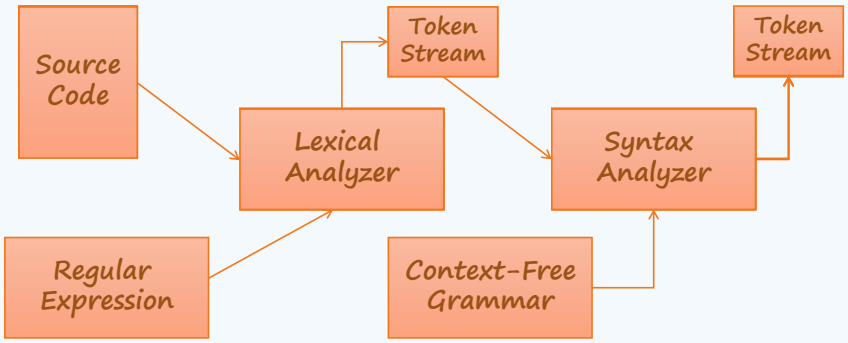
# Syntax Analysis

**Zakia Zinat Choudhury**
Lecturer
Department of Computer Science & Engineering
University of Rajshahi

## CONTENTS

# The Role of the Parser

A syntax analyzer or parser takes the input from a lexical analyzer in the form of token streams. The parser analyzes the source code (token stream) against the production rules to detect any errors in the code. The output of this phase is a parse tree.




## Limitations of Syntax Analyzers

Syntax analyzers receive their inputs, in the form of tokens, from lexical analyzers. Lexical analyzers are responsible for the validity of a token supplied by the syntax analyzer.

Syntax analyzers have the following drawbacks -

- × it cannot determine if a token is valid
- × it cannot determine if a token is declared before it is being used
- × it cannot determine if a token is initialized before it is being used
- × it cannot determine if an operation performed on a token type is valid or not.

# Syntax Error Handling

- **Lexical errors** include misspellings of identifiers, keywords, or operators.
- **Syntactic errors** include misplaced semicolons or extra or missing braces; that is, '((" or ")."
- **Semantic errors** include type mismatches between operators and operands.
- **Logical error** is a mistake in a program's source code that results in incorrect or unexpected behavior. It is a type of runtime error that may simply produce the wrong output or may cause a program to crash while running.



# Context-Free Grammars

**A Context-Free Grammar(CFG) is a set of rules or productions used to generate patterns of strings.**

CFG has four tuples (V/N,T/Σ,P/R,S)

V → Variable or Non-Terminals
T → Termminals or Σ
P → Productions or Rules
S → Star Symbol

# Description of Four Tuples of CFG

- **Terminals(Σ):** A set of terminal symbols, sometimes referred to as "tokens." Terminals are the basic symbols from which strings are formed.
- **Non-Terminals(V):** A set of nonterminals, sometimes called "syntactic variables." Each nonterminal represents a set of strings of terminals. Non terminal symbol appears on the left side of the production.
- **A set of productions(P):** The productions of a grammar specify the manner in which the terminals and non-terminals can be combined to form strings. Each production consists of a nonterminal, called the head or left side of the production, an arrow, and a sequence of terminals and/or nonterminals, called the body or right side of the production.
- **Start Symbol(S):** One of the non-terminals is designated as the start symbol (S); from where the production begins.



# Derivations

A derivation is basically a sequence of production rules, in order to get the input string. During parsing, two decisions are taken 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.

**Left-most Derivation**

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.

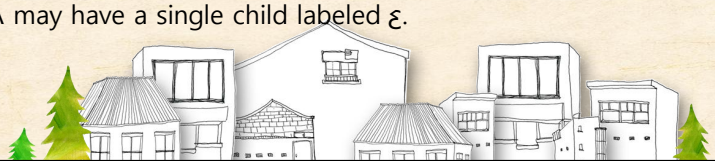
**Right-most Derivation**

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.

