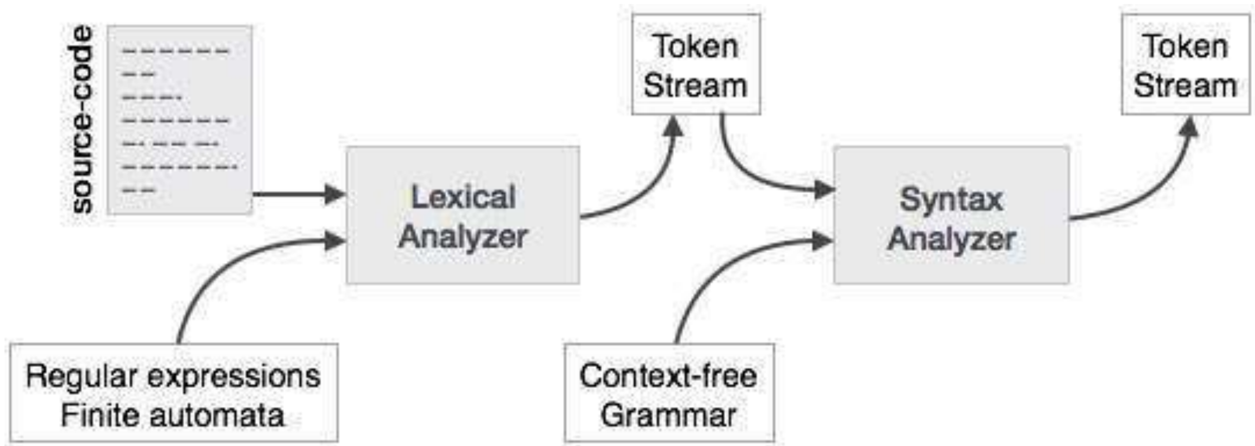
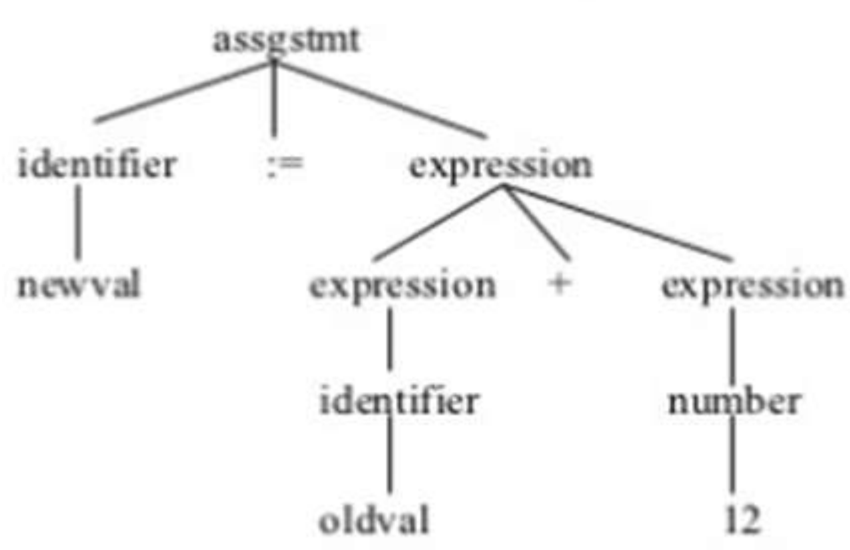
# Unit III – Syntax & Semantic Analysis

- 2nd phase of compilation process
- Syntax
- DS – Syntax tree /parse tree



# Syntax Analysis/parser

- Parse tree
  - Leaf node - terminals
  - Inner nodes – Non-Terminal



# Role of Syntax Analyzer

- Token streams → production rules → parse tree → error → error recovering strategies'
- Opening and closing brace
- *Errors:*
  - Arithmetic expression with unbalanced parenthesis
  - Error handler – clear and accurate, slow down of current program processing should be avoided
  - Error Recovery stratregies
    - Panic mode
    - Phrase level
    - Error production
    - Global Correction
- *Issues:*
  - Data type mismatch for an operation → Semantic Analysis

