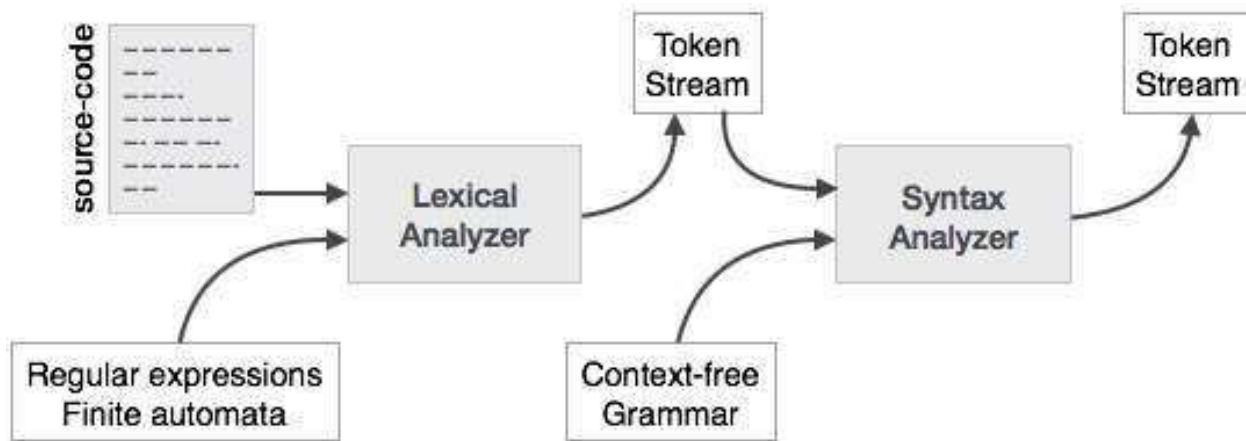




Unit III – Syntax & Semantic Analysis



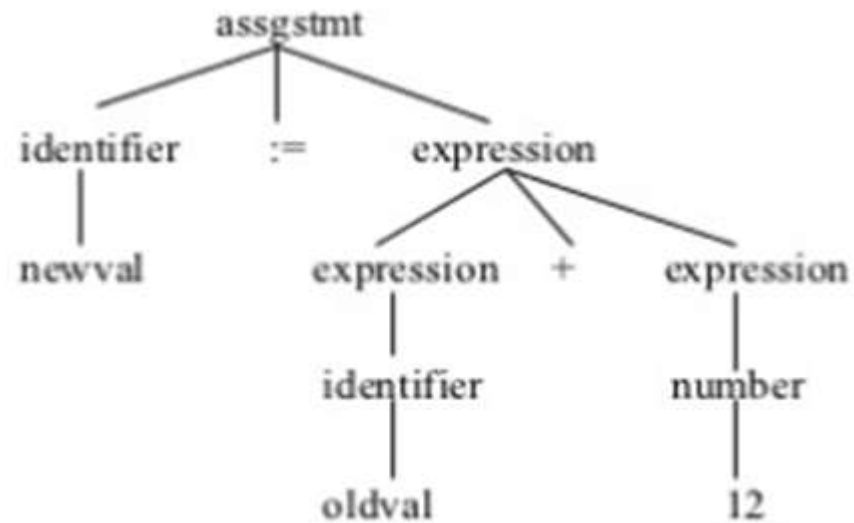
- 2nd phase of compilation process
- Syntax
- DS – Syntax tree /parse tree





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 strategies
 - Panic mode
 - Phrase level
 - Error production
 - Global Correction
- Issues:
 - Data type mismatch for an operation → Semantic Analysis



Context Free Grammar (CFG)

