Grammar

&

Languages

- **Example:**

<sentence> –> <noun-phrase> <verb-phrase>	(1)
<noun-phrase> –> <proper-noun>	(2)
<noun-phrase> –> <determiner> <common-noun>	(3)
<proper-noun> –> Ram	(4)
<proper-noun> –> Sham	(5)
<common-noun> –> car	(6)
<common-noun> –> Sangli	(7)
<determiner> –> a	(8)
<determiner> –> the	(9)
<verb-phrase> –> <verb> <adverb>	(10)
<verb-phrase> –> <verb>	(11)
<verb> –> drives	(12)
<verb> –> eats	(13)
<adverb> –> slowly	(14)
<adverb> –> frequently	(15)

- **Derivation forming a sentence:**

<sentence> => <noun-phrase> <verb-phrase>	by (1)
=> <proper-noun> <verb-phrase>	by (2)
=> Ram <verb-phrase>	by (4)
=> Ram <verb> <adverb>	by (10)
=> Ram drives <adverb>	by (12)
=> Ram drives frequently	by (15)

- **Informally, Grammar consists of:**
  - A set of replacement ***rules***,

    each having a Left-Hand Side (LHS) and a Right-Hand Side (RHS)
  - Two types of symbols; ***variables* and *terminals***
  - LHS of each rule must have at *least one variable* to generate
  - RHS of each rule is a string of *zero or more variables and terminals*
  - A *string* consists of *only terminal*

Example

$L(G3) = \{a^n b^n c^n \mid n >= 1\}$

// Let G=(Vn,Vt,S,P)//

G3=({S,B,C}, {a,b,c}, S P)

S→ aSBC

S→aBC

CB→BC

aB→ab

bB→bb

bC→bc

cC→cc

S→aSBC

→aaBCBC

→aaBBCC

→aabBCC

→aabbCC

→aabbcC

→aabbcc

Example of CFGs

Simple arithmetic expressions:

$$E \rightarrow \text{int}$$
$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow (E)$$

- One non-terminal: E
- Several terminals: int, +, *, (,)
    - Called terminals because they are never replaced
- By convention the non-terminal for the first production is the start one

Derivation Example

- Grammar

  $E \rightarrow E + E \mid E * E \mid (E) \mid int$

- String

  $int * int + int$

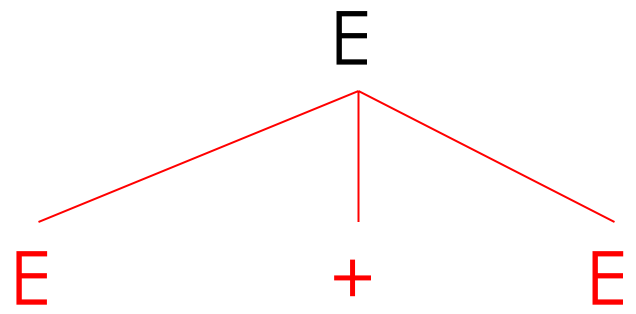
Derivation in Detail (1)

E

E

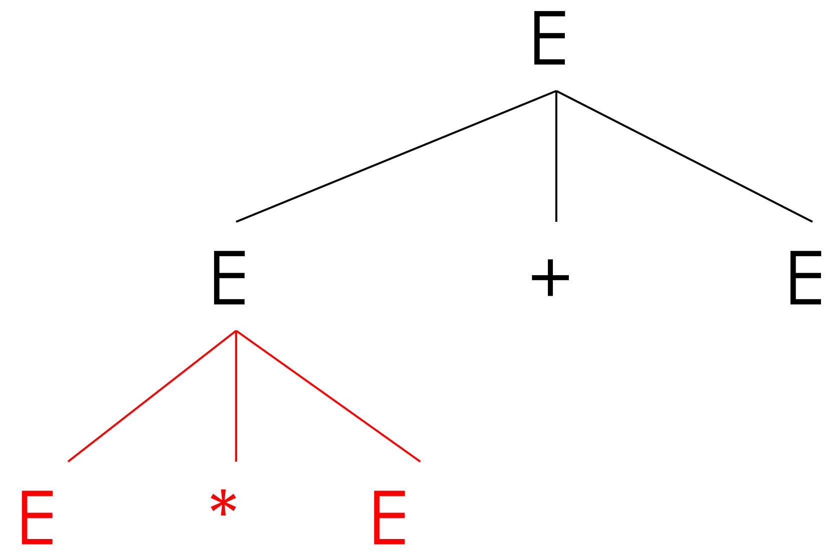
Derivation in Detail (2)

$$E$$
$$\rightarrow E + E$$



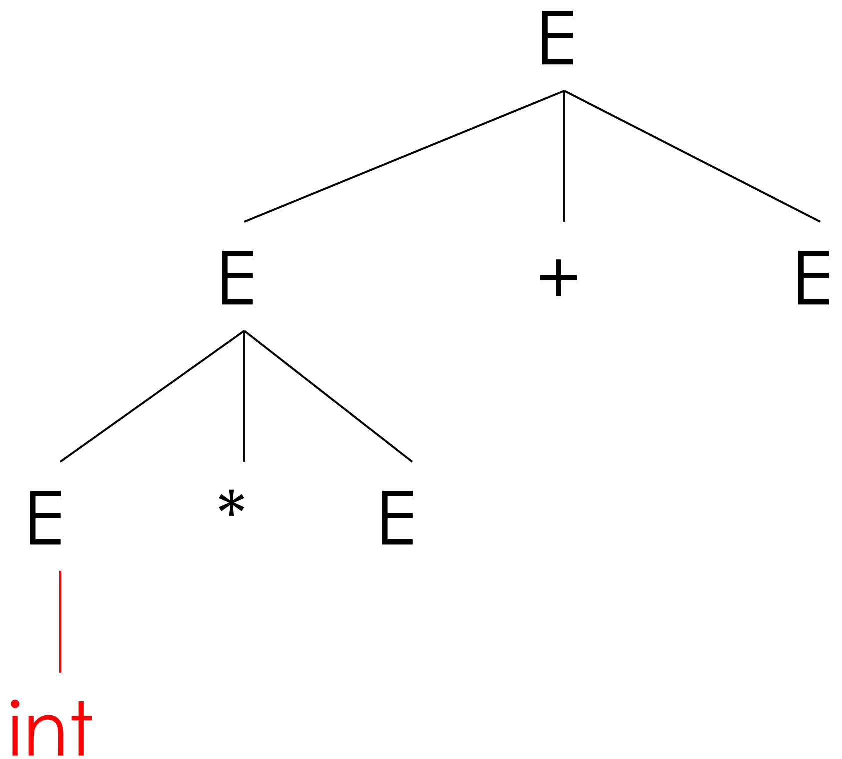
Derivation in Detail (3)

$$
\begin{aligned}
& E \\
\rightarrow\quad & E + E \\
\rightarrow\quad & E * E + E
\end{aligned}
$$



