PES UNIVERSITY ONLINE

Automata Formal Languages & Logic

Preet Kanwal

Department of Computer Science & Engineering

Automata Formal Languages & Logic

Unit 2

Preet Kanwal

Department of Computer Science & Engineering

Regular Grammar

Automata Formal Languages and Logic

Unit 2 - Regular Grammar (outline)

- Formal Definition of a Grammar
- Terminology : Sentence , Sentential Form
- Examples on how to construct a Regular Grammar for a given L

Grammar

- **V - Variables** — expr, var, S, A, B…, w,x,y
- **T - Terminals** — a, b, c …, 0, 1, … id, num
- **P - Production Rules** — α or $\beta = (V \cup T)^*$; $\alpha \rightarrow \beta$
- **S - Start Symbol** — L(G) = {w | S ⇒ w, where w ∈ Σ^*}

Sentence (or a String)

A sentence is made up only of terminals.

Example:

For the Grammar S->aS | a |λ

Sentence (or string) is { a or λ or aa or aaaa)

Sentential Form

A sentential form is made up over set of terminals and non-terminals (VUT)*

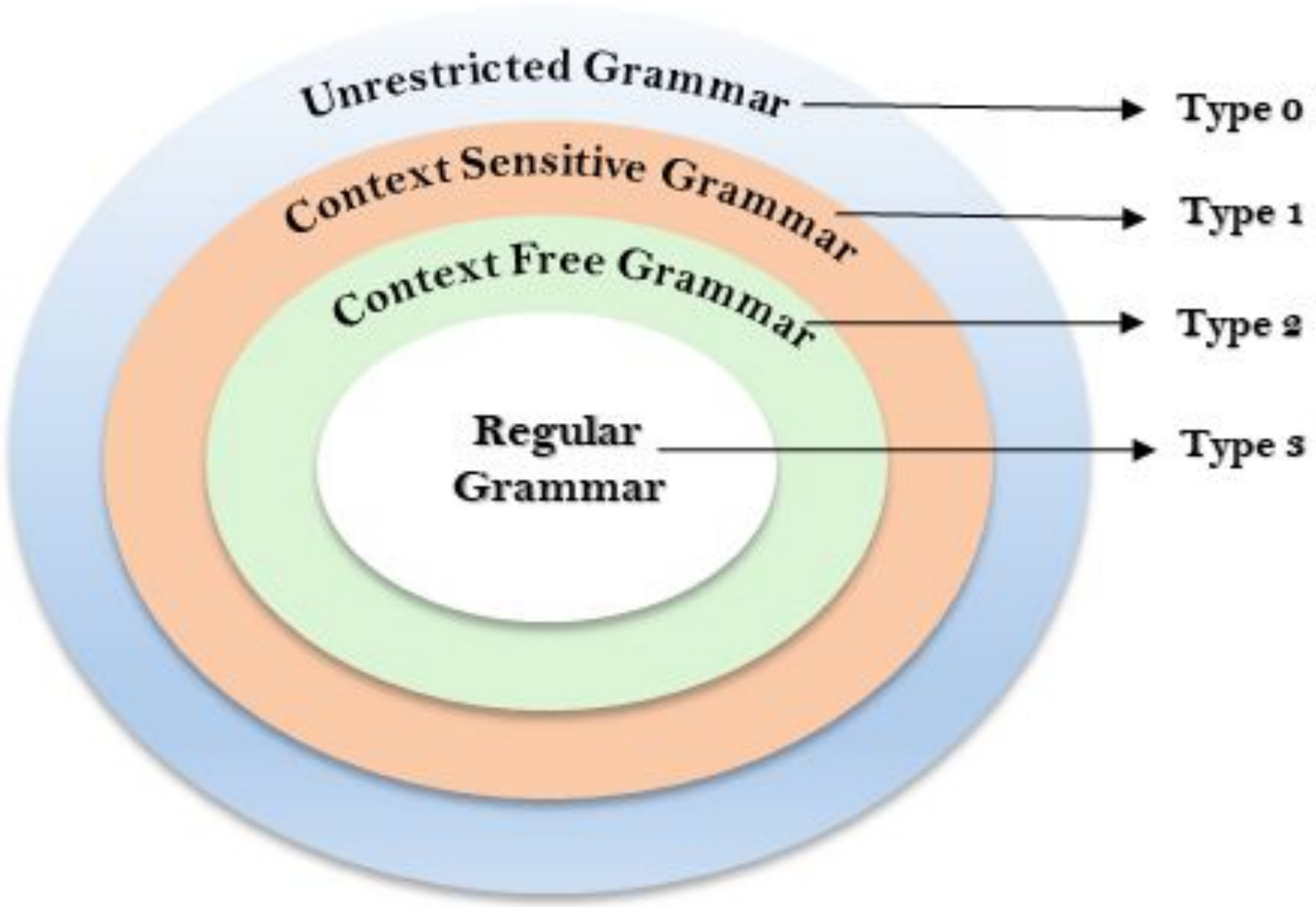
Example:

For the Grammar S->aS | a |λ

Sentential form is { a or λ or aS or aa or or aaS aaaa)

Automata Formal Languages and Logic

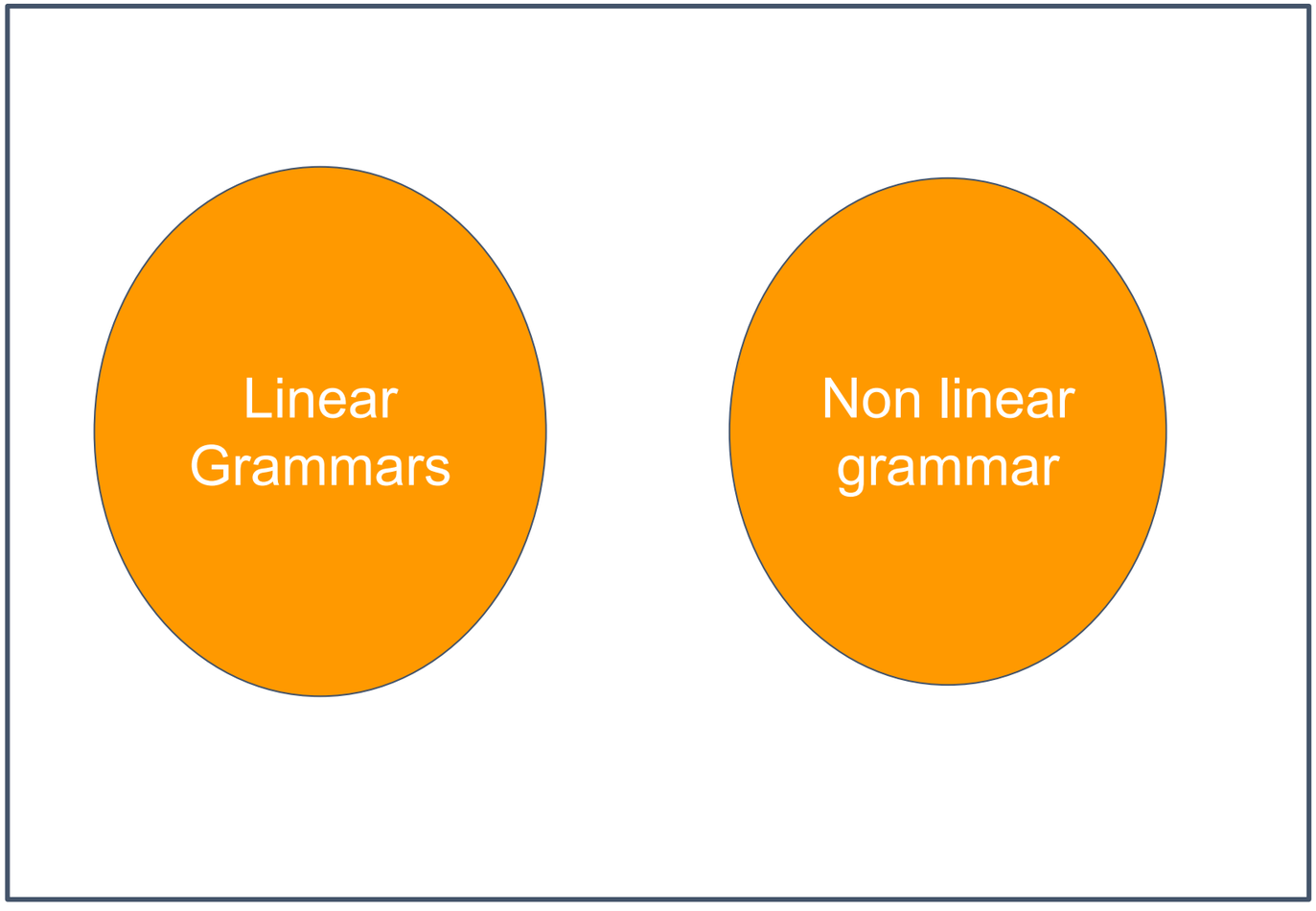
Unit 2 - Regular Grammar

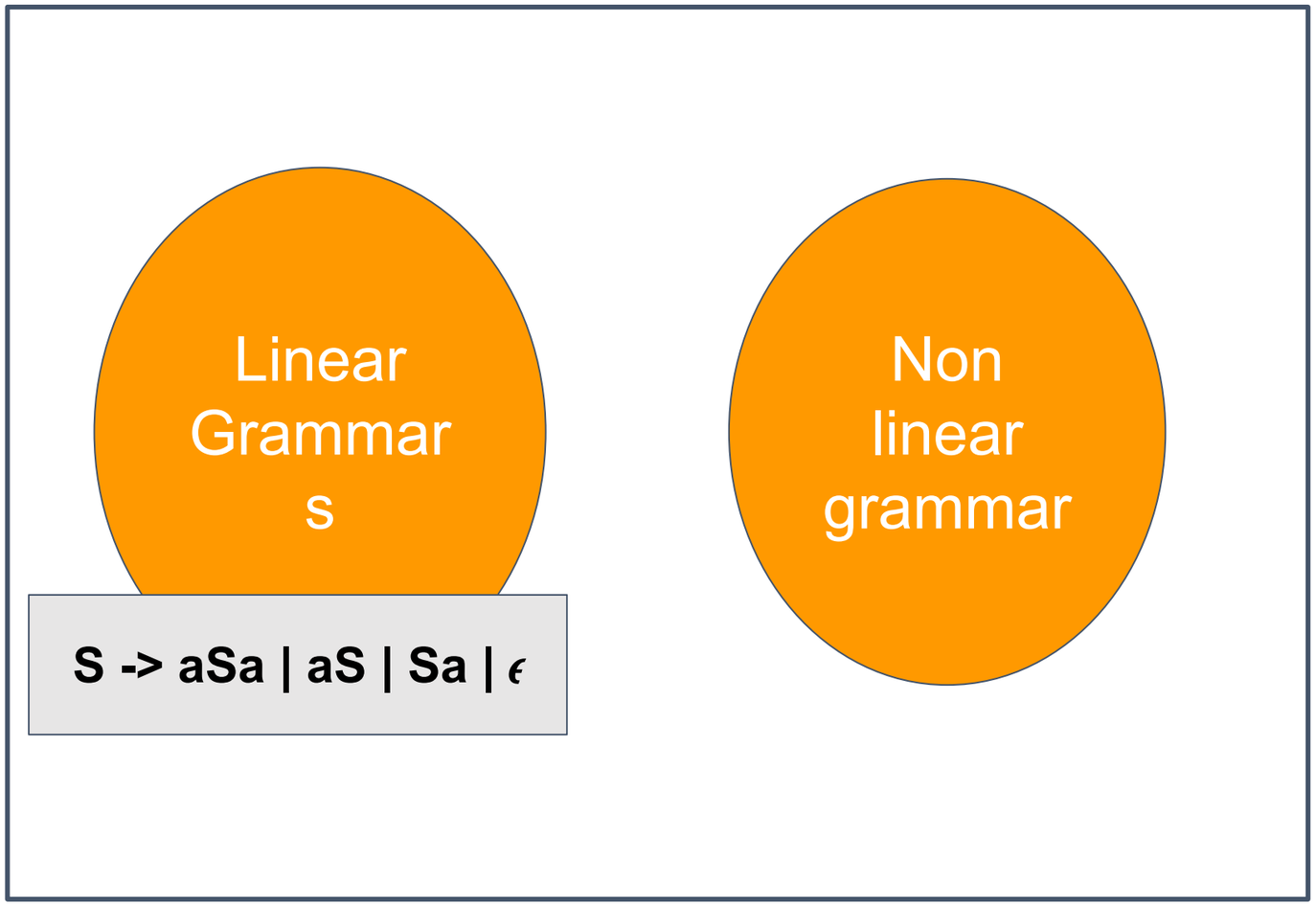


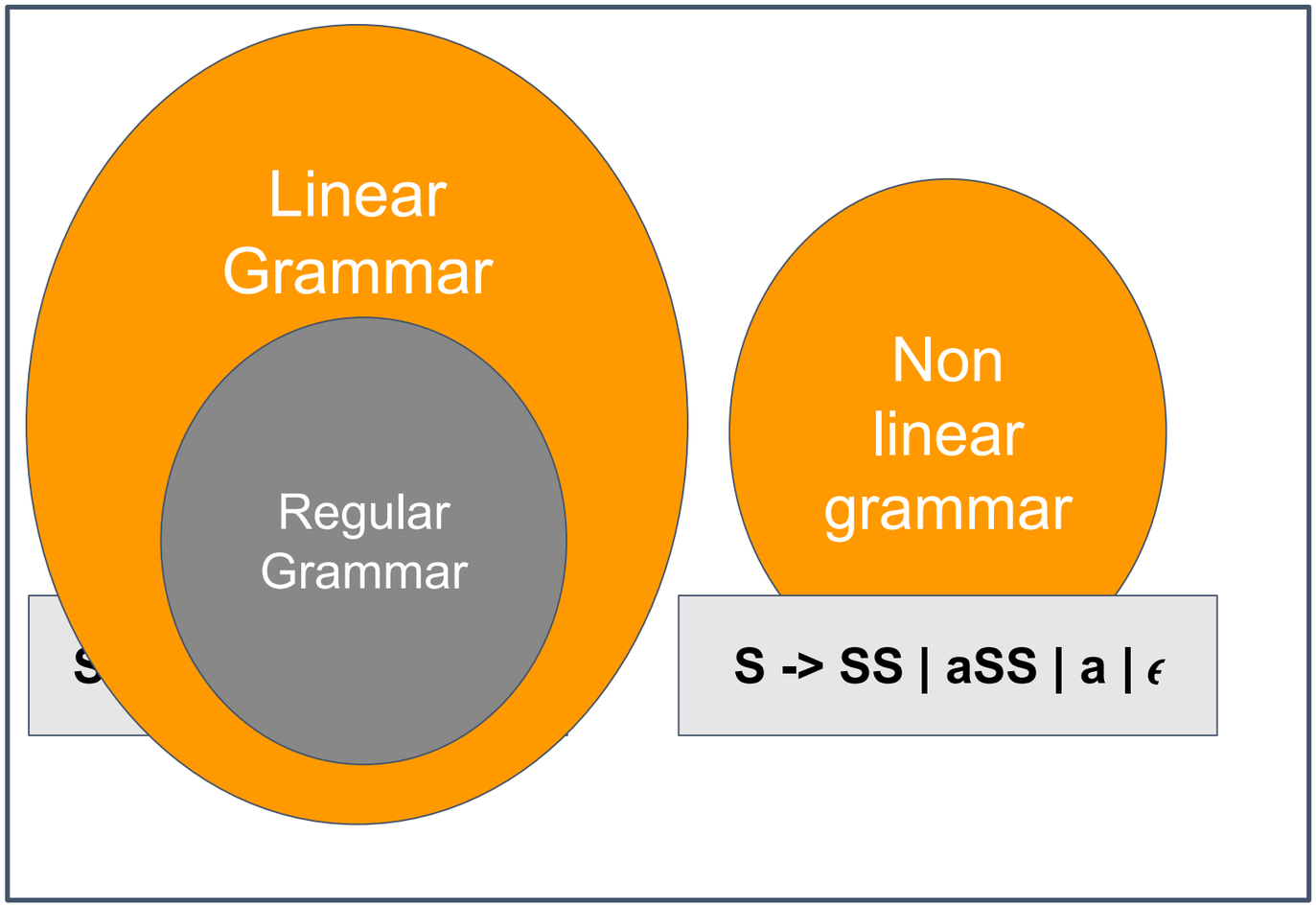
Chomsky Hierarchy

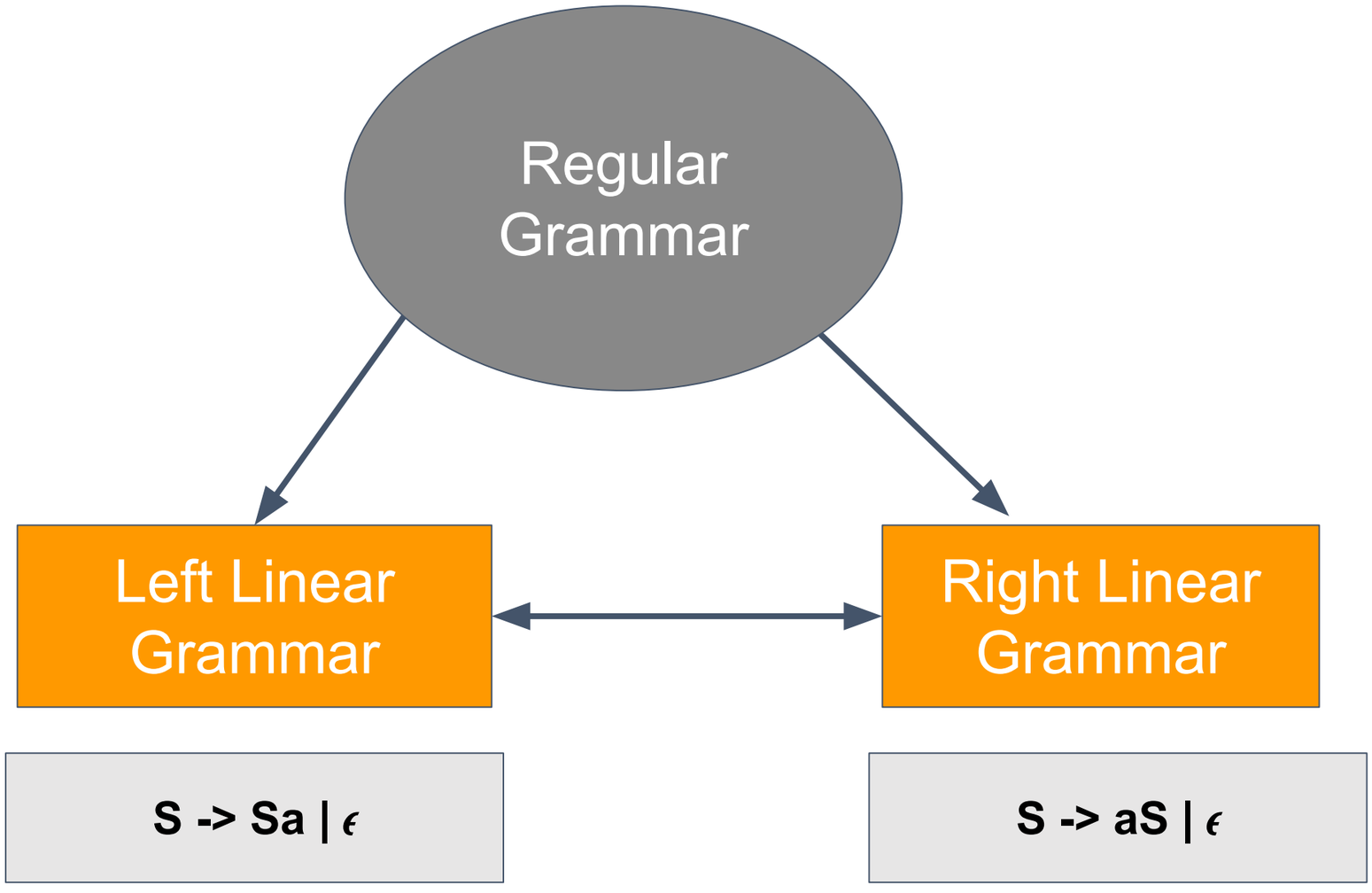
Noam Chomsky

https://www.javatpoint.com/automata-chomsky-hierarchy


Linear Grammars
Non linear grammar


Linear Grammars
Non linear grammar