# Context Free Grammar (CFG)

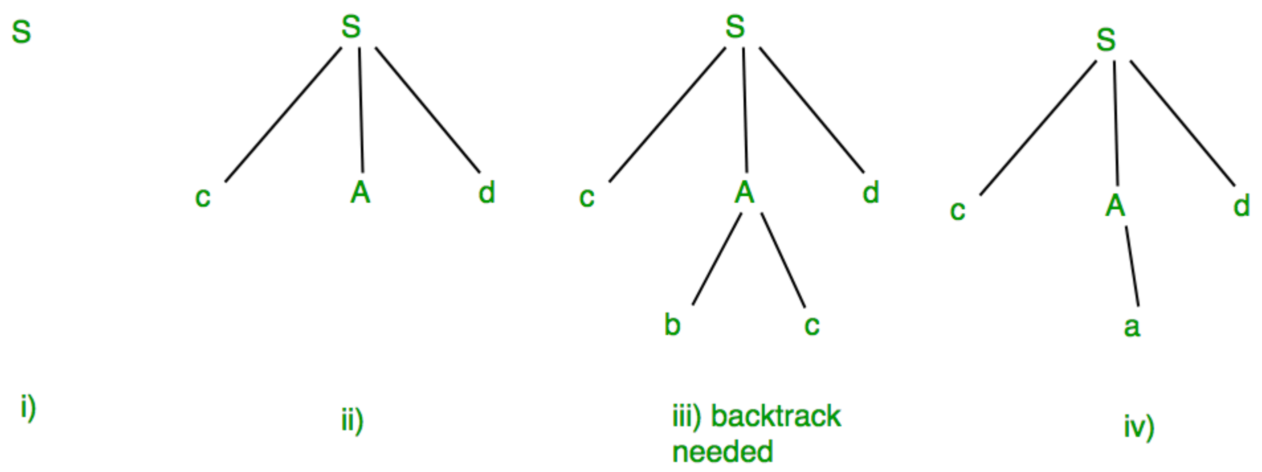
- Syntax analyzer
  - program - rules implied by CFG
  - Parse tree

- CFG
  - G =(V,T,P,S)
  - Type 2 (Context Free Grammar)
    - $\alpha \rightarrow \beta$
    - $\alpha \in V$
    - $\beta \in (V+T)^*$

| Grammar Accepted | Automaton |
| --- | --- |
| Unrestricted grammar | Turing Machine |
| Context-sensitive grammar | Linear-bounded automaton |
| Context-free grammar | Pushdown automaton |
| Regular grammar | Finite state automaton |

# Example for Syntax Analyzer

- Derivation →sequence of production rules (INPUT STRING)
- S -> cAd
- A -> bc|a
- Input string is “cad”



# Syntax Analyzer - Derivation

- Sentential (String derivable from start symbol)
- Which Production rules is to be used for Non-Terminal
  - Left-most Derivation (left to right Non-terminal will be replaced)
  - Right-most Derivation (Right to Left)
- *Example:*
- S→(L)|a
- L→L,S|S
- Construct the string (a, a)
- *Left Most Derivation*
- S→(L)
- S→(L,S)
- S→(S,S)
- S→(a,S)
- S→(a,a)

- *Right Most Derivation*
- S →(L)
- S → (L,**S**)
- S → (**L**,a)
- S → (**S**,a)
- S → (a,a)

# Derivation – Example

- *Example 2:*
- E→E+E|E*E
- E→id
- String→**id+id*id**
- *Left most derivation*
- E →**E**+E
- E→id+**E**
- E→id+**E***E
- E→id+id***E**
- E→id+id*id

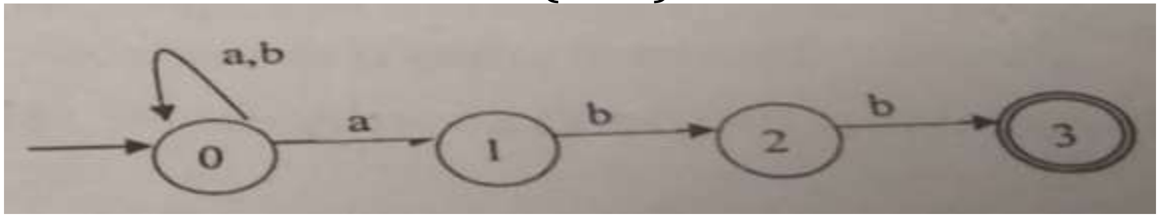
- *Right most derivation*
- E→E+**E**
- E→E+E***E**
- E→E+**E***id
- E→**E**+id*id
- E→id+id*id