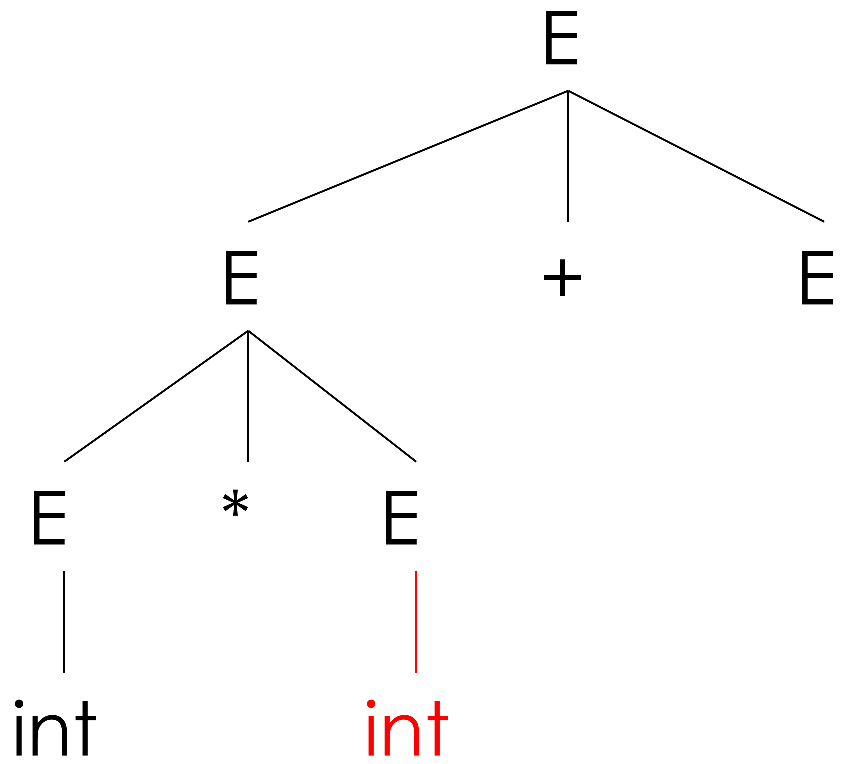
Derivation in Detail (4)

$$
\begin{aligned}
& E \\
\rightarrow\quad & E + E \\
\rightarrow\quad & E * E + E \\
\rightarrow\quad & \text{int} * E + E
\end{aligned}
$$



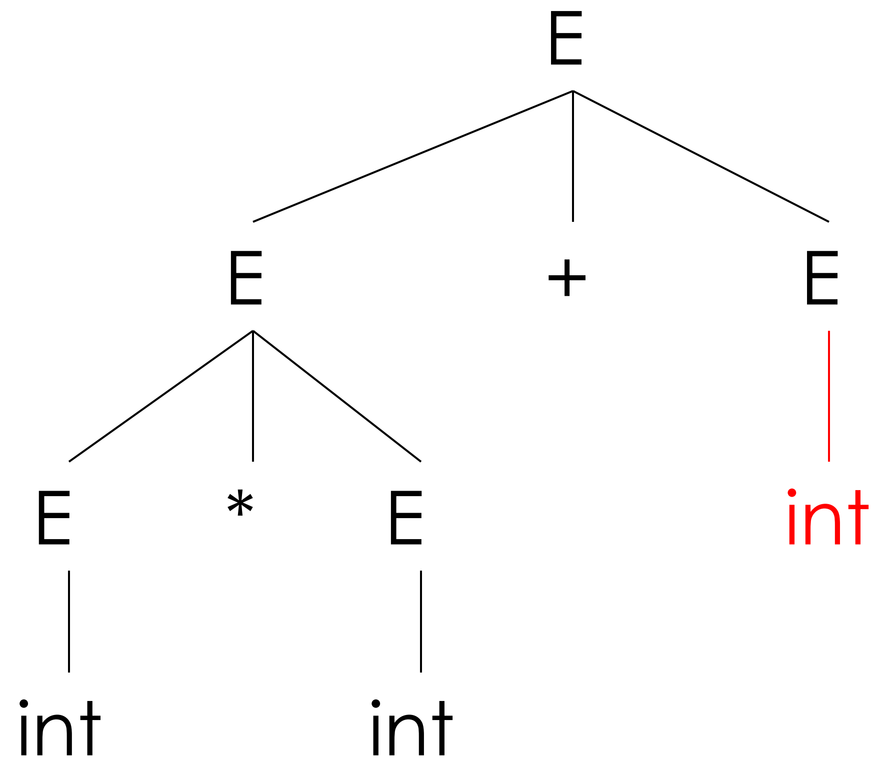
Derivation in Detail (5)

$$
\begin{aligned}
& E \\
\rightarrow \quad & E + E \\
\rightarrow \quad & E * E + E \\
\rightarrow \quad & \text{int} * E + E \\
\rightarrow \quad & \text{int} * \text{int} + E
\end{aligned}
$$



Derivation in Detail (6)

$$
\begin{aligned}
& E \\
\rightarrow\ & E + E \\
\rightarrow\ & E * E + E \\
\rightarrow\ & \text{int} * E + E \\
\rightarrow\ & \text{int} * \text{int} + E \\
\rightarrow\ & \text{int} * \text{int} + \text{int}
\end{aligned}
$$



Grammar

Let $G=(Vn,Vt,S,P)$

Vn→ Finite set of non-terminals

Vt→ Finite set of terminals

S→ starting symbol; $S \in Vn$

P→ Finite set of production rules

$\alpha \xrightarrow{P} \beta \qquad \alpha, \beta \in (Vn \cup Vt)^*$

$(Vn \cup Vt)^* \, Vn \, (Vn \cup Vt)^* \rightarrow (Vn \cup Vt)^*$

Types of Languages

- Type 0 – Unrestricted Language
- Type 1 – Context Sensitive Language
- Type 2- Context Free Language
- Type 3- Regular Language