S -> aSa | aS | Sa | ϵ


Linear
Grammar
Regular
Grammar
Non
linear
grammar
S -> SS | aSS | a | ϵ


Regular Grammar
Left Linear Grammar
Right Linear Grammar
S -> Sa | ϵ
S -> aS | ϵ

Construct a regular grammar for a given language description

Construct a regular grammar for the given language

Language	RE	RG
L = {a}	a	S -> a

Construct a regular grammar for the given language

Language	RE	RG
L = {a, b}	a+b	S -> a | b or S -> a S -> b

Construct a regular grammar for the given language

Language	RE	RG
L = {ab}	ab	S -> ab

Construct a regular grammar for the language $L=\{a^n|n>=0\}$

Language	RE	RG
L = {λ,a,aa,aaa,...}	a*	S -> aS|λ

Construct a regular grammar for the language $L=\{a^n|n>=1\}$

Language	RE	RG
L = {a,aa,aaa,...}	a^+	S -> aS | a

Construct a regular grammar for the regular expression (a+b)*

Language	RE	RG
L = {λ,a,b,abb,baaaab,.....} L={any number of ‘as and b’s}	(a+b)*	S -> aS | bS | λ

Construct a regular grammar for the regular expression $(a+b)^+$

Language	RE	RG
L = {λ,a,b,abb,baaaab,.....} L={any number of 'as and b's}	$(a+b)^+$	S -> aS | bS | a | b

Construct a regular grammar for the regular expression $(ab)^*$

Language	RE	RG
L = {λ,a,b,ab,abababab,.....} L={any number of ab's}	$(ab)^*$	S -> abS | λ

Construct a regular grammar for the language $L=\{a^m b^n |n,m>=0\}$

Language	RE	RG
L={λ,a,b,bb,abb,.....} L={any number of a's followed by any number of b's}	a*b*	S -> A|B A:any number of a's B:any number of b's A->aA|λ B->bB|λ

