left-most Derivation Tree**
- If the sentential form of an input is scanned and replaced from left to right, it is called left-most derivation. The sentential form derived by the left-most derivation is called the left-sentential form.
- **Right-most Derivation Tree**
- If we scan and replace the input with production rules, from right to left, it is known as right-most derivation. The sentential form derived from the right-most derivation is called the right-sentential form.

Left Derivation Tree

- ★ Choose the **Leftmost** node to expand.

Generate string **aabaa** from the given grammar.

Grammar S→aAS | aSS | Empty, A→SbA | ba

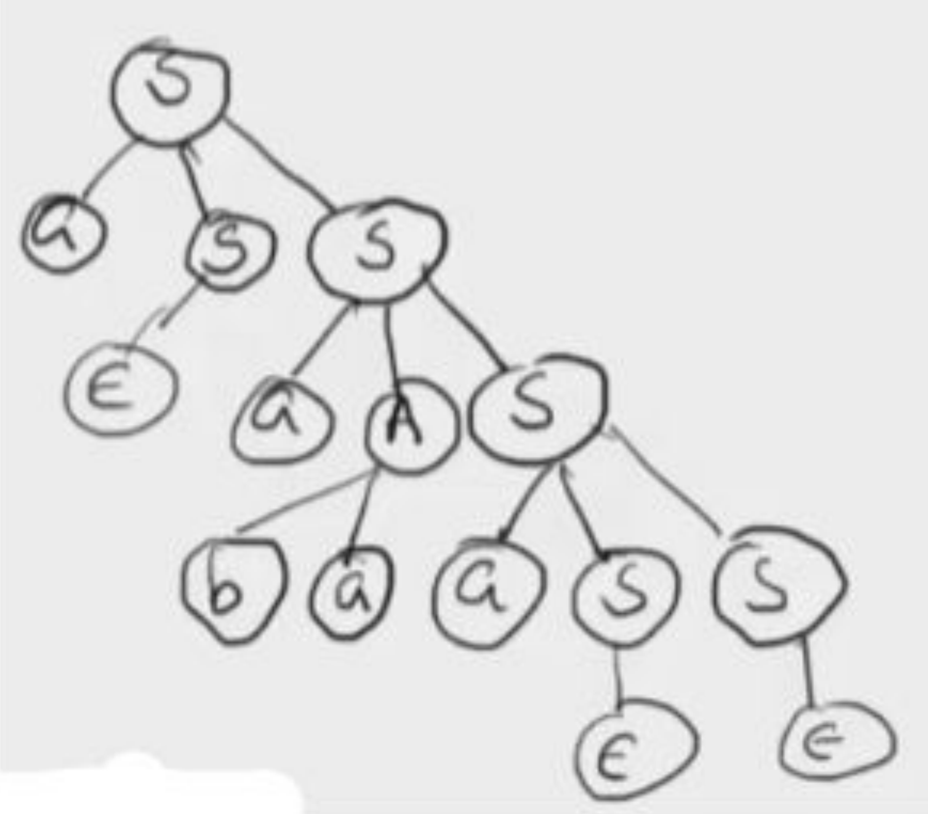


Right Derivation Tree

★ Choose the **Rightmost** node to expand.

Generate string **aabaa** from the given grammar.

Grammar S→aAS | aSS | Empty, A→SbA | ba



Derivation Tree Example 02

- Production rules: Input string: id+id*id

  $E \rightarrow E+E$

  $E \rightarrow E*E$

  $E \rightarrow id$

The left-most derivation is:	The right-most derivation is:
$E \rightarrow E * E$	$E \rightarrow E + E$
$E \rightarrow E + E * E$	$E \rightarrow E + E * E$
$E \rightarrow id + E * E$	$E \rightarrow E + E * id$
$E \rightarrow id + id * E$	$E \rightarrow E + id * id$
$E \rightarrow id + id * id$	$E \rightarrow id + id * id$

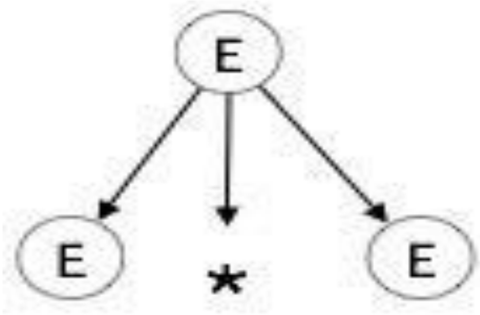
Parse Tree/Derivation Tree Generation

- A parse tree is a graphical depiction of a derivation.
- It is convenient to see how strings are derived from the start symbol.
- The start symbol of the derivation becomes the root of the parse tree.