Definitions

Type 0 - Unrestricted Language

$$\alpha \xrightarrow{p} \beta$$

$\alpha, \beta \in (Vn \cup Vt)^*$

Definitions

Type 1 - Context Sensitive Language

$$\alpha \xrightarrow{P} \beta$$

$\alpha, \beta \in (Vn \cup Vt)^*$

$|\alpha| <= |\beta|$

Definitions

Type 2 - Context Free Language

$$\alpha \xrightarrow{p} \beta$$

$\alpha, \beta \in (Vn \cup Vt)^*$

$|\alpha| <= |\beta|$; $\alpha \in Vn$

Definitions

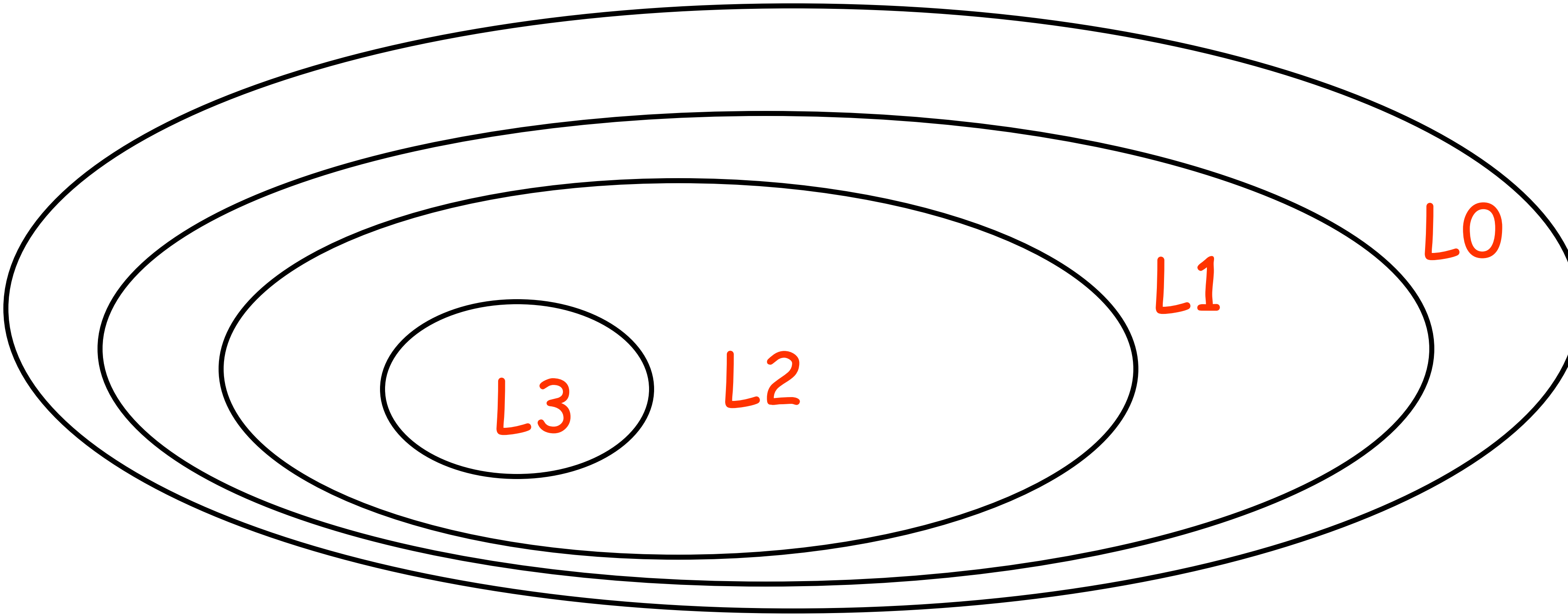
Type 3 - Regular Language

$$\alpha \xrightarrow{p} \beta$$

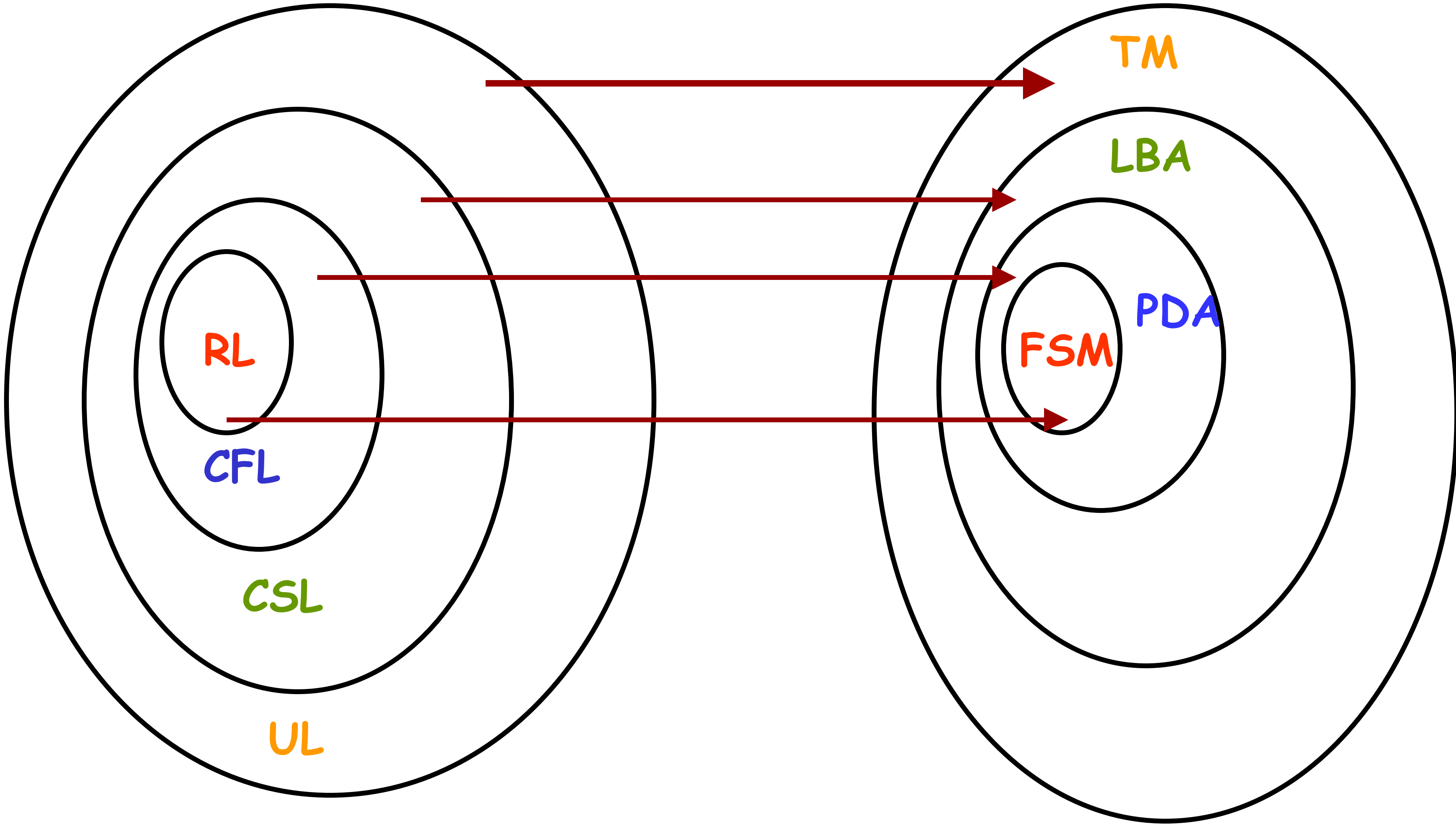
$\alpha, \beta \in (Vn \cup Vt)^*$

$|\alpha| <= |\beta|$; $\alpha \in Vn$; $\beta = mN$, $m \in Vt$ & $N \in Vn$

Relations



Relations



Sample CFG

1. $E \rightarrow I$ // Expression is an identifier
2. $E \rightarrow E+E$ // Add two expressions
3. $E \rightarrow E*E$ // Multiply two expressions
4. $E \rightarrow (E)$ // Add parenthesis
5. $I \rightarrow L$ // Identifier is a Letter
6. $I \rightarrow ID$ // Identifier + Digit
7. $I \rightarrow IL$ // Identifier + Letter
8. $D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$ // Digits
9. $L \rightarrow a \mid b \mid c \mid \ldots A \mid B \mid \ldots Z$ // Letters

Regular Expression: (letter)(letter + digit)*

Example CFG for $\{0^k1^k \mid k \geq 0\}$:

G = ({S}, {0, 1}, P, S) *// Remember: G = (V, T, P, S) //*

P:

(1) $S \rightarrow 0S1$ or just simply $S \rightarrow 0S1 \mid \varepsilon$

(2) $S \rightarrow \varepsilon$

- **Example Derivations:**

$S \rightarrow \varepsilon$ (2) $\Rightarrow \varepsilon$ //string1//

$S \rightarrow 0S1$ (1)

$S \rightarrow \varepsilon$ (2) $\Rightarrow 01$ //string2//

Hence,

$S \rightarrow 0S1$ (1)

$\rightarrow 00S11$ (1)

$\rightarrow 000S111$ (1)

$\rightarrow 000111$ (2)

Example CFG for Language of palindromes

$\Sigma = \{0,1\}$

$S \rightarrow \varepsilon$
$S \rightarrow 0$
$S \rightarrow 1$
$S \rightarrow 0S0$
$S \rightarrow 1S1$

More compactly: $S \rightarrow \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$

Hence,
$S \rightarrow 0S0$
$\rightarrow 01S10$
$\rightarrow 01010$

Example CFG for Language:
L= $\{a^m\ b^n\ c^{m+n} \mid m,n \geq 0\}$

Rewrite as $\{a^m\ b^n\ c^n\ c^m \mid m,n \geq 0\}$:

$S \to S' \mid \mathbf{a}\ S\ \mathbf{c}$

$S' \to \varepsilon \mid \mathbf{b}\ S'\ \mathbf{c}$

Hence,

$S \to aSc$

$\to aaS'cc$

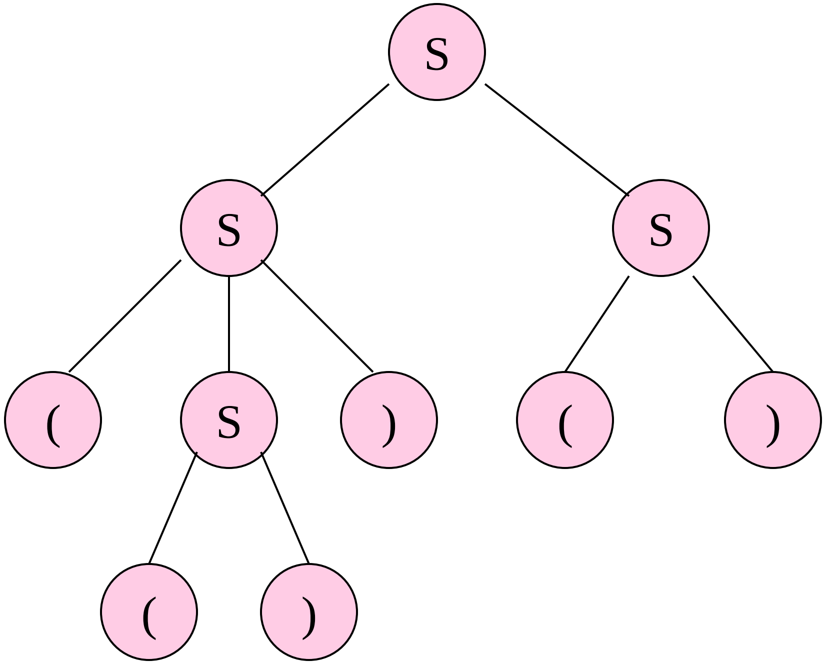
$\to aabS'ccc$

$\to aab\ \varepsilon\ ccc$

$\to aabccc$

Example Parse Tree

$S \rightarrow SS \mid (S) \mid (\ )$



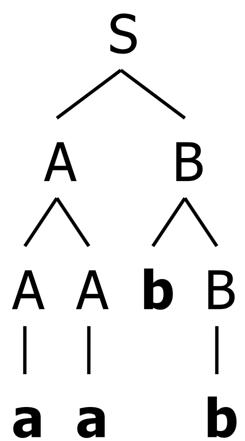
Parse Trees

$S \ \rightarrow A \mid A\ B$
$A \ \rightarrow \varepsilon \mid \mathbf{a} \mid A\ \mathbf{b} \mid A\ A$
$B \ \rightarrow \mathbf{b} \mid \mathbf{b}\ \mathbf{c} \mid B\ \mathbf{c} \mid \mathbf{b}\ B$

➔ Sample derivations:

$S \Rightarrow AB \Rightarrow AAB \Rightarrow \mathbf{a}AB \Rightarrow \mathbf{aa}B \Rightarrow \mathbf{aab}B \Rightarrow \mathbf{aabb}$
$S \Rightarrow AB \Rightarrow A\mathbf{b}B \Rightarrow A\mathbf{bb} \Rightarrow AA\mathbf{bb} \Rightarrow A\mathbf{abb} \Rightarrow \mathbf{aabb}$

- These two derivations use same productions, but in different orders
- This ordering difference is often uninteresting
- ***Derivation trees give way to abstract away ordering differences***



- **Example:**

S –> AB

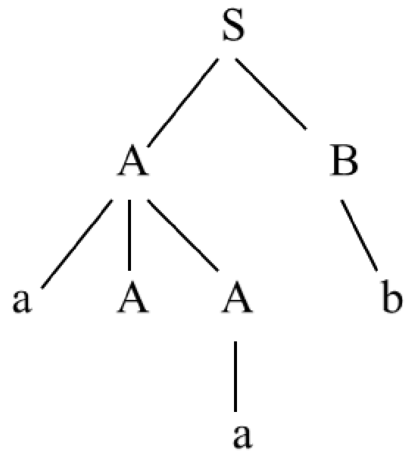
A –> aAA

A –> aA

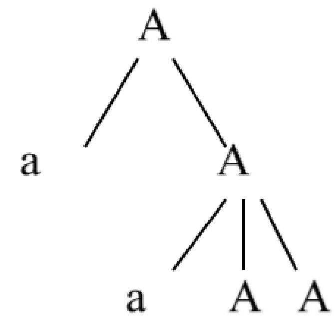
A –> a

B –> bB

B –> b



yield = aAab



yield = aaAA

<u>Derivation / Parse Tree-</u>

- Root node of a parse tree is the start symbol of the grammar
- Each leaf node of a parse tree represents a terminal symbol
- Each interior node of a parse tree represents a non-terminal symbol
- Parse tree is independent of the order in which the productions are used during derivations
- Concatenating the leaves of a parse tree from the left produces a string of terminals
- This string of terminals is called as **yield of a parse tree**

Derivation Trees

$S \rightarrow A \mid A\ B$
$A \rightarrow \varepsilon \mid \mathbf{a} \mid A\ \mathbf{b} \mid A\ A$
$B \rightarrow \mathbf{b} \mid \mathbf{b}\ \mathbf{c} \mid B\ \mathbf{c} \mid \mathbf{b}\ B$

w = **aabb**

Other derivation trees for this string?

?