Construct a regular grammar for the given language with even number of a's. $L=\{a^{2n} \mid n>=0\}$

Language	RE	RG
L={λ, aa, aaaa, aaaaaa	(aa)*	S -> aaS | λ

Construct a regular grammar for the given language with odd number of a's. $L=\{a^{2n+1} \mid n>=0\}$

Language	RE	RG
L={a, aaa, aaaaa, aaaaaaa	(aa)*a	S -> aaS | a

Construct a regular grammar for the given language with number of a's as multiples of 4. $L=\{a^{4n} \mid n>=0\}$

Language	RE	RG
L={λ, aaaa, aaaaaaaa, aaaaaaaaaaaa, ….	(aaaa)*	S -> aaaaS | λ

Convert finite automata to regular grammar for the language to accept at least one ‘a’ over the alphabet Σ={a,b}

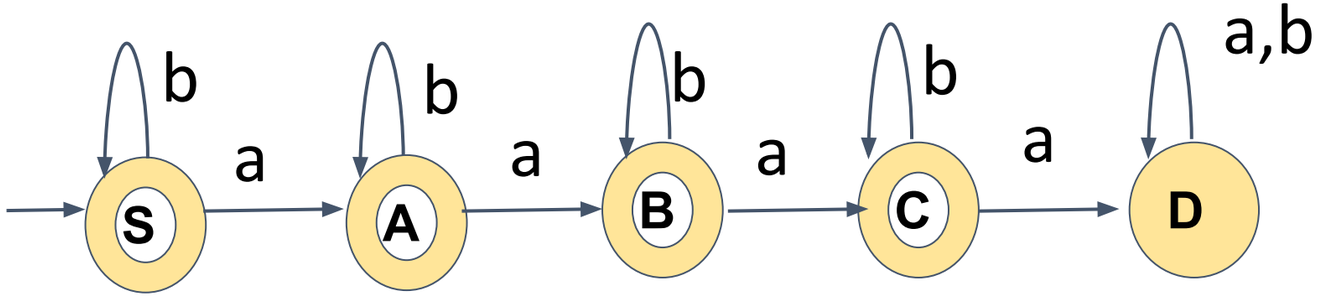
Language	RE	RG
L={a, bb... a, ba any # of a’s & b’s	b* a (a+b)*	S -> bS | aA A -> aA | bA | λ


Regular Grammar
Finite Automata

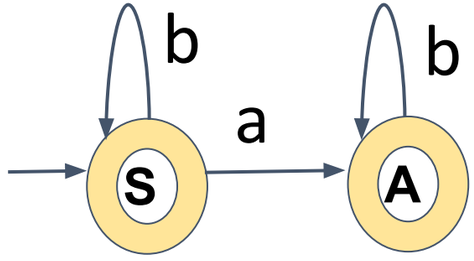
Let’s look at examples to convert :

1. Finite Automata to Regular Grammar
2. Regular Grammar to Finite Automata

1.Convert finite automata to regular grammar for the language to accept at most 3 ‘a’s over the alphabet Σ={a,b}.