# Writing a Grammar

| REGULAR EXPRESSION | CONTEXT-FREE GRAMMAR |
|---|---|
| It is used to describe the tokens of programming languages. | It consists of a quadruple where S → start symbol, P → production, T → terminal, V → variable or non- terminal. |
| It is used to check whether the given input is valid or not using **transition diagram.** | It is used to check whether the given input is valid or not using **derivation**. |
| The transition diagram has set of states and edges. | The context-free grammar has set of productions. |
| It has no start symbol. | It has start symbol. |
| It is useful for describing the structure of lexical constructs such as identifiers, constants, keywords, and so forth. | It is useful in describing nested structures such as balanced parentheses, matching begin-end's and so on. |

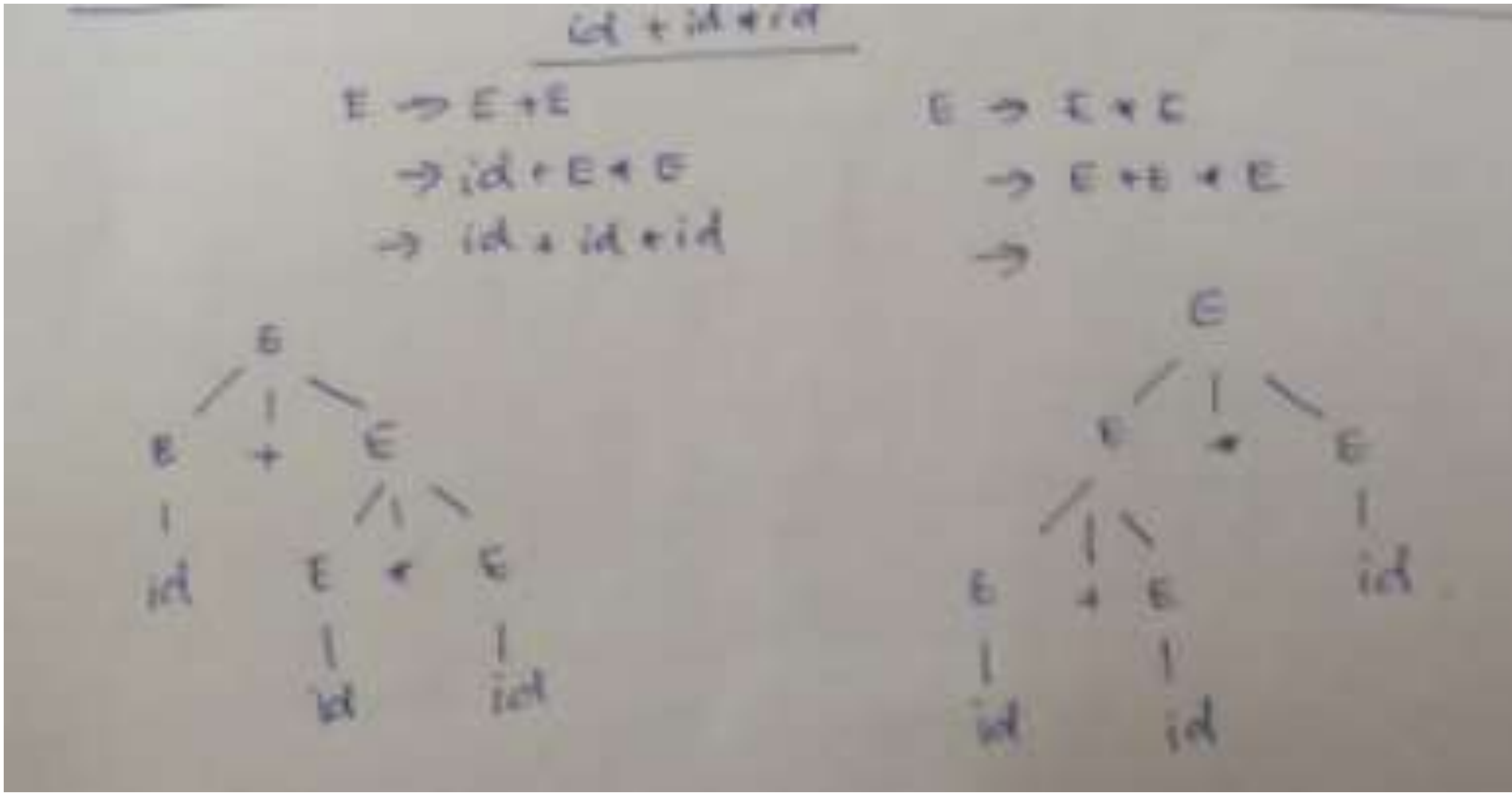
# Writing a Grammar

- Write the Grammar for RE = (a+b)*abb




- A0→aA0|bA0|aA1
- A1→bA2
- A2→bA3
- A3→Epsilon
- To make it parsable – rewrite the Grammar
  - *Eliminating the Ambiguous Grammar*
  - *Eliminating Left-Recursion*
  - *Eliminating Left-Factoring*