Derivation Trees

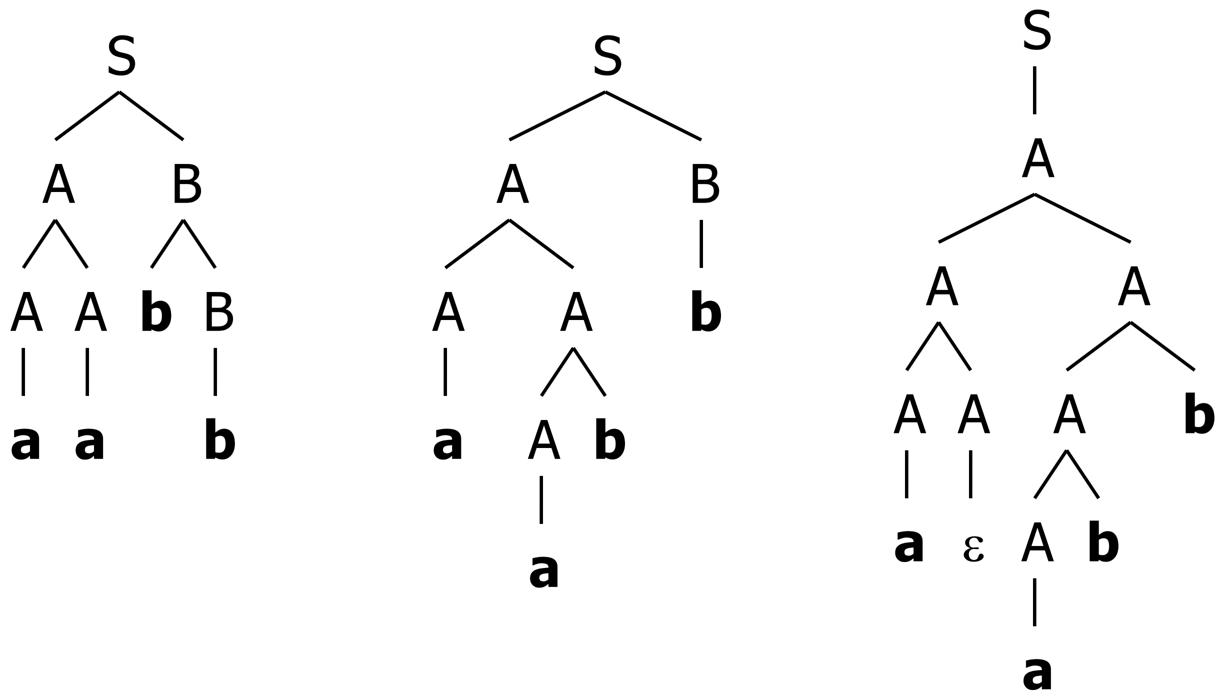
$S \rightarrow A \mid A\,B$

$A \rightarrow \varepsilon \mid \mathbf{a} \mid A\,\mathbf{b} \mid A\,A$

$B \rightarrow \mathbf{b} \mid \mathbf{b}\,\mathbf{c} \mid B\,\mathbf{c} \mid \mathbf{b}\,B$

w = **aabb**

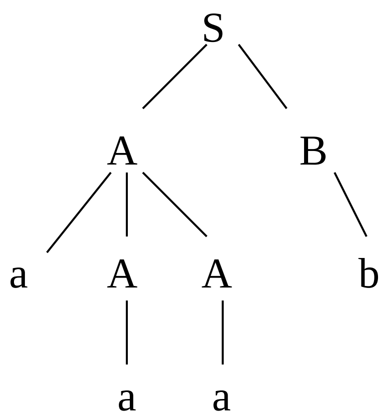
Other derivation trees for this string?



..few more are also possible...

- **Observation1:** Every derivation should correspond to atleast one derivation tree.

S => AB
=> aAAB
=> aaAB
=> aaaB
=> aaab



Rules:
S –> AB
A –> aAA
A –> aA
A –> a
B –> bB
B –> b

- **Observation2:** Every derivation tree may correspond to one or more derivations.

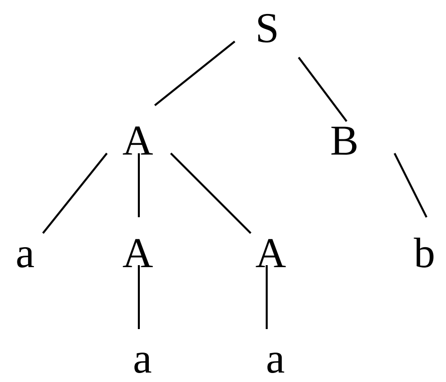
leftmost:	*rightmost:*	*mixed:*
S => AB	S => AB	S => AB
=> aAAB	=> Ab	=> Ab
=> aaAB	=> aAAb	=> aAAb
=> aaaB	=>aAab	=> aaAb
=> aaab	=> aaab	=> aaab

Definition: A derivation is *leftmost (rightmost)* if at each step in the derivation a production is applied to the leftmost (rightmost) non-terminal in the sentential form.

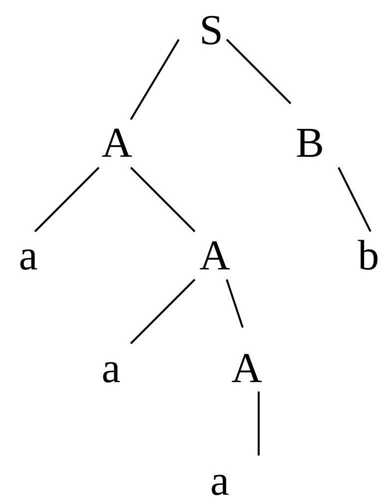
- **Example:** Consider the string *aaab* and the preceding grammar:

S –> AB
A –> aAA
A –> aA
A –> a
B –> bB
B –> b

S => AB
=> aAAB
=> aaAB
=> aaaB
=> aaab



S => AB
=> aAB
=> aaAB
=> aaaB
=> aaab



Note: The string has two left-most derivations, and therefore has two distinct parse trees.

- **Definition:** Let G be a CFG. Then G is said to be <u>ambiguous</u> if there exists an x in L(G) with >1 leftmost derivations or >1 rightmost derivations.

Note:

- Given a CFL, there may be more than one CFG with

  L = L(G1) = L(G2)

  However, G1 and/or G2 may not be ambiguous.
- Some CFLs can have both ambiguous and unambiguous grammars
- Some CFLs, however, can be generated only by an ambiguous grammar
- *A CFL that can be generated only by ambiguous grammars is called inherently ambiguous*

