Unit 3.
Generative Grammars

How a string can be generate ?

A context free grammar can be used to generate strings in the corresponding language as follows:

let X = the start symbol s

while there is some nonterminal Y in X do

apply any one production rule using Y,

e.g. Y -> w

Context Free Grammars

A context free grammar G has:

- A set of terminal symbols, T
- A set of nonterminal symbols, N
- A start symbol, S, which is a member of N
- A set P of production rules of the form A -> w, where A is a nonterminal and w is a string of terminal and nonterminal symbols.

Context Free Grammar Examples

- KPL grammar is a CFG
- A simple example: the grammar of nested parentheses
  G = (N, T, P, S) where

  N = {S}

  T ={ **(**, **)** }

  P ={ S→ **(**S**)** , S→SS, S→ε }

Context Free Grammar Examples

- The grammar of decimal numbers

S -> +A | -A |A

A -> B.B | B

B -> BC | C

C -> 0 | 1 | 2 |. . . .|9

Derivations

- When X consists only of terminal symbols, it is a string of the language denoted by the grammar.
- Each iteration of the loop is a derivation step.
- If an iteration has several nonterminals to choose from at some point, the rules of derviation would allow any of these to be applied.

Leftmost and Rightmost Derivations

- In practice, parsing algorithms tend to always choose the leftmost nonterminal, or the rightmost nonterminal, resulting in strings that are leftmost derivations or rightmost derivations

Derivation Tree (parse tree)

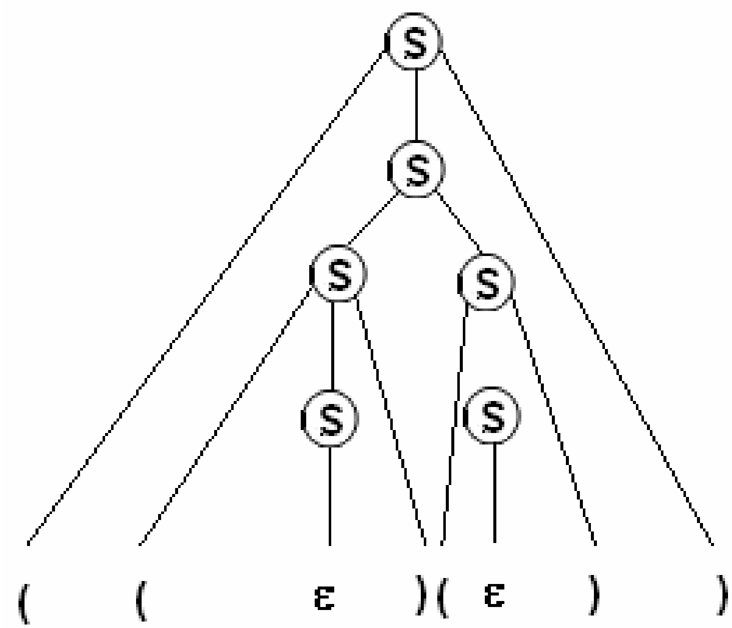
Derivation tree is constructed with

1) Each tree vertex is a variable (nonterminal) or terminal or epsilon
2) The root vertex is S
3) Interior vertices are from N, leaf vertices are from T or epsilon
4) An interior vertex A has children, in order, left to right,

   X1, X2, ... , Xk when there is a production in P of the

   form A -> X1 X2 ... Xk
5) A leaf can be epsilon only when there is a production A -> epsilon

   and the leaf's parent can have only this child.