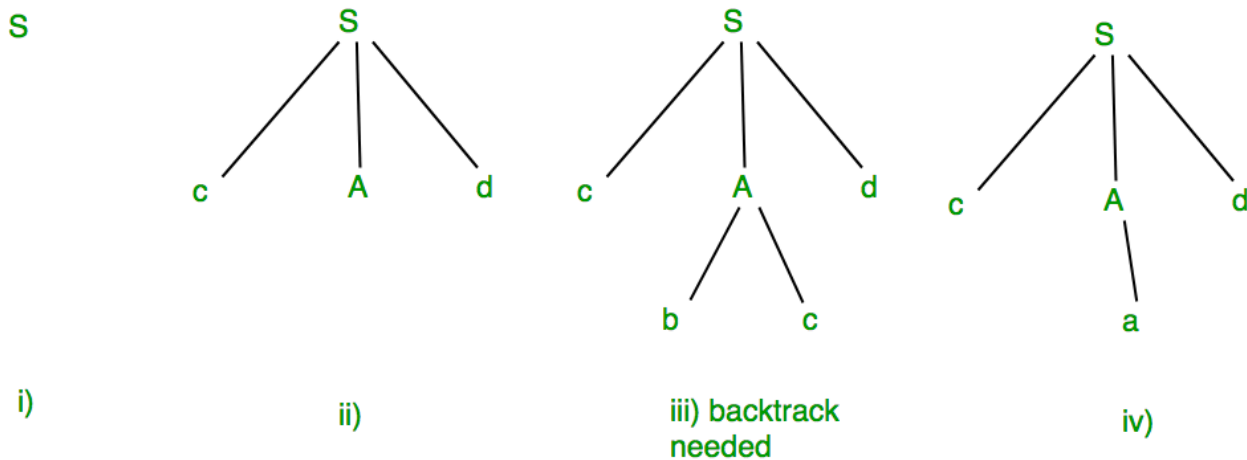
- Syntax analyzer
 - program - rules implied by CFG
 - Parse tree
- CFG
 - $G = (V, T, P, S)$
 - Type 2 (Context Free Grammar)
 - $\alpha \rightarrow \beta$
 - $\alpha \in V$
 - $\beta \in (V+T)^*$

| 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 \rightarrow sequence of production rules (INPUT STRING)
- $S \rightarrow cAd$
- $A \rightarrow bc|a$
- 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:

- $S \rightarrow (L) | a$
- $L \rightarrow L, S | S$
- Construct the string (a, a)

- Left Most Derivation

- $S \rightarrow (L)$
- $S \rightarrow (L, S)$
- $S \rightarrow (S, S)$
- $S \rightarrow (a, S)$
- $S \rightarrow (a, a)$

- Right Most Derivation

- $S \rightarrow (L)$
- $S \rightarrow (L, S)$
- $S \rightarrow (L, a)$
- $S \rightarrow (S, a)$
- $S \rightarrow (a, a)$



Derivation – Example



- Example 2:

- $E \rightarrow E + E \mid E * E$

- $E \rightarrow \text{id}$

- $\text{String} \rightarrow \mathbf{id + id * id}$

- Left most derivation

- $E \rightarrow E + E$

- $E \rightarrow \text{id} + E$

- $E \rightarrow \text{id} + E * E$

- $E \rightarrow \text{id} + \text{id} * E$

- $E \rightarrow \text{id} + \text{id} * \text{id}$

- Right most derivation

- $E \rightarrow E + E$

- $E \rightarrow E + E * E$

- $E \rightarrow E + E * \text{id}$

- $E \rightarrow E + \text{id} * \text{id}$

- $E \rightarrow \text{id} + \text{id} * \text{id}$



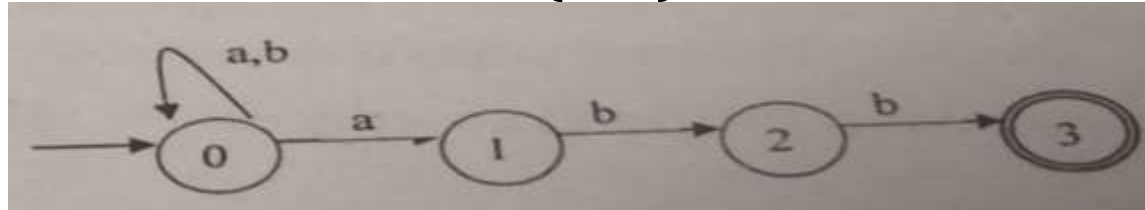
Writing a Grammar

| REGULAR EXPRESSION | CONTEXT-FREE GRAMMAR |
|---|--|
| It is used to describe the tokens of programming languages. | It consists of a quadruple where $S \rightarrow$ start symbol, $P \rightarrow$ production, $T \rightarrow$ terminal, $V \rightarrow$ 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

