MATHEMATICAL FOUNDATION FOR COMPUTER SCIENCE

Prepared by: Er. Ankit Kharel

Nepal college of information technology

FINITE STATE AUTOMATA

- *Sequential Circuits and Finite state Machine*
- *Finite State Automata*
- *Non-deterministic Finite State Automata*
- *Language and Grammars*
- *Language and Automata*
- *Regular Expression*

LANGUAGE AND GRAMMAR

- Merriam-Webster's Dictionary describes language as "the words, their pronunciation, and the methods of combining them used and understood by a community" .
- But this description of language is for natural languages • The rules of natural languages are very complex and difficult to characterize completely.
- Hence, comes the Formal language .
- Formal languages are used to model natural languages and to communicate with the computers .
- As it is possible to specify completely the rules by which certain formal languages are constructed .

LANGUAGE AND GRAMMAR

- Let A be a finite set of alphabets.
- A (formal) language L over A is a subset of A^*, the set of all strings over A.
- For example: Let A = {a, b}. The set L of all strings over A containing an odd number of a's is a language over A.
- One way to define a language is to give a list of rules that the language is assumed to obey(GRAMMAR)

GRAMMAR

A grammar is also called generator that can generate the language.

Let's consider Grammar,

$S \rightarrow aA$

$A \rightarrow aA/bA/\varepsilon$

Capital Symbols : Non- terminals

Small Symbols : Terminals

$\alpha \rightarrow \beta$ is known a production rules which means α can be written as β.

<u>Example:</u>

S→aA	S→aA	S→aA
=a	=aA	=abA
	=abA	=ab
	=abaA	
	=aba	

If we use Grammar mentioned above we can make all string that starts with a.

$$L(G) = \{w \mid w \in \Sigma^*, S \xrightarrow{*} w\}$$

FORMAL DEFINITION OF GRAMMAR:

A phrase-structure grammar (or, simply, grammar) G is defined by quadruple,

$G=\{ N , T , P , \sigma \}$ where,

N = Finite set non-terminal symbols (Uppercase)

T = Finite non-empty set terminal symbols where(Lowercase)

P = Finite non-empty set of productions rules

σ = starting symbol $\sigma \in N$

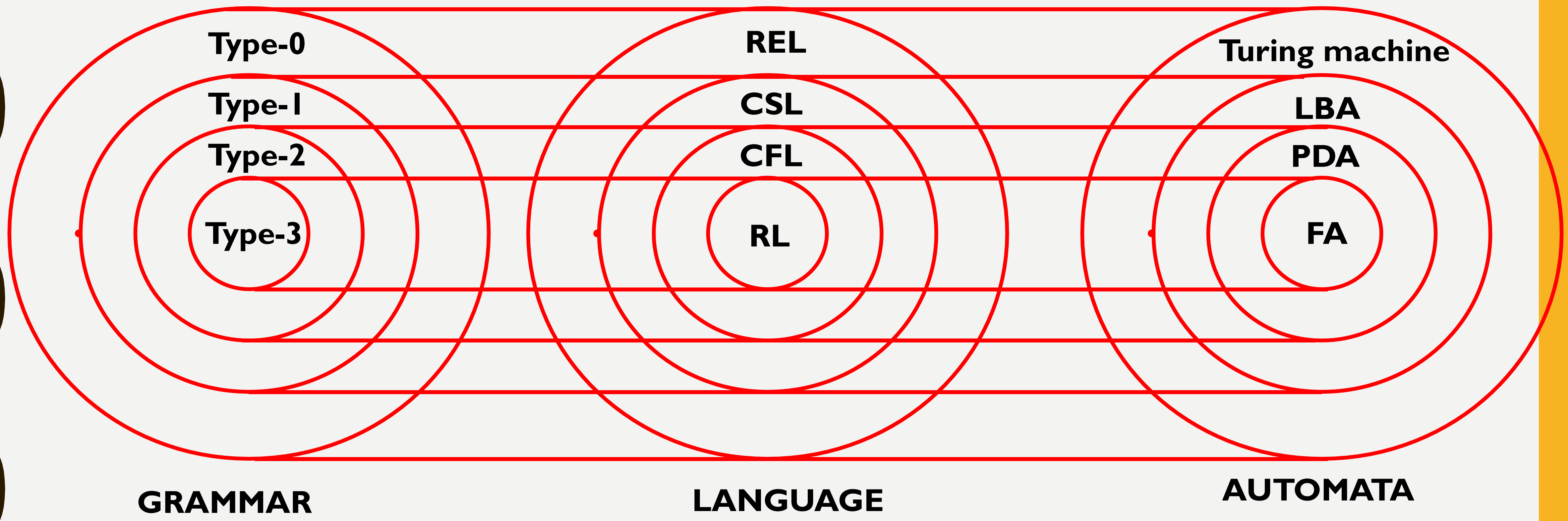
The production rule $\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta}$ is valid if:

i) $\alpha \in (T \cup N)^* N (T \cup N)^*$ i.e. α must have at least one non-terminal symbol

ii) $\beta \in (T \cup N)^*$ i.e. β can consist of any combination of nonterminal and terminal symbols.