Let’s look at each transition and perform the conversion

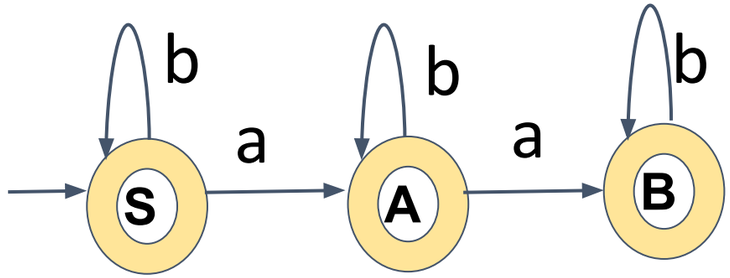


$S \rightarrow bS \mid aA \mid \lambda$

$A \rightarrow bA \mid \lambda$

λ on RHS indicates S and A are final states

Let’s look at each transition and perform the conversion



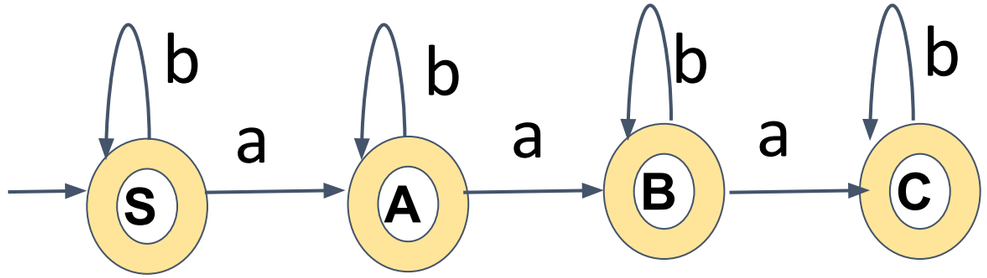
$S \rightarrow bS \mid aA \mid \lambda$

$A \rightarrow bA \mid aB \mid \lambda$

$B \rightarrow bB \mid aC \mid \lambda$

λ on RHS indicates S,A and B are final states

Let’s look at each transition and perform the conversion



$S \rightarrow bS \mid aA \mid \lambda$

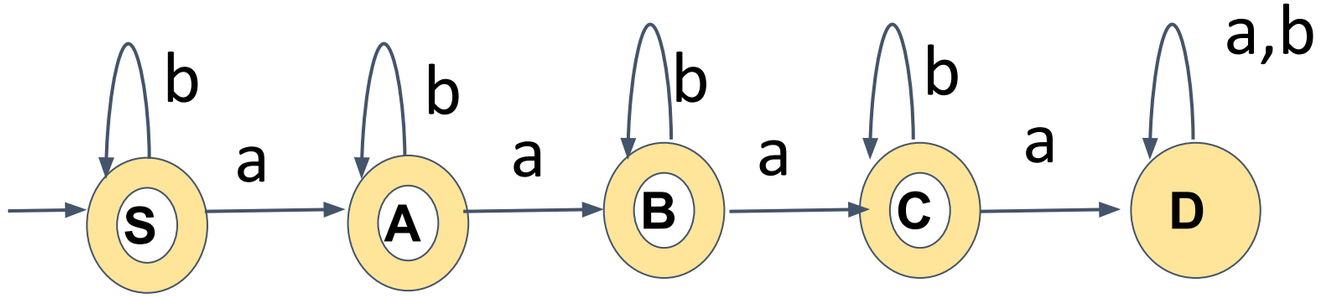
$A \rightarrow bA \mid aB \mid \lambda$

$B \rightarrow bB \mid aC \mid \lambda$

$C \rightarrow bC \mid aD \mid \lambda$

Let's look at each transition and perform the conversion

Since D is a dead state, we will not encode C-> aD and D -> aD | bD as D is a useless Non-terminal(that means it never terminates)



