UPES

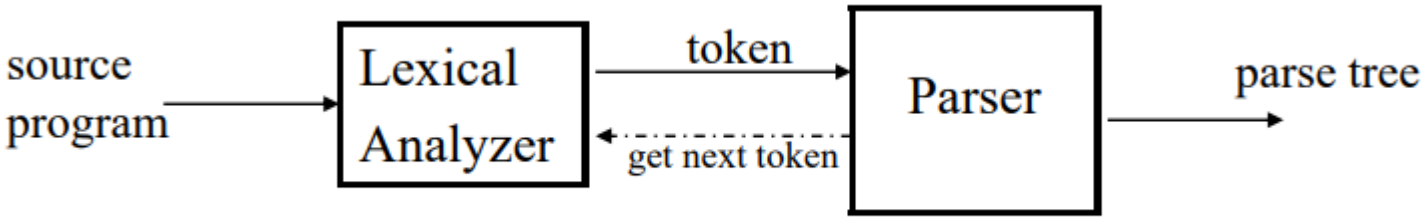
UNIVERSITY WITH A PURPOSE

Content of this lecture

- Syntax Analyzer

Role of Syntax Analyzer

- Syntax Analyzer (Parser) creates the syntactic structure of the given source program. This syntactic structure is called parse tree.
- Context Free Grammar (CGF) is used to specify the syntax of programming language.



UPES
UNIVERSITY WITH A PURPOSE

Steps of Syntax Analyzer

- Parser checks whether a given source program satisfies the rules implied by a CFG or not
- If it satisfies, the parser creates the parse tree of that program
- Otherwise the parser gives the error messages

What is CFG

- CFG gives a precise syntactic specification of a programming language which can be defined in the recursive manner.
- In a CFG, we have:
  - Finite set of terminals. These are basic symbols of which strings in the language are composed. We use small alphabets to denote terminals and these are non replaceable. E.g. begin, if, else, +
  - Finite set of non-terminals/syntactic variable/syntactic category: These are special symbols that denotes the set of strings. We use capital alphabets to denote non terminals and these are replaceable. E.g. expression, statement etc.
  - Finite set of productions rules which defines the way in which the non terminal may be built up from another non terminal and terminals. E.g.
    - A → α where A is a non-terminal and α is a string of terminals and non-terminals (including the empty string) and
    - |A| = 1
  - A start symbol which is one of the non-terminal symbols. E.g.
    - E → E + E | E – E | E * E | E / E | - E | (E) | id

UPES
UNIVERSITY WITH A PURPOSE

CFG Terminology

- L(G) is the language generated by grammar G, which is a set of sentence.
- Every sentence generated by grammar G is in L
- Every string in L can be generated by grammar G.
- S → α if α contains non terminal and terminals then it is called as a sentential form of G and if α does not contains non terminal, it contains terminal only then it is called sentence of G.

Derivation of String

- Derivation of string denotes the sequence of application of production rules to produce a group of terminals only while starting from the start symbol.
- E.g. if E → E + E | E – E | E * E | E / E | - E | (E) | id is the given set of production rules and we want to generate id+id*id then it can be derived as follows:

- E → E * E
- E → E + E*E
- E → id + E*E
- E → id + id*E
- E → id + id*id

Derivation of String

- Derivation of string can be achieved in two ways:
  - Left most derivation: Non terminal occurring first from left hand side will always be used for substitution.
  - Right most derivation: Non terminal occurring first from right hand side will always be used for substitution.
  - E.g. if E → E + E | E – E | E * E | E / E | - E | (E) | id is the given set of production rules and we want to generate id+id*id then it can be derived in left most and right most derivation as follows:

Right most derivation

E → E + E
E → E + E*E
E → E + E*id
E → E + id*id
E → id + id*id

Left most derivation

E → E * E
E → E + E*E
E → id + E*E
E → id + id*E
E → id + id*id

UPES
UNIVERSITY WITH A PURPOSE

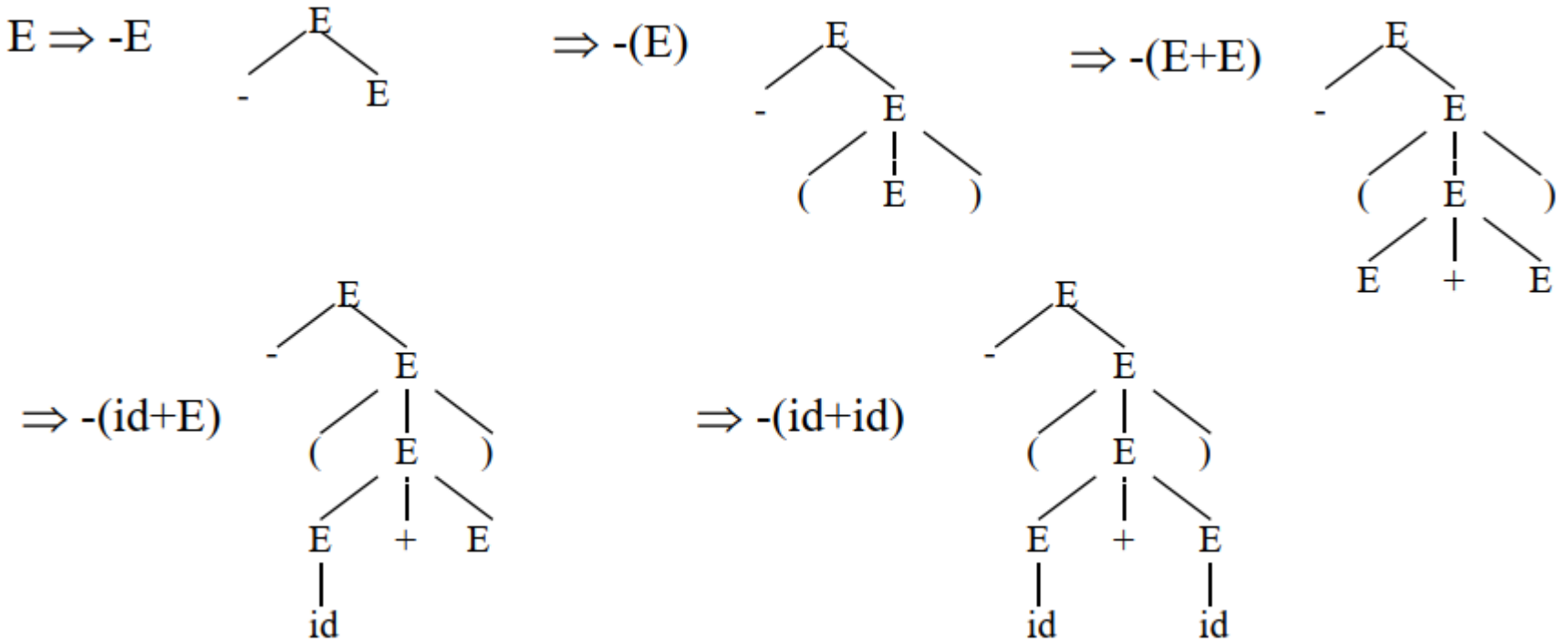
Parse Tree/Syntax Tree/ Derivation Tree/Generation Tree