CHOMSKY HIERARCHY:

- Chomsky Hierarchy is a brand classification of the various types of grammar available. Grammars are classified by the form of their production category represents a class of languages that can be recognized by different automata.



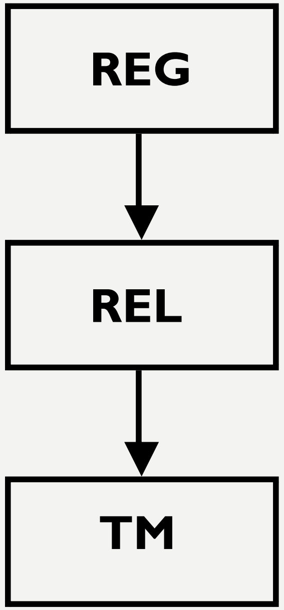
TYPE-0 (RECURSIVE ENUMERABLE GRAMMAR):

- Type – 0 Grammar(REG/Unrestricted grammar/Phase structured grammar) generates recursively enumerable language(REL). The production have no restriction. They generate the language that are recognized by a Turing Machine(TM).
- The production is in the form:

$$\alpha \rightarrow \beta \text{ ;}$$

$\alpha \in (T \cup N)^* N (T \cup N)^*$

$\beta \in (T \cup N)^*$



Example:

S→ACaB

Bc→acB

CB→DB

TYPE-1 (CONTEXT SENSITIVE GRAMMAR):

- Type – I Grammar(CSG/Length Increasing Grammar/Non-contracting grammar) generates Context Sensitive Language(CSL) which is accepted by Linearly Bounded Automata(LBA).

- The production is in the form:

$$\alpha \to \beta \;;$$

$\alpha \in (T \cup N)^* \, N \, (T \cup N)^*$

$\beta \in (T \cup N)^+$

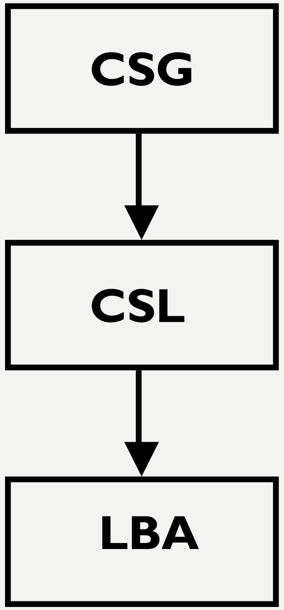
$|\alpha| \leq |\beta|$

<u>Example:</u>

AB→AbBc

A→bcA

B→a



Exception:

$S \rightarrow \varepsilon$

- S should be a start symbol but should not appear in RHS of production.

Example:

$$\left.\begin{array}{l} S \rightarrow aSb \\ S \rightarrow \varepsilon \end{array}\right\} \textbf{Not allowed} \qquad \left.\begin{array}{l} S \rightarrow AB \\ S \rightarrow \varepsilon \end{array}\right\} \textbf{Allowed}$$

$$\alpha \mathbf{A} \beta \rightarrow \alpha \, \partial \, \beta$$

Left Context **Right Context**

where $\boldsymbol{\alpha}$, $\boldsymbol{\beta} \in (N \cup T)^*$, $A \in N$ and $\partial \in (N \cup T)^+$ - the grammar is called context sensitive grammar.

<u>Example:</u>

$aAb \rightarrow aBb$

$cAd \rightarrow cCd$

TYPE-2 (CONTEXT FREE GRAMMAR):

- Type – 2 Grammar(CFG) generates Context Free Language(CFL) which is accepted by Push Down Automata(PDA).