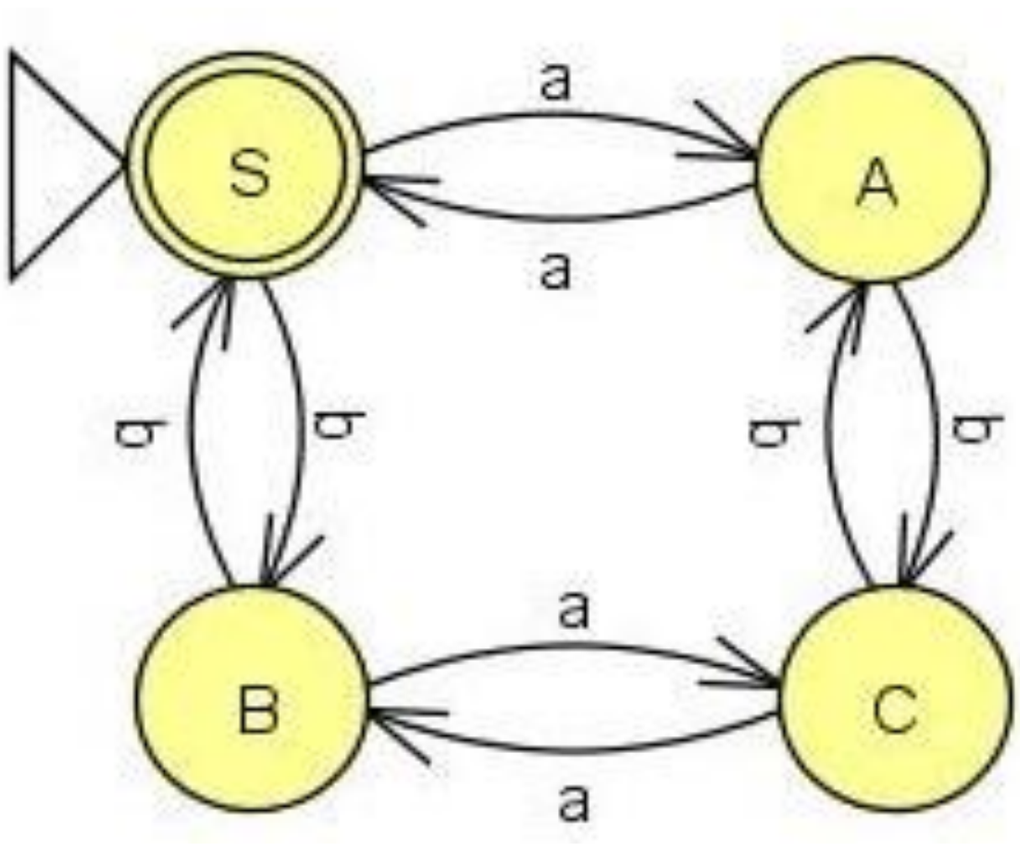
$S \rightarrow bS \mid aA \mid \lambda$

$A \rightarrow bA \mid aB \mid \lambda$

$B \rightarrow bB \mid aC \mid \lambda$

$C \rightarrow bC \mid \lambda$

2.Convert a given finite automata accepting $L=\{n_a(w) \bmod 2=0$ and $n_b(w) \bmod 2=0\}$ to regular grammar.



2.Convert a given finite automata accepting L={n_a(w)mod 2=0 and n_b(w)mod 2=0 } to regular grammar.

Start state of automata will be the start symbol of the grammar.

We start with S,S on seeing terminal a it moves to state A(S$\rightarrow$ aA) and on seeing terminal b it moves to state B (S$\rightarrow$ bB).

Since S is also the final state,we introduce the production S$\rightarrow$ λ.

Grammar is

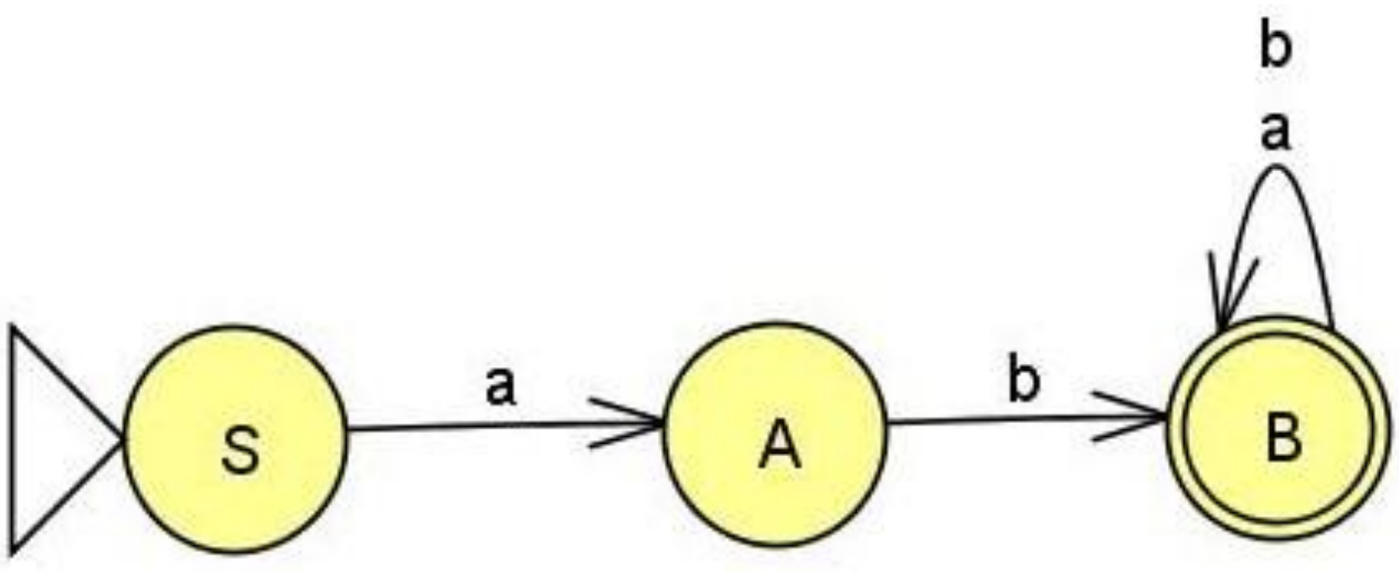
S$\rightarrow$ aA|bB| λ

A$\rightarrow$ aS|bC

B$\rightarrow$ aC|bS

C$\rightarrow$ aB|bA

3. Converting a given finite automata accepting L={abw,w∈{a,b}*} to regular grammar.



3. Converting a given finite automata accepting L={abw,w∈{a,b}*} to regular grammar.