Grammar Ambiguity

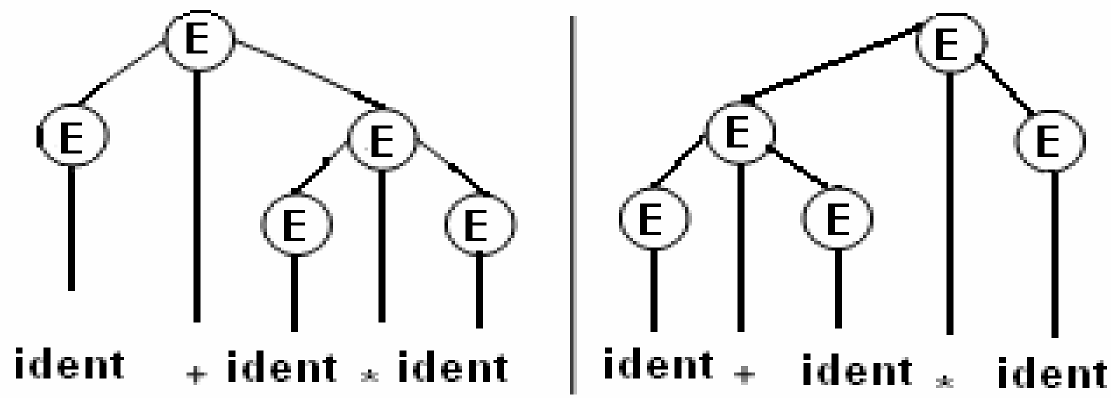
Grammar

E -> E + E

E -> E * E

E -> (E)

E -> ident



allows two different derivations for strings such as ident + ident * ident (e.g. x + y * z)

The grammar is ambiguous

Ambiguity Elimination

E -> E + T

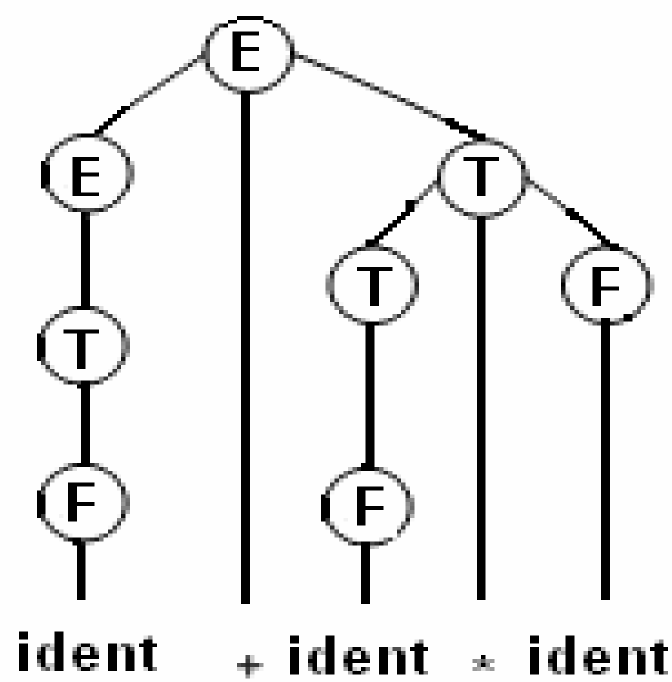
E -> T

T -> T * F

T -> F

F -> (E)

F -> ident



(by adding some nonterminals and production rules to force operator precedence)

Recursion

- **A production is recursive if** X □* ω1X ω2 Can be used to represent repetitions and nested structures

Direct recursion X □ω_1X ω_2

Left recursion X = b | Xa. X □X a □X a a □X a a a □b a a a a a ... **Right recursion** X = b | a X. X □a X □a a X □a a a X □... a a a a a b **Central recursion** X = b | "(" X")". X □(X) □((X)) □(((X))) □(((... (b)...)))

Indirect recursion X □* ω_1X ω_2
Example

Expr = Term {"+" Term}. Expr □Term □Factor □"(" Expr ")" Term = Factor {"*" Factor}. Factor = id | "(" Expr ")".

Removing Left Recursion

Let the left-recursive productions in which A occurs as lhs be

$$A \rightarrow A\alpha_1$$

....

$$A \rightarrow A\alpha_r$$

and the remaining productions in which A occurs as lhs be

$$A \rightarrow \beta_1$$

....

$$A \rightarrow \beta_s$$

Removing Left Recursion

Let K_A denote a symbol which does not already occur in the grammar.

Replace the above productions by:

$$A \to \beta_1 K_A \mid . . . \mid \beta_s K_A$$
$$K_A \to \varepsilon \mid \alpha_1 K_A \mid \ldots \mid \alpha_r K_A$$

Clearly the grammar G' produced is equivalent to G.