- The production is in the form:

$$\alpha \rightarrow \beta \; ;$$

$\alpha \in N$; $|\alpha| = 1$

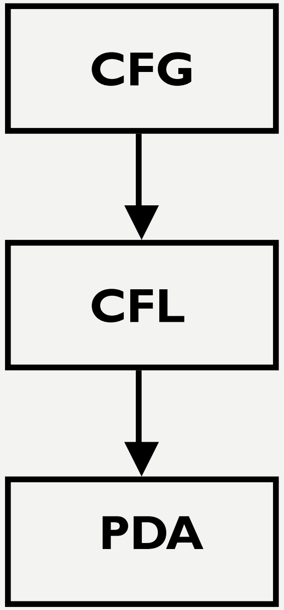
$\beta \in (T \cup N)^*$

Example:

S→Xa

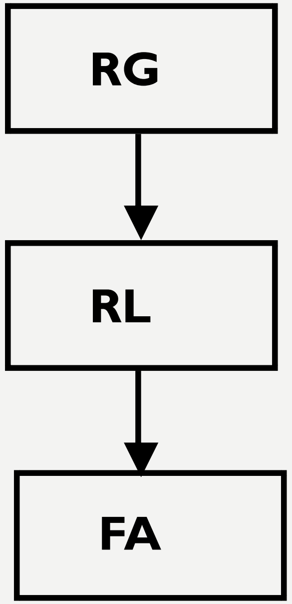
B→acB

C→a



TYPE-3 (REGULAR GRAMMAR):

- Type – 3 Grammar(RG) generates Reuglar Language(RL) which is accepted by Finite Automata(FA).



1. Left Linear Grammar:

   A→a

   A→Ba

   - $A, B \in N$
   - $|A| = |B| = 1$
   - $a \in T^*$

Example:

A→abc

A→aBa(invalid)

A→Ca

1. Right Linear Grammar:

   A→a

   A→aB

   - $A, B \in N$
   - $|A| = |B| = 1$
   - $a \in T^*$

Example:

A→a

A→aBa(invalid)

A→Ca(invalid)

A→aC

Q. Consider the following Grammar:

$$S \rightarrow ACaB$$
$$Bc \rightarrow acB$$
$$CB \rightarrow DB$$
$$aD \rightarrow Db$$

Determine whether the given grammar is Context-sensitive, Context-Free, Regular or None of these.

Solution:

The Given grammar is:

$$S \rightarrow ACaB$$
$$Bc \rightarrow acB$$
$$CB \rightarrow DB$$
$$aD \rightarrow Db$$

(a)Checking For Regular(Type-3)

The production rule for regular grammar is given by,

$$A \rightarrow a$$
$$A \rightarrow Ba$$
$$A, B \in N$$
$$|A| = |B| = 1$$
$$a \in T^*$$

Since the production, $S \rightarrow ACaB$ violates the rule, It is not REGULAR GRAMMAR (b)Checking For Context- Free(Type-2)

The production rule for Context-Free grammar is given by,

$$\boldsymbol{\alpha \rightarrow \beta ;}$$

$$\alpha \in N ; |\alpha| = 1$$

$$\beta \in (T \cup N)^*$$

Since the production, Bc → acB violates the rule, It is not CONTEXT FREE GRAMMAR.

(c)Checking For Context- Sensitive (Type-1)

The production rule for Context-Sensitive grammar is given by,

$$\boldsymbol{\alpha \rightarrow \beta ;}$$

$$\alpha \in (T \cup N)^* N (T \cup N)^*$$

$$\beta \in (T \cup N)^*$$

$$|\alpha| \leq |\beta|$$

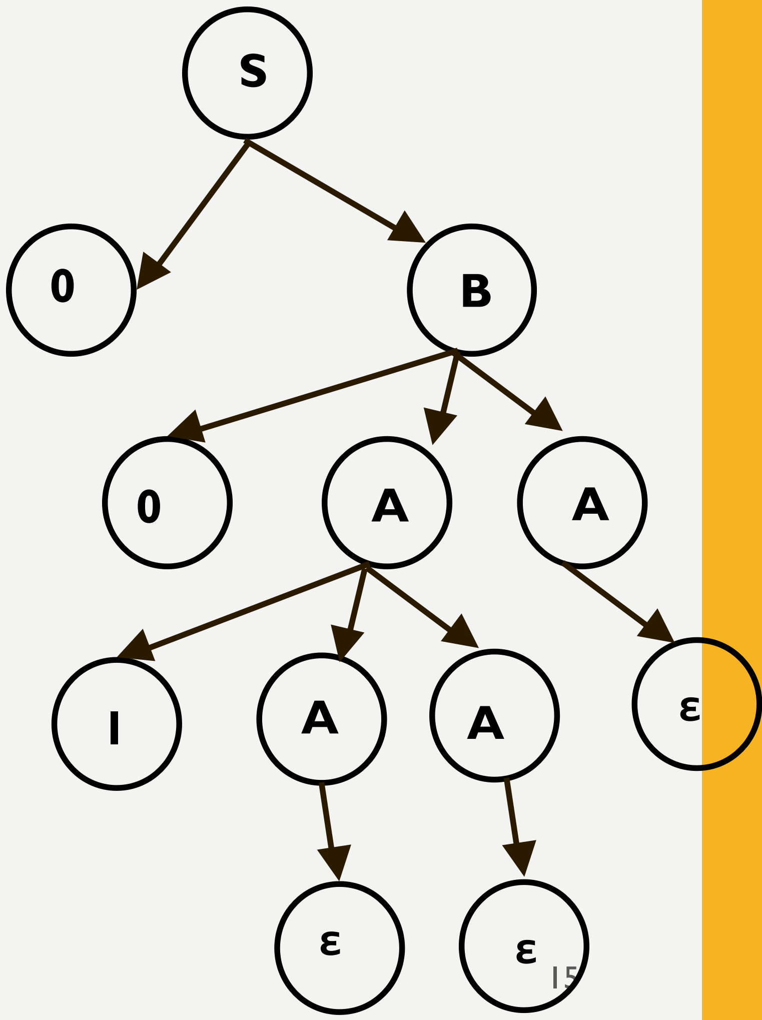
Ever production given in the grammar satisfies above rule, Therefore, it is