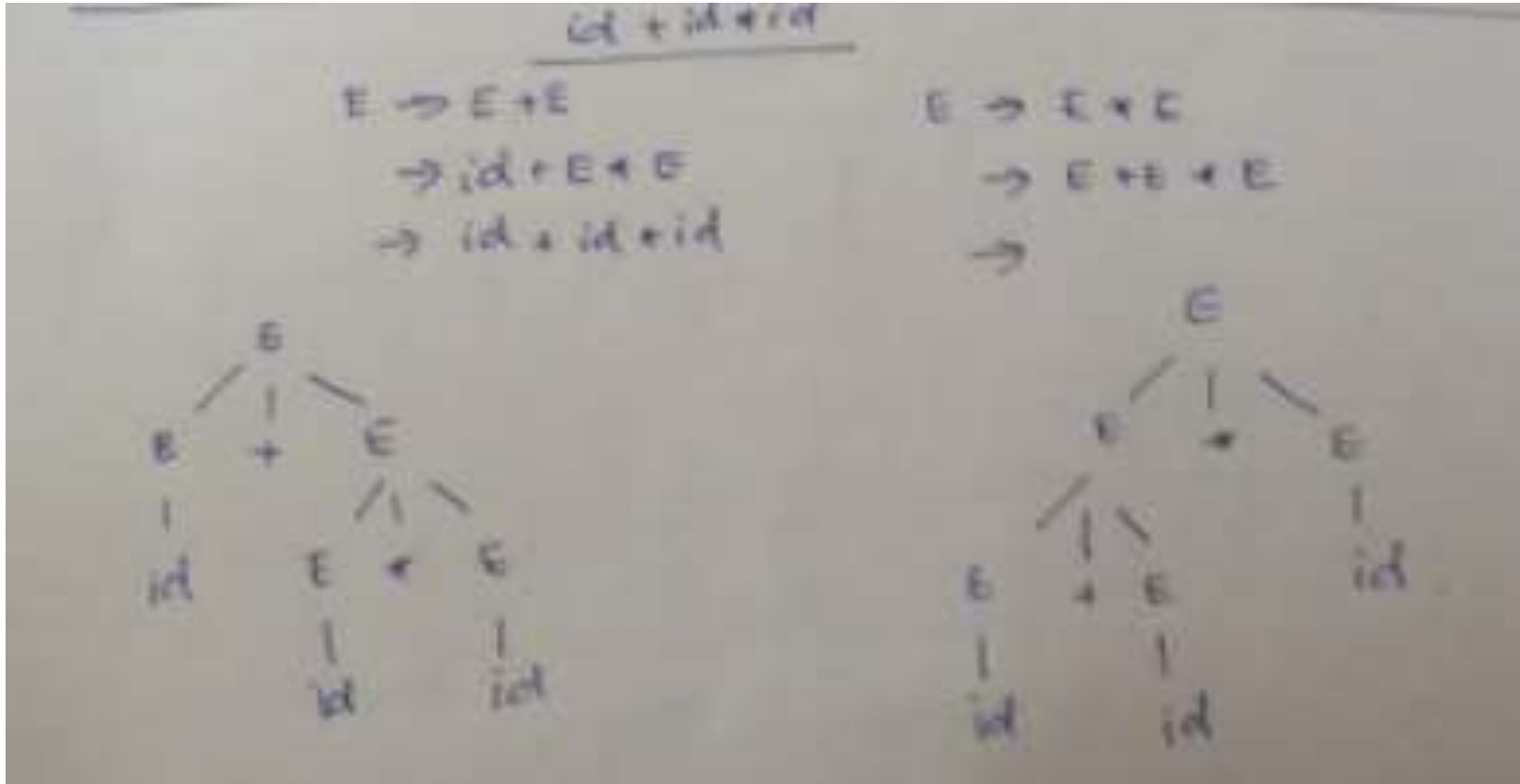
- Write the Grammar for RE = $(a+b)^*abb$



- $A_0 \rightarrow aA_0 | bA_0 | aA_1$
- $A_1 \rightarrow bA_2$
- $A_2 \rightarrow bA_3$
- $A_3 \rightarrow \text{Epsilon}$
- 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 :

- $E \rightarrow E + E \mid E * E$

- $E \rightarrow id$

- String $\rightarrow id + id * id$

- Left most derivation(1)

- $E \rightarrow E + E$

- $E \rightarrow E + E * E$

- $E \rightarrow id + id * id$

- Left most derivation(2)

- $E \rightarrow E * E$

- $E \rightarrow E + E * E$

- $E \rightarrow id + id * id$

- Eliminating Ambiguity

- $E \rightarrow E + T \mid T$

- $T \rightarrow T * F \mid F$

- $F \rightarrow id$

- Left most derivation

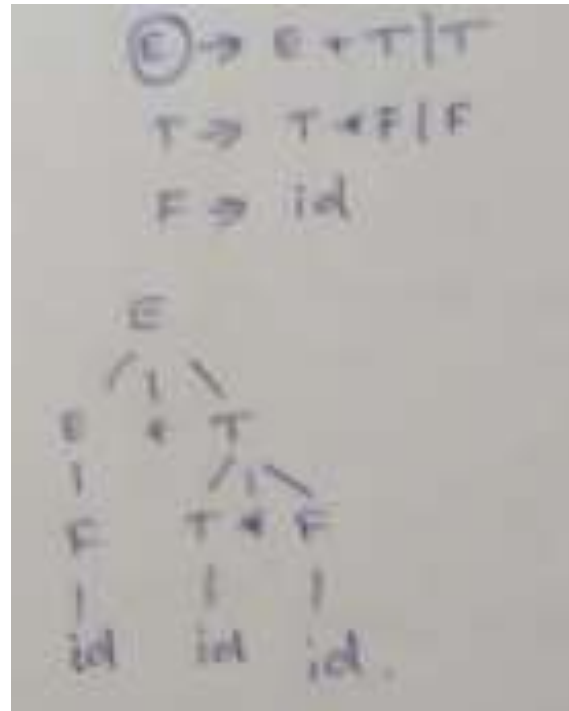
- $E \rightarrow E + T$

- $E \rightarrow T + T * F$

- $E \rightarrow F + F * F \rightarrow id + id * id$



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*
- Example:
- $A \rightarrow qB|qC$
 - $A \rightarrow qD$
 - $D \rightarrow B|C$



Eliminating left Recursion

- Left Recursion
 - Left most non-terminal in a production of non-terminal is the non-terminal itself
- $A \rightarrow A\alpha | \beta$
- Where α, β are sequence of terminals/non-terminals
- **General Form:**
- $A \rightarrow A\alpha | \beta \quad A \rightarrow \beta A'$
 - $A' \rightarrow \alpha A' | \epsilon$
- **Example1:**
- $E \rightarrow E+T | T, \alpha \rightarrow +T, \beta \rightarrow T$
 - $E \rightarrow TE'$
 - $E' \rightarrow +TE' | \epsilon$

$$A \rightarrow A \alpha | \beta$$

$$\rightarrow \beta \rightarrow A \alpha \rightarrow A \alpha \alpha \rightarrow A \alpha \alpha \alpha \rightarrow \beta \alpha \alpha \alpha \rightarrow \beta \alpha^*$$

- $\alpha^* = \{\text{empty}, \alpha, \alpha \alpha, \alpha \alpha \alpha, \alpha \alpha \alpha \alpha, \dots\}$
- $A \rightarrow \beta$
- $A \rightarrow A \alpha \rightarrow \beta \alpha$
- $A \rightarrow A \alpha \rightarrow A \alpha \alpha \rightarrow \beta \alpha \alpha$
- $A \rightarrow A \alpha \rightarrow A \alpha \alpha \rightarrow A \alpha \alpha \alpha \rightarrow \beta \alpha \alpha \alpha$



Eliminating left Recursion

- **General Form:**
- $A \rightarrow A\alpha | \beta$
 - $A \rightarrow \beta A'$
 - $A' \rightarrow \alpha A' | \epsilon$
- Example 2:
- $A \rightarrow AB \alpha | Aa | a$
 - $\alpha 1 \rightarrow$
 - $\alpha 2 \rightarrow$
 - $\beta \rightarrow$
- Example 3:
- $A \rightarrow AC | Aad | bd | c$



Examples

Ex:
 $S \rightarrow iEtS \mid iEtSeS \mid a$
 $E \rightarrow c$
Left Factoring
 $S \rightarrow iEtSS' \mid a$
 $S' \rightarrow \epsilon \mid eS$
 $E \rightarrow c$

one parse tree is $a \rightarrow a$
 ① Eliminate left recursion and left factoring.
Ans: $S \rightarrow SS^+ \mid a$
 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
 - $S \rightarrow T * S \mid T$
 - $T \rightarrow U + T \mid U$
 - $U \rightarrow a \mid b$
 - Ambiguous
 - Unambiguous