# Parse Tree

A parse tree is a graphical representation of a derivation. It is convenient to see how strings a re derived from the start symbol. The start symbol of the derivation becomes the root of the parse tree. When a CFG is given, a parse tree can be found accrding to the grammar with following properties:

1. The root is labeled by the start symbol.
2. Each leaf is labeled by a terminal or by ε.
3. Each interior node is labeled by a nonterminal.
4. If A is the nonterminal labeling some interior node and X1, X2, . . . , Xn are the labels of the children of that node from left to right, then there must be a production **A → X1 X2 ..... Xn**. Here, X1, X2, . . . , Xn, each stand for a symbol that is either a terminal or a nonterminal. As a special case, if **A → ε** is a production, then a node labeled A may have a single child labeled ε.



# Ambiguity

A grammar that produces more than one parse tree for some sentence is said to be ambiguous.

Put another way, an ambiguous grammar is one that produces more than one leftmost derivation or more than one rightmost derivation for the same sentence.

For Example:

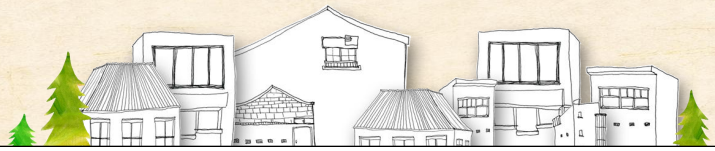
The Grammar is:

E→E+E

E→E-E

E→id

For the string id + id – id, how many parse tree will be geanerated?



# Recursion

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.

Recursion two types:

Left and right recursion

# Elimination of Left Recursion

A grammar is left recursive if it has a nonterminal A such that there is a derivation $A \rightarrow A\alpha$ for some string $\alpha$. Top-down parsing methods cannot handle left-recursive grammars, so a transformation is needed to eliminate left recursion.

# Elimination of Left Recursion

Left Recursion

The left-recursive pair of productions

$A \rightarrow A\alpha / \beta$

could be replaced by the non-left-recursive productions

$A \rightarrow \beta A'$

$A' \rightarrow \varepsilon | \alpha A'$

Example:

$E \rightarrow E+T / T$ [left-recursive production]

($A$: $E$, $\alpha$: $+T$, $\beta$: $T$)

non-left-recursive productions are

$E \rightarrow TE'$

$E' \rightarrow \varepsilon | +TE'$

# Left Factoring

**Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive, or top-down, parsing. When the choice between two alternative A-productions is not clear, we may be able to rewrite the productions to defer the decision until enough of the input has been seen that we can make the right choice.**

# Left Factoring

Left - Factoring

The productions are

$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \cdots\cdots \mid \alpha\beta_n \mid \gamma$

The left - factored grammars / productions are

$A \to \alpha A' \mid \gamma$

$A' \to \beta_1 \mid \beta_2 \mid \beta_3 \mid \cdots\cdots \mid \beta_n$

Example :

$S \to iEtS \mid iEtS\,eS \mid a$

($\alpha$ = $iEtS$, $\beta_1$ = $\varepsilon$, $\beta_2$ = $eS$, $\gamma$ = $a$)

The left - factored grammars are

$S \to iEtSS' \mid a$

$S' \to \varepsilon \mid eS$



Assignment

Consider the context - free grammar

a) $S \to SS+ \mid SS* \mid a$ and input string aa+a*

b) $S \to 0S1 \mid 01$ and input string 000111

c) $S \to +SS \mid *SS \mid a$ and input string +*aaa

1. Give a left - most derivation
2. Give a right - most derivation
3. Give a parse tree
4. Is the grammar ambiguous or unambiguous? Justify your answer.

a) Sajeeb Chakraborty, Rumi Umme, Md. Meem Mursalin Chowdhury, Humayun Ahmad Rajib,Rico, Fahim Nirob

b) Abdur Rahim Sheikh, itsRakibul, Bobi, Mehedi Hasan,Prithu Rani Roy

c) Jahid hasan, Rafi, Rifat, Asif Himu, Riya, zahir

# THANK YOU