CONTEXT SENSITIVE GRAMMAR

DERIVATION TREE FOR CFG:

- A derivation Tree or Parse Tree is an ordered rooted tree that graphically represents the semantic information of string derived from a Context Free Grammar.

  1. Root Vertex : Must be labelled by start symbol
  2. Vertex : Labelled by Non- Terminal symbols
  3. Leaves : Labelled by Terminal Symbols



- Consider the following grammar:

  G={ N ,T , P , σ } where Production rule is given by:

  S→0B

  A→1AA/ ε

  B→0AA

Construct Derivation Tree for the string "001"

S→0B

00AA

001AAA

001

DERIVATION TREE FOR CFG:

1. **LEFTMOST DERIVATION:**

   A leftmost Derivation Tree is obtained by applying production function to the leftmost variable in each step.

Consider the following grammar:

G={ N ,T , P , σ } where Production rule is given by:

S→aAS/aSS/e

A→SbA/ba

Construct Derivation Tree for the string "aabaa"

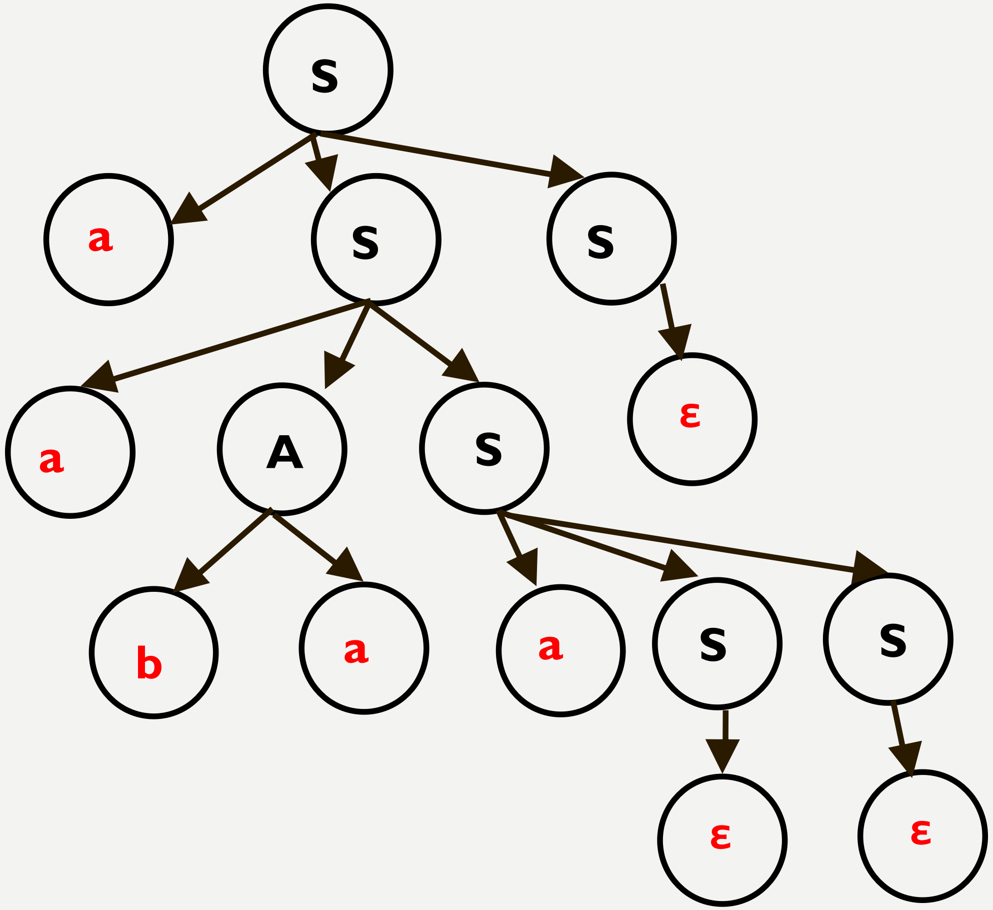
S→aSS

aaASS

aabaSS

aabaaSSS

aabaa



DERIVATION TREE FOR CFG:

2. RIGHTMOST DERIVATION:

A rightmost Derivation Tree is obtained by applying production function to the rightmost variable in each step.

Consider the following grammar:

G={ N ,T , P , σ } where Production rule is given by:

S→aAS/aSS/e

A→SbA/ba

Construct Derivation Tree for the string "aabaa"

S→aSS

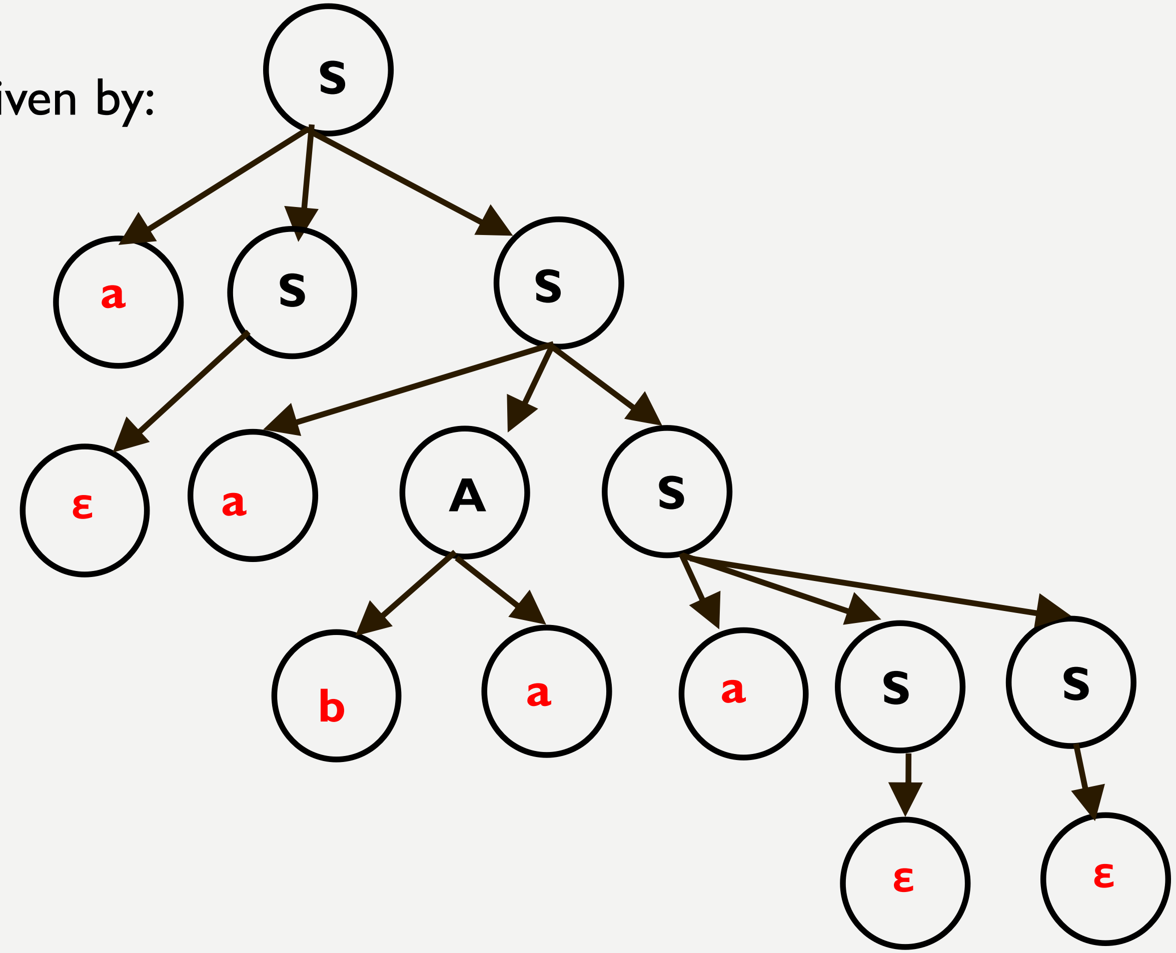
aSaAS

aSaAaSS

aSaAa

aSabaa

aabaa



DERIVATION TREE FOR CFG:

G={ N ,T , P , σ } where Production rule is given by:

S→aB/bA

A→a/aS/bAA

B→b/bS/aBB

Construct left Derivation Tree for the string "aabbabba"

S→aB

aaBB

aabSB

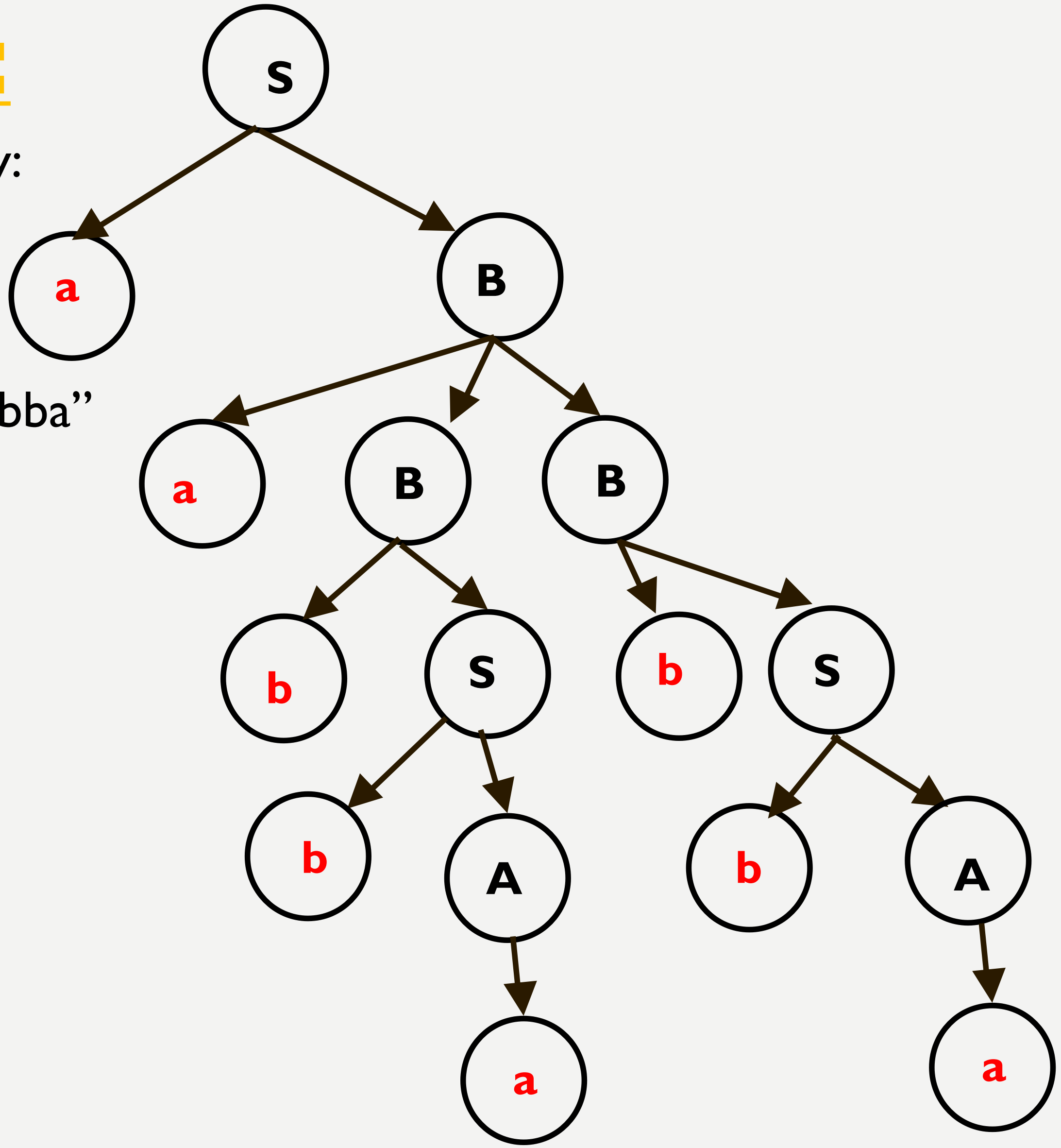
aabbAB

aabbaB

aabbabS

aabbabbA

aabbabba



DERIVATION TREE FOR CFG:

G={ N ,T , P , σ } where Production rule is given by:

S→aB/bA

A→a/aS/bAA

B→b/bS/aBB

Construct right Derivation Tree for the string "aabbabba"

S→aB

aaBB

aaBbS

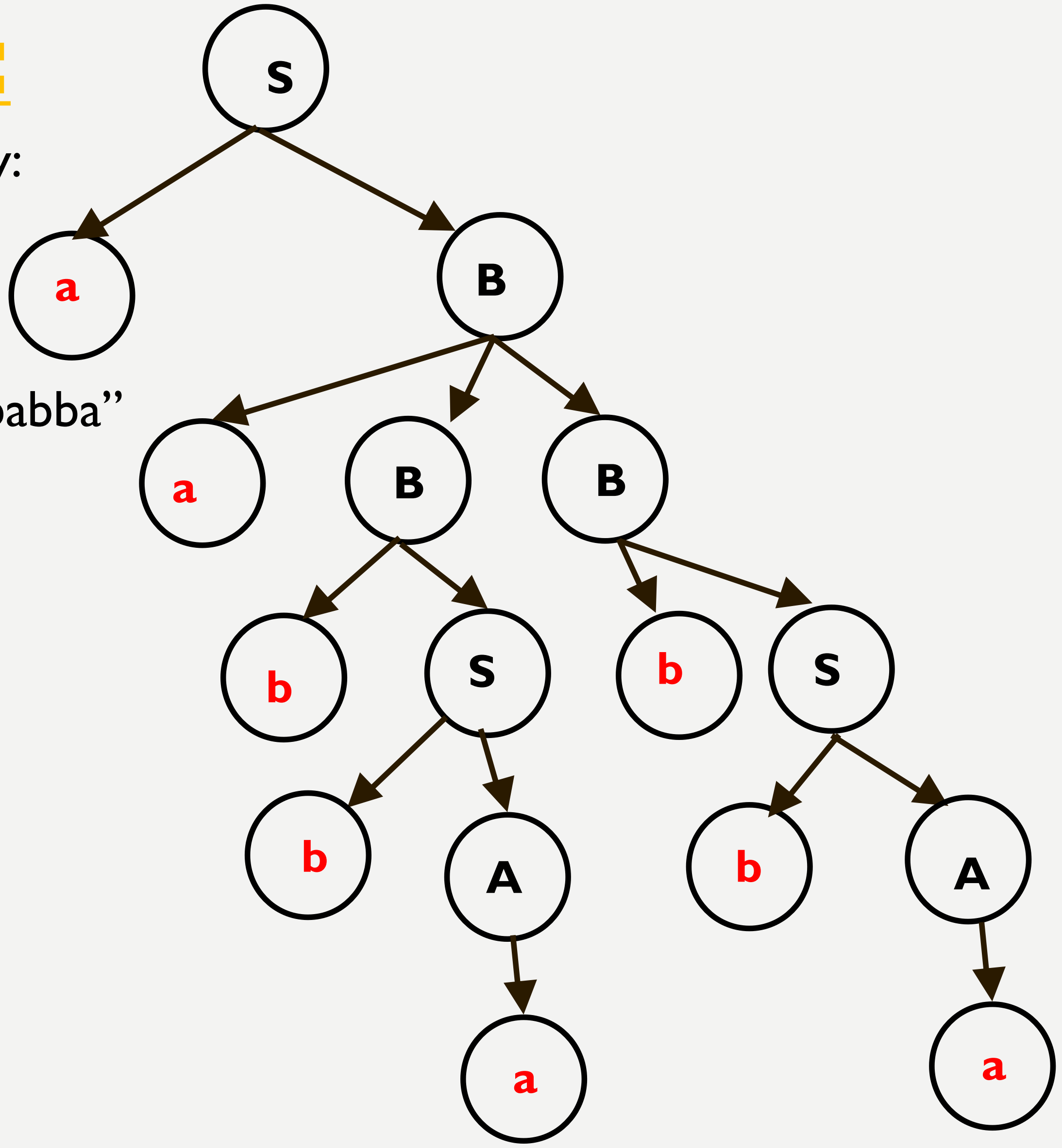
aaBbbA

aaBbba

aabSbba

aabbAbba

aabbabba



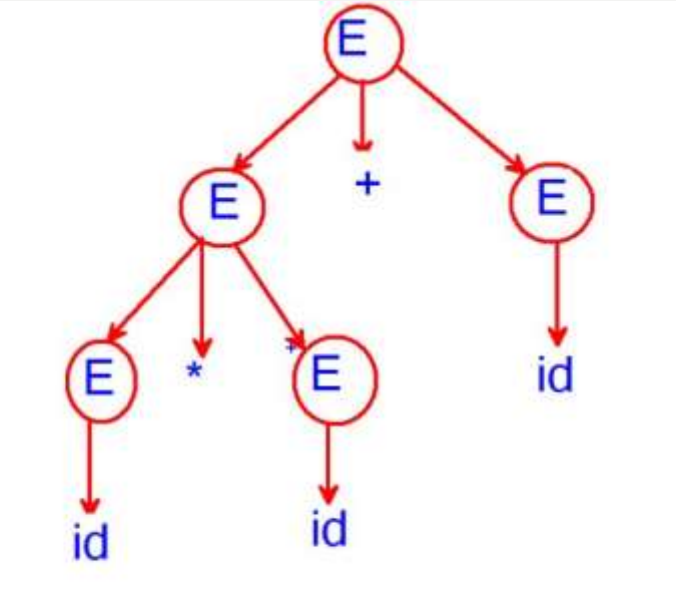
DERIVATION TREE FOR CFG:

Consider the grammar G={ N ,T , P , σ } where
Production rule is given by:

E→E + E
E→E * E
E→(E)
E→id

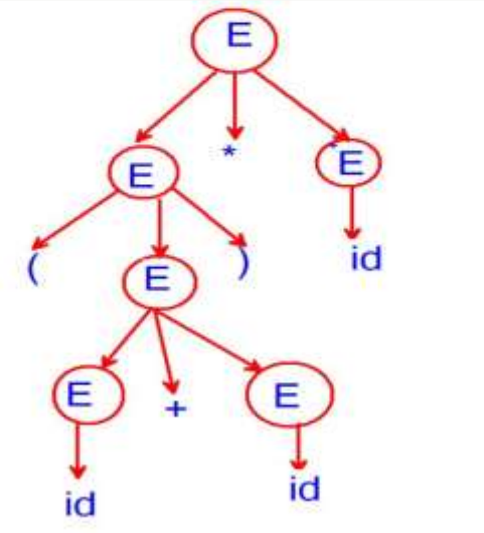
Construct Derivation Tree for the **id * id + id**

E→E + E
E * E + E
id * id + id



Construct Derivation Tree for the **(id + id) * id**

E→E * E
(E) * E
(E + E) * E
(id + id) * id



BNF(BACKUS NORMAL FORM):

- An alternative way to state of productions of a grammar is by using Backus Normal Form(BNF). It is meta syntax for CFG.

Syntax:

<LHS> : : = RHS

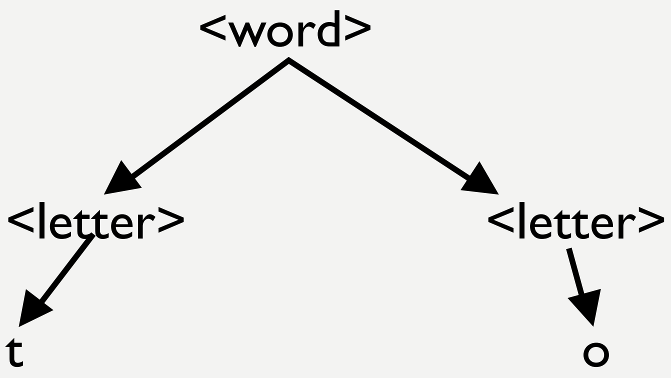
(Non – terminals) *(Terminals)*

Example:

<letter> : : = a/b/c/d/e/t/o

<word> : := <letter><letter>

(This generates word consisting of two letter)



BNF(BACKUS NORMAL FORM):

- **Grammer for integers:**

  An integer is defined as a string consisting of an optional sign(+ or -) followed by a string of digits(0 though 9)

The following Grammar generates all string:

<digit> : : = 0/1/2/3/4/5/6/7/8/9
<sign> : : = +/-
<unsigned integer> : : = <digit>/<digit><unsigned integer>
<signed integer> : : = <sign><unsigned integer>
<integer> : : = <signed integer>/<unsigned integer>

Derive integer -102 using above grammar and construct derivation tree.

BNF(BACKUS NORMAL FORM):

<digit> : : = 0/1/2/3/4/5/6/7/8/9

<sign> : : = +/-

<unsigned integer> : : = <digit>/<digit><unsigned integer>

<signed integer> : : = <sign><unsigned integer>

<integer> : : = <signed integer>/<unsigned integer>

<integer> : : = <signed integer>
<sign><unsigned integer>
-<digit><unsigned integer>
-1<digit><unsigned integer>
-10<digit>
-102