Constructing the Derivation/Parse Tree

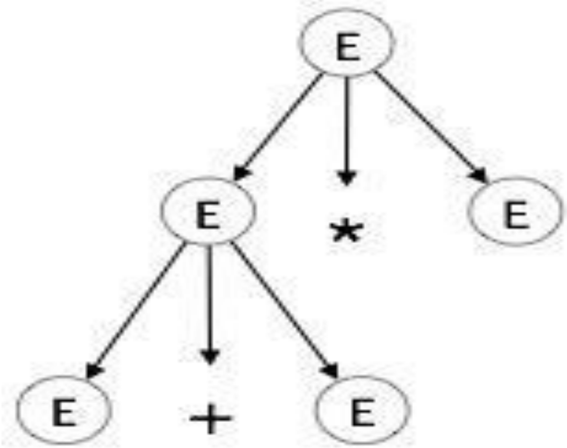
- Step 1:

E → E * E



- Step 2:

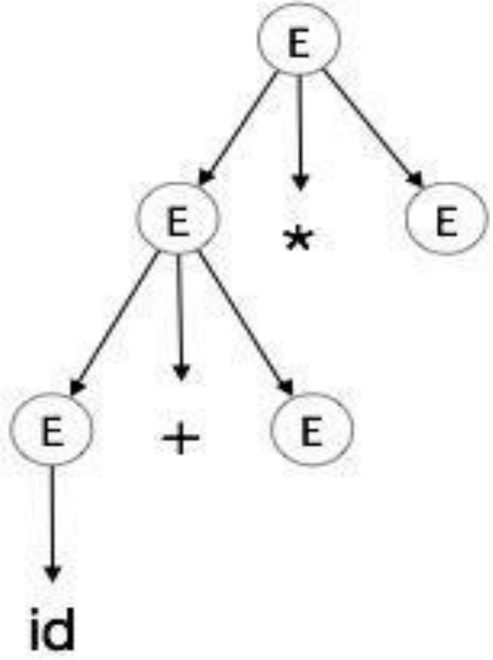
E → E + E * E



Constructing the Parse Tree(cntd.)

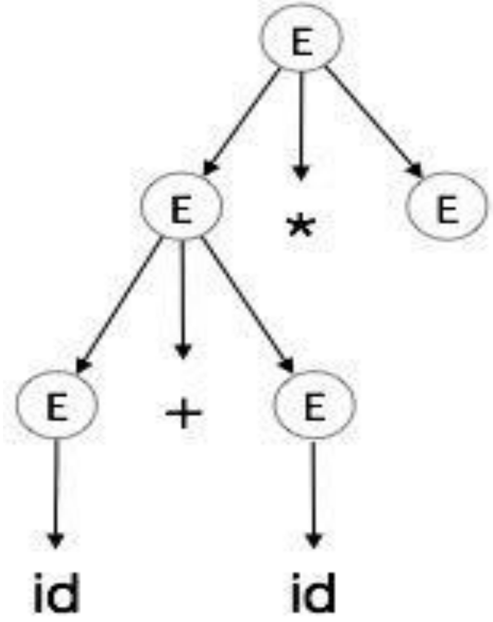
□ Step 3:

E ⟶ id + E * E



□ Step 4:

□ E ⟶ id + id * E



Constructing the Parse Tree(cntd.)

- Step 5:

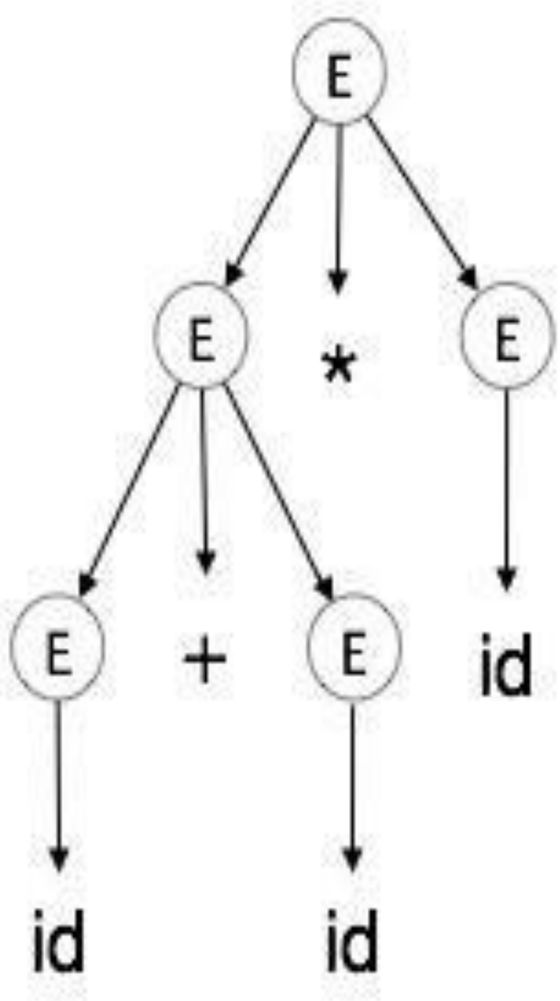
E → id + id * id

- In a parse tree:

-All leaf nodes are terminals.

-All interior nodes are non-terminals.

-In-order traversal gives original input string



Exercise

- Consider the following grammar

  S-> ABA|aaBA

  A->aBa|bA

  B->bBB|ba

Show how the string **abaabbabaabaa** can be generated by the grammar (assuming A->ABA is starting production rule) by left hand derivation and right hand derivation.

Exercise

- How can you derive 9-2+4 by using by using following production rules show the steps.

  list□ list + digit

  list□ list - digit

  list□ digit

  digit□ 1|2|3|4|5|6|7|8|9|0

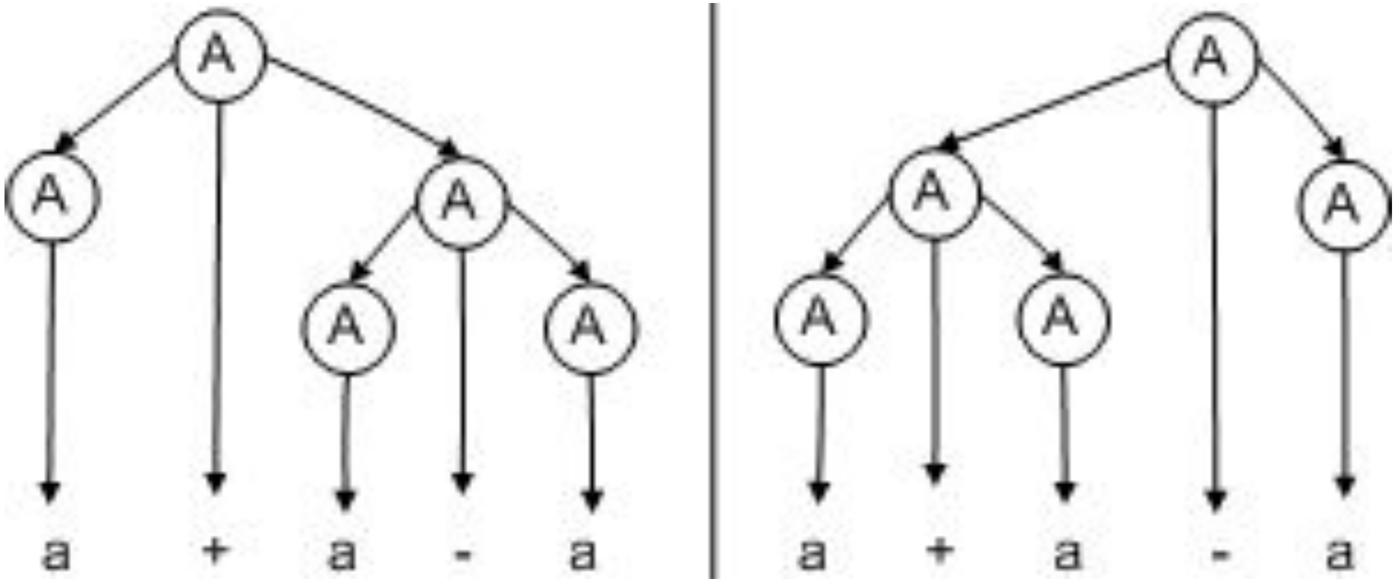
Ambiguity

- A grammar G is said to be ambiguous if it has more than one parse tree (two or more left/two or more right derivation) for at least one string.
- **Example**
- The context free grammar
- A → A+A | A−A | a
- is ambiguous since there are two leftmost derivations
- For the string a+a–a, the above grammar generates two parse trees:

Ambiguous Parse Trees

Grammar: A → A + A | A − A | id

Input String: a + a – a



Thank You