i.e. Let L be a CFL. If every CFG G with

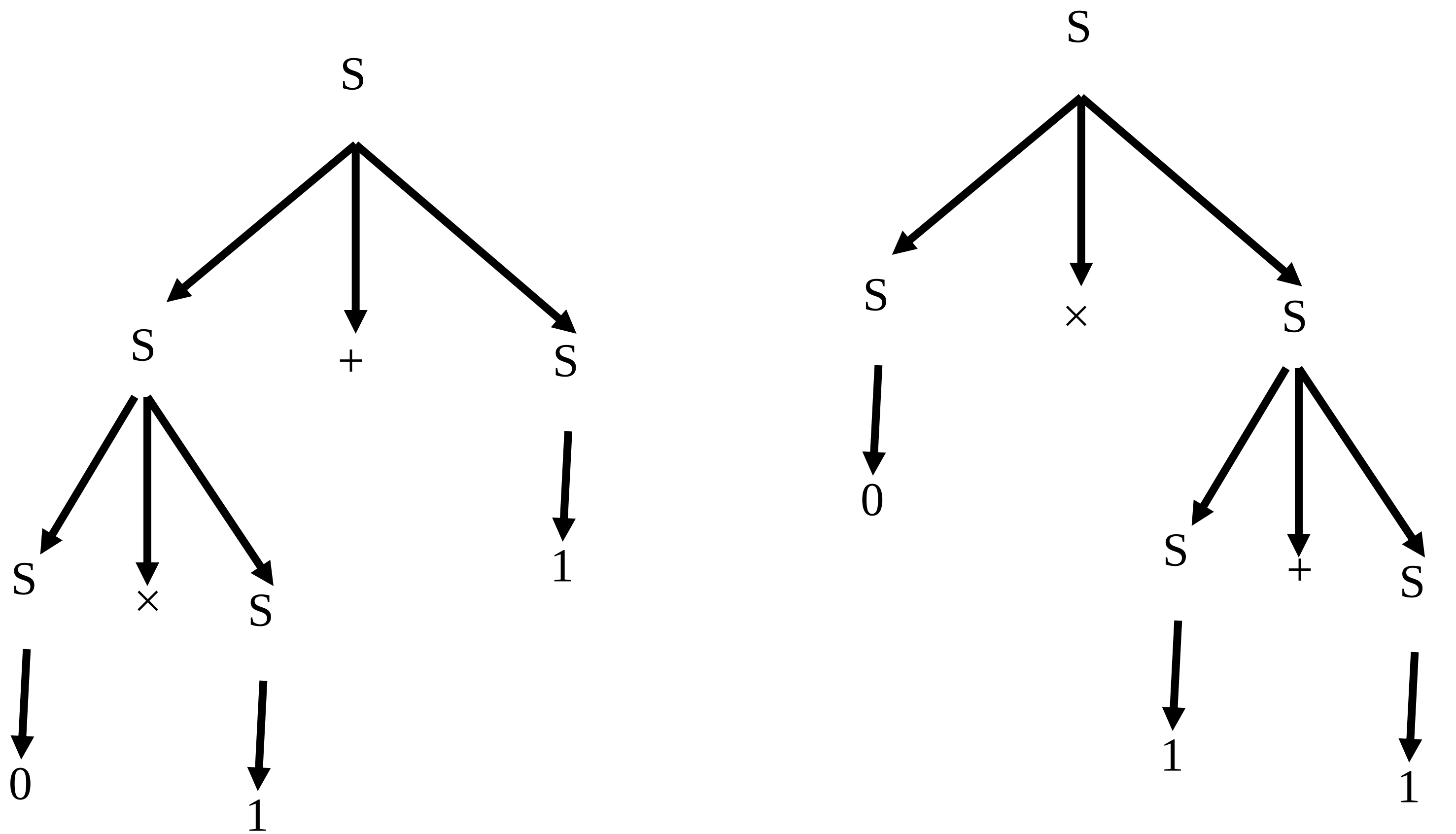
L = L(G) is ambiguous, then L is <u>inherently ambiguous</u>.

Example of inherently ambiguous Grammar producing:

$L = \{0^i 1^j 2^k \mid i=j \vee j=k\}$

Ambiguity and Derivation Trees

W= 0×1+1



E → I $\quad \sum$ **={0,…,9, +, *, (,)}**

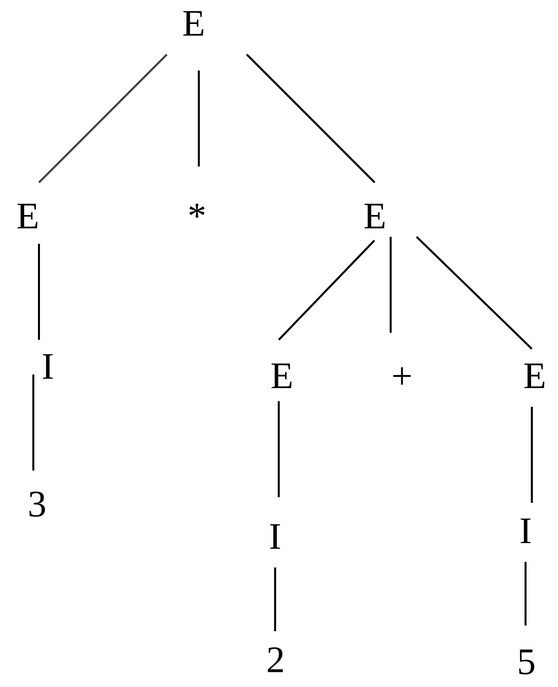
E → E + E

E → E * E

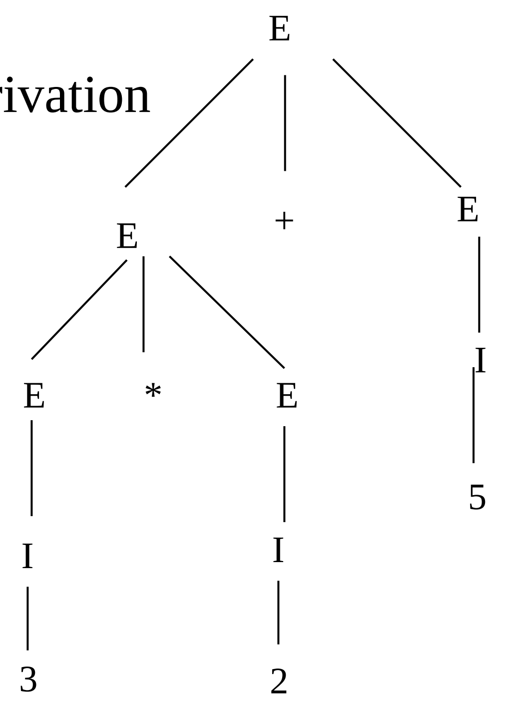
E → (E)

I → ε | 0 | 1 | … | 9

E=>E*E
=>I*E
=>3*E+E
=>3*I+E
=>3*2+E
=>3*2+I
=>3*2+5



Another leftmost derivation
E=>E+E
=>E*E+E
=>I*E+E
=>3*E+E
=>3*I+E
=>3*2+I
=>3*2+5



Linear Grammar

- A **linear grammar** is a CFG that has at most one nonterminal in the right hand side of each of its productions
- A **linear language** is a language generated by some linear grammar

----

- the **left-linear** or left-regular grammars, in which **all nonterminals** in right-hand sides are **at the left ends**
- the **right-linear** or right-regular grammars, in which **all nonterminals** in right-hand sides are **at the right ends**
- A Regular Grammar is a grammar that is left-linear or right-linear

Q) If Regular Grammar is ambiguous?

-**Regular grammar** is either right or left linear *(all terminals grouped to one end);* whereas **context free grammar** is basically any combination of terminals and non-terminals. ...

- **Regular grammars** are non-ambiguous; there is only one production rule for a given non-terminal, whereas there can be more than one **in the** case of a **context**-**free grammar**.