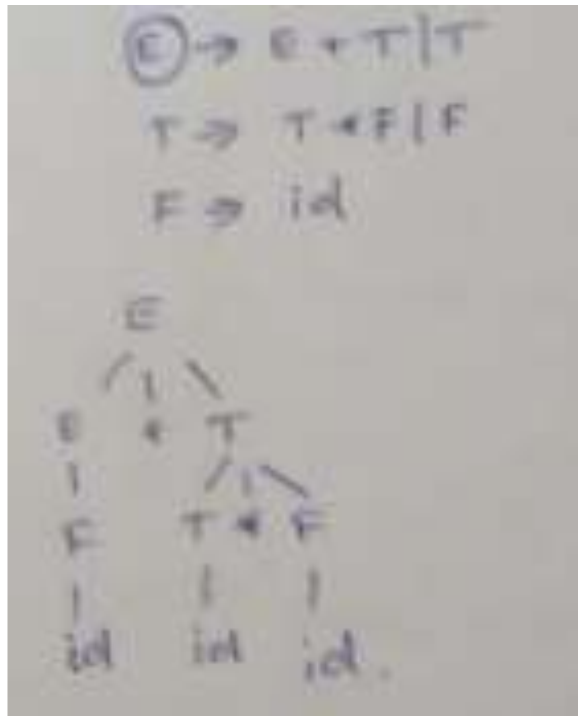
# Parse Tree



# Eliminating the Ambiguous Grammar

- ***Ambiguous grammar*** – More than one parse tree (left/right derivation)
- Names of a person (Unique)
- *Example :*
- **E→E+E|E*E**
- **E→id**
- String→**id+id*id**
- *Left most derivation(1)*
- E →E+**E**
- E→E+E*E
- E→id+id*id
- *Left most derivation(2)*
- E→E*E
- E→E+E*E
- E→id+id*id
- **Eliminating Ambiguity**
- **E→E+T|T**
- **T→T*F|F**
- **F→id**
- *Left most derivation*
- E→E+T
- E→T+T*F
- E→F+F*F →id+id*id

# Parse Tree after Eliminating Ambiguity



# Eliminating Left Factoring

- Left Factoring – removing the common left factor that appears in two productions of same non-terminal
- Eliminating left factoring helps in ***avoiding the backtracking***
- <u>Example:</u>
- A→qB|qC
  - A→qD
  - D→B|C

# Eliminating left Recursion

- Left Recursion
  - Left most non-terminal in a production of non-terminal is the non-terminal itself
- A→Aα|β
- Where α , β are sequence of terminals/non-terminals
- **General Form:**
- **A→Aα|β A→ βA'**
  - **A'→ αA'| ε**
- Example1:
- **E→E+T|T , α →+T, β →T**
  - E→TE'
  - E'→+TE'| ε

**A→A α| β**

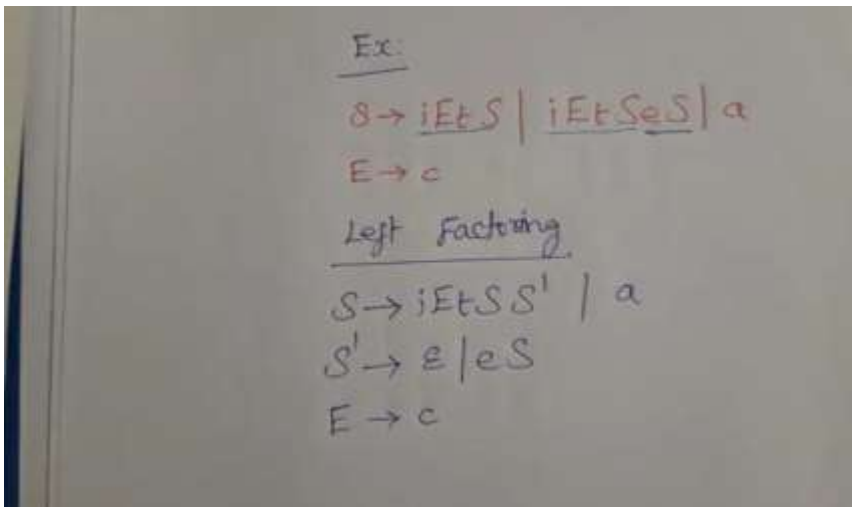
**→ β →A α→A α α→A α α α→ β α α α→ β α***