- It is a diagramatical method to show the derivation of the string of terminals only while starting from the start symbol.
- Start symbol will always be root node of the tree.
- Each interior node of the tree is labelled by some of the non terminal.
- Leaf nodes of the parse tree are labelled by terminal symbols.

UPES
UNIVERSITY WITH A PURPOSE

Parse Tree/Syntax Tree/ Derivation Tree/Generation Tree

- If E → E + E | E – E | E * E | E / E | - E | (E) | id is the given set of production rules and we want to draw the parse tree for –(id+id) then its generation process is shown below:

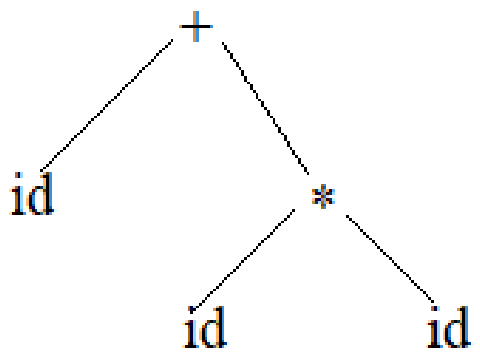


Expression Tree

- A variant of parse tree is called Expression tree which is used to show algebraic expressions only.
- It is always a binary tree.
- Non terminal symbols are not shown in this tree.
- Operators become the internal nodes
- Other non operator terminal symbols becomes the external nodes.

Expression Tree

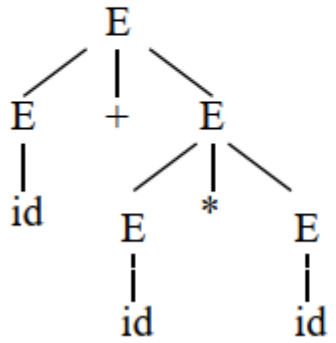
- If E → E + E | E – E | E * E | E / E | - E | (E) | id is the given set of production rules and we want to draw the expression tree for id+id*id



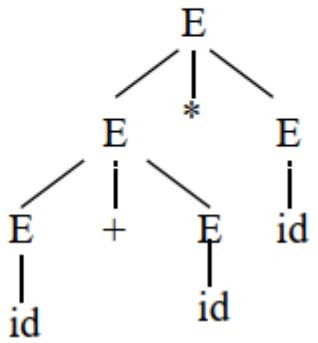
Ambiguity

- A grammar that produces more than one parse tree (either both from left most derivation or both from right most derivation) for the same string/sentence is said to be an ambiguous grammar.
- If E → E + E | E – E | E * E | E / E | - E | (E) | id is the given set of production rules and we want to draw the left most derivation based parse tree for id+id*id then we have two parse tree for this string.

$$E \Rightarrow E+E \Rightarrow id+E \Rightarrow id+E*E$$
$$\Rightarrow id+id*E \Rightarrow id+id*id$$



$$E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow id+E*E$$
$$\Rightarrow id+id*E \Rightarrow id+id*id$$



Ambiguity

if E_1 then if E_2 then S_1 else S_2

Interpretation-1: S2 being executed when E_1 is false (thus attaching the else to the first if)

if E_1 then (if E_2 then S_1) else S_2

Interpretation-I1: E_1 is true and E_2 is false (thus attaching the else to the second if)

if E_1 then (if E_2 then S_1 else S_2)

UPES
UNIVERSITY WITH A PURPOSE

Ambiguity

- Sometimes by specifying the rules of precedence and associativity, we can make ambiguous grammar to dis ambiguous grammar.

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

disambiguate the grammar

precedence:

^	(right to left)
*	(left to right)
+	(left to right)

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

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

$$F \rightarrow G\hat{}F \mid G$$

$$G \rightarrow id \mid (E)$$

UPES
UNIVERSITY WITH A PURPOSE

THANK YOU

UPES
UNIVERSITY WITH A PURPOSE

Dr Anurag Jain
Assistant Professor (SG), Virtualization Department, School of Computer Science,
University of Petroleum & Energy Studies, Dehradun - 248 007 (Uttarakhand)
Email: anurag.jain@ddn.upes.ac.in , dr.anuragjain14@gmail.com
Mobile: (+91) -9729371188