Further CFG…

An ambiguous Grammar: Ex2

$S \rightarrow AB \mid CD$

$A \rightarrow 0A1 \mid 01$ //A generates equal 0's and 1's

$B \rightarrow 2B \mid 2$ // B generates any number of 2's

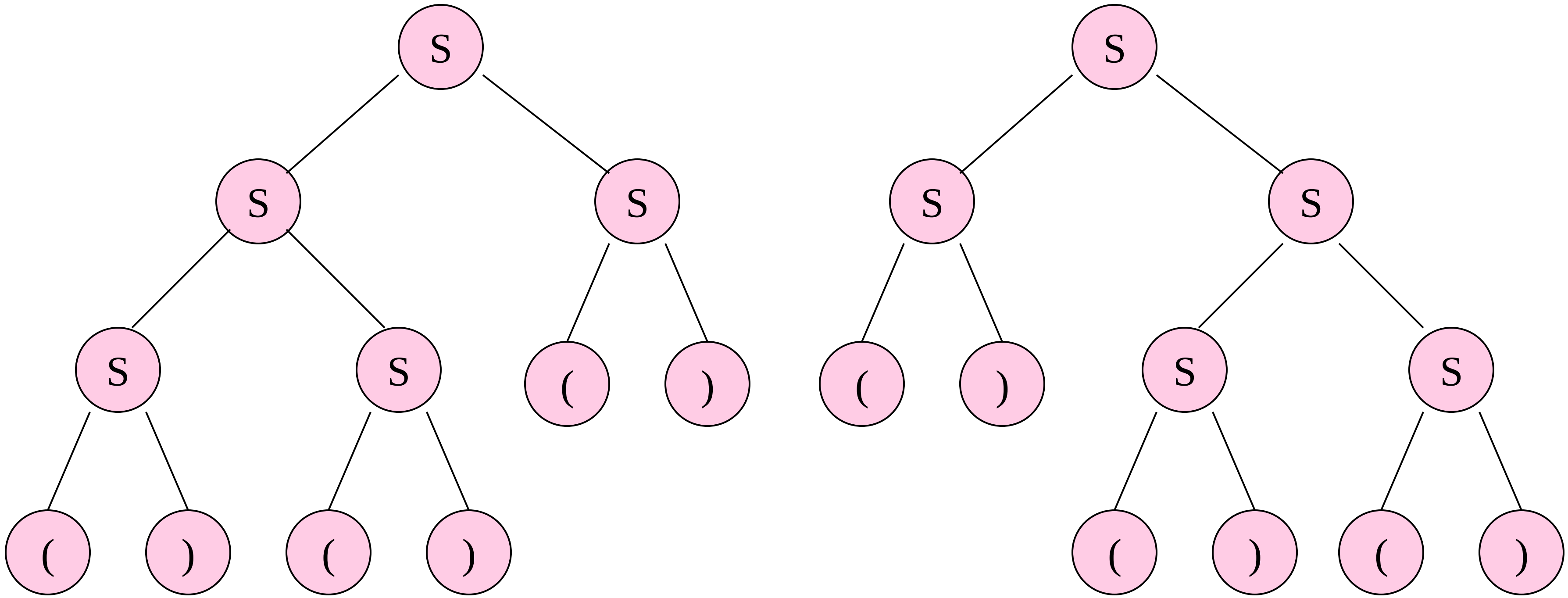
$C \rightarrow 0C \mid 0$ // C generates any number of 0's

$D \rightarrow 1D2 \mid 12$ // D generates equal 1's and 2's

And there are two derivations of every string
with equal numbers of 0's, 1's, and 2's.
e.g.:
$S \rightarrow AB \rightarrow 01B \rightarrow 012$
$S \rightarrow CD \rightarrow 0D \rightarrow 012$

An ambiguous Grammar: Ex3

$S \rightarrow SS \mid (S) \mid ()$ and $w = ()()()$



An ambiguous Grammar: Ex4

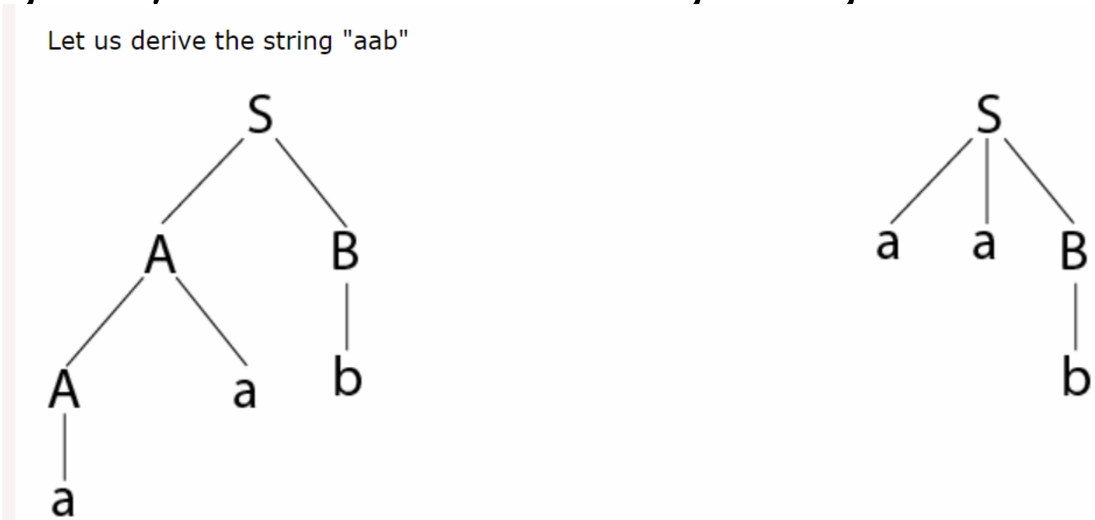
Consider a grammar G is given as follows:

$S \rightarrow AB \mid aaB$

$A \rightarrow a \mid Aa$

$B \rightarrow b$

If G is ambiguous, construct an unambiguous grammar equivalent to G.



→ The given grammar is ambiguous.

Unambiguous grammar will be:

$S \rightarrow AB$

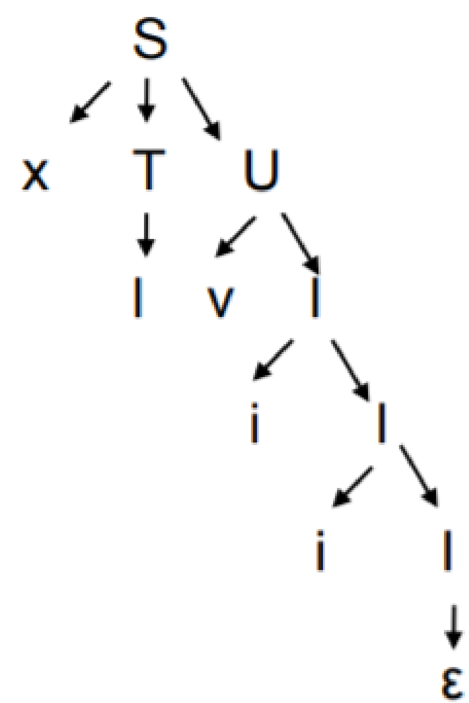
$A \rightarrow Aa \mid a$

$B \rightarrow b$

Consider the grammar with production;
with terminals {c, l, x, v, i}
c = 100, l = 50, x = 10, v = 5, i = 1