- **α* ={empty, α, α α, α α α, α α α α,…….}**
- **A→ β**
- **A→A α → β α**
- **A→A α→A α α→ β α α**
- **A→A α→A α α→A α α α→ β α α α**

# Eliminating left Recursion

- **General Form:**
- **A→Aα|β**
  - A→ βA’
  - A’→ αA’| ε
- Example 2:
- A→AB α|Aa|a
  - α1 →
  - α2 →
  - β →
- Example3:
- A→AC|Aad|bd|c

# Examples



Ex:

$S \rightarrow iEtS \mid iEtSeS \mid a$

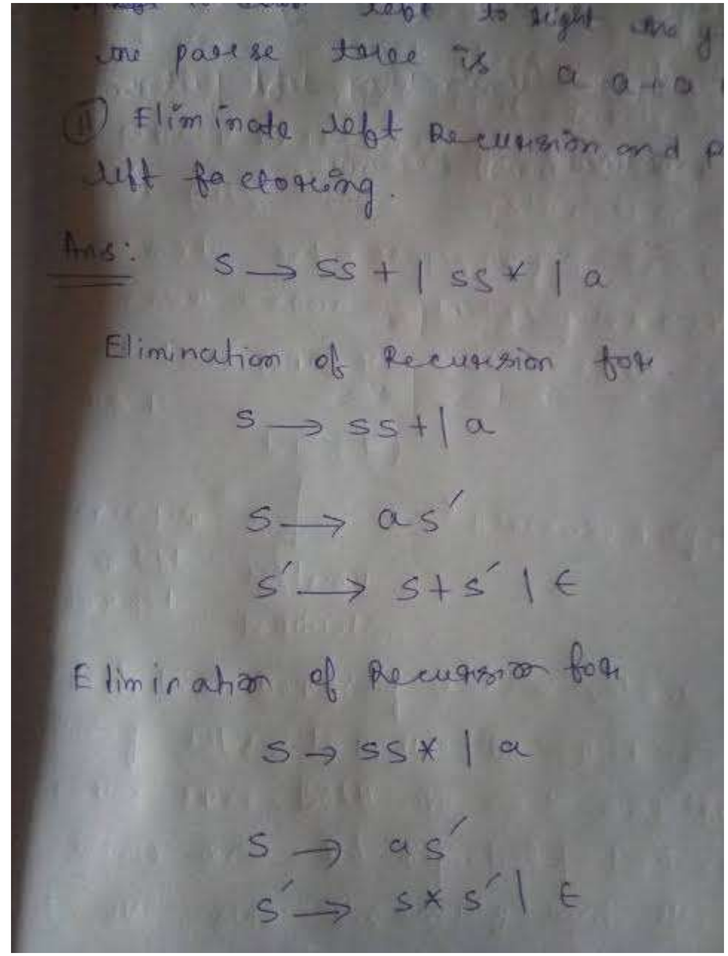
$E \rightarrow c$

Left Factoring

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow \varepsilon \mid eS$

$E \rightarrow c$



the parse tree is a a+a

(ii) Eliminate left Recursion and p

left factoring.

Ans: $S \rightarrow SS+ \mid SS* \mid a$

Elimination of Recursion for

$S \rightarrow SS+ \mid a$

$S \rightarrow aS'$

$S' \rightarrow S+S' \mid \epsilon$

Elimination of Recursion for

$S \rightarrow SS* \mid a$

$S \rightarrow aS'$

$S' \rightarrow S*S' \mid \epsilon$

# Quiz

1. A given grammar is called ambiguous if
   - 1.Two or more productions have the same non-terminal on the left -hand side.
   - 2.A derivation tree has more than one associated sentence.
   - 3.There is a sentence with more than one derivation tree corresponding to it.
   - 4.Brackets are not present in the grammar
2. Given the grammar
   - S -> T * S| T
   - T ->U + T | U
   - U -> a | b
     - Ambiguous
     - Unambiguous