-Draw a parse tree for 47: "xlvii".

$$\begin{aligned}
S &\rightarrow \mathbf{x}TU \mid \mathbf{l}X \mid X \\
T &\rightarrow \mathbf{c} \mid \mathbf{l} \\
X &\rightarrow \mathbf{x}X \mid U \\
U &\rightarrow \mathbf{i}Y \mid \mathbf{v}I \mid I \\
Y &\rightarrow \mathbf{x} \mid \mathbf{v} \\
I &\rightarrow \mathbf{i}I \mid \epsilon
\end{aligned}$$



Is this grammar ambiguous? **NO**

L2= $\{a^n b^n \mid n>=1\}$

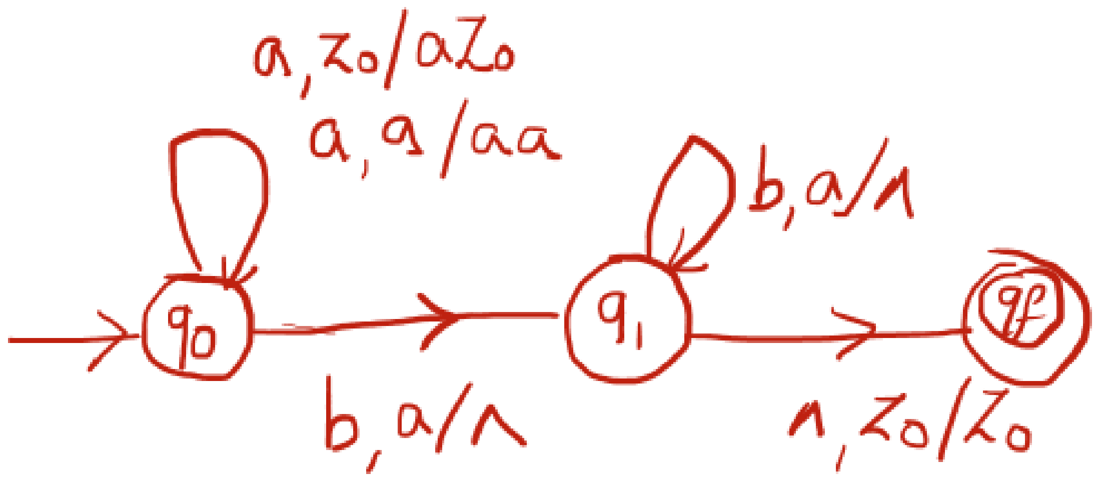
L3= $\{a^n b^n \mid n>=0\}$

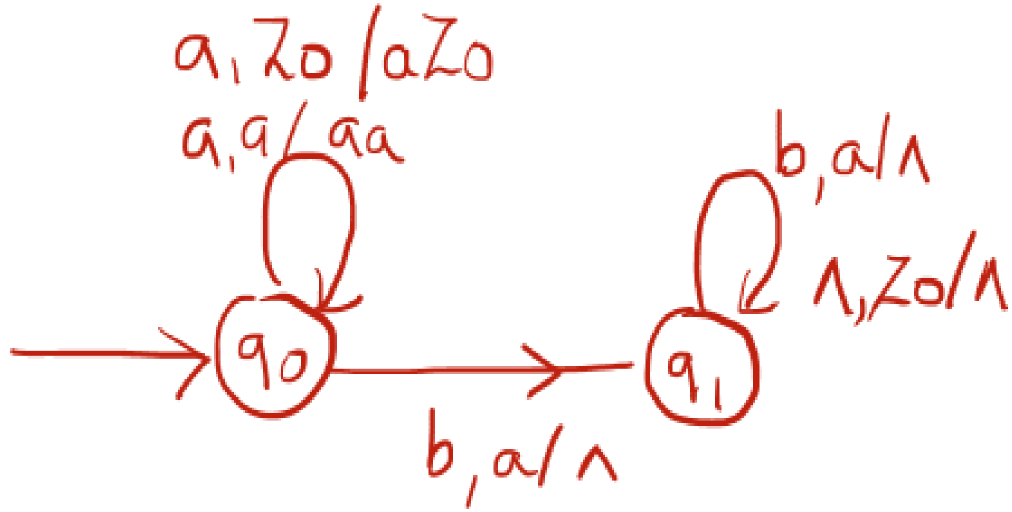
Note: G(L2) and G(L3) are Linear

G(L2) ≡ $S \rightarrow aSb \mid ab$

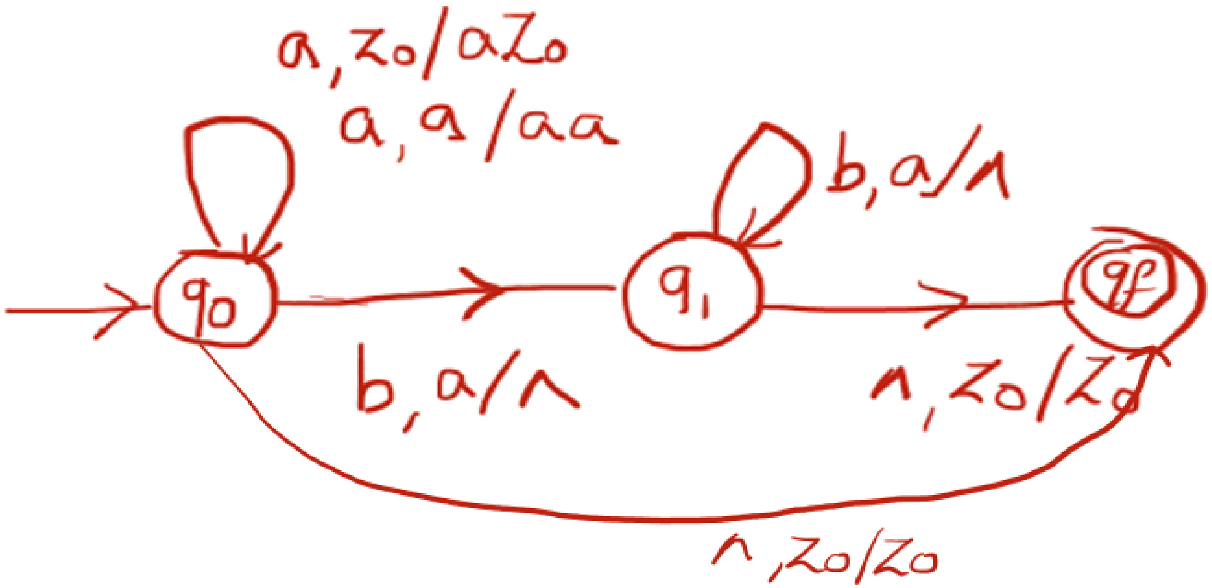
G(L3) ≡ $S \rightarrow aSb \mid \wedge$

$L2 = \{a^n b^n \mid n >= 1\}$





L3= $\{a^n b^n \mid n>=0\}$



Applications of CFG for:-

- defining **programming languages**
- **parsing** the program by constructing syntax tree
- translation of **programming languages**
- describing arithmetic expressions
- construction of **compilers**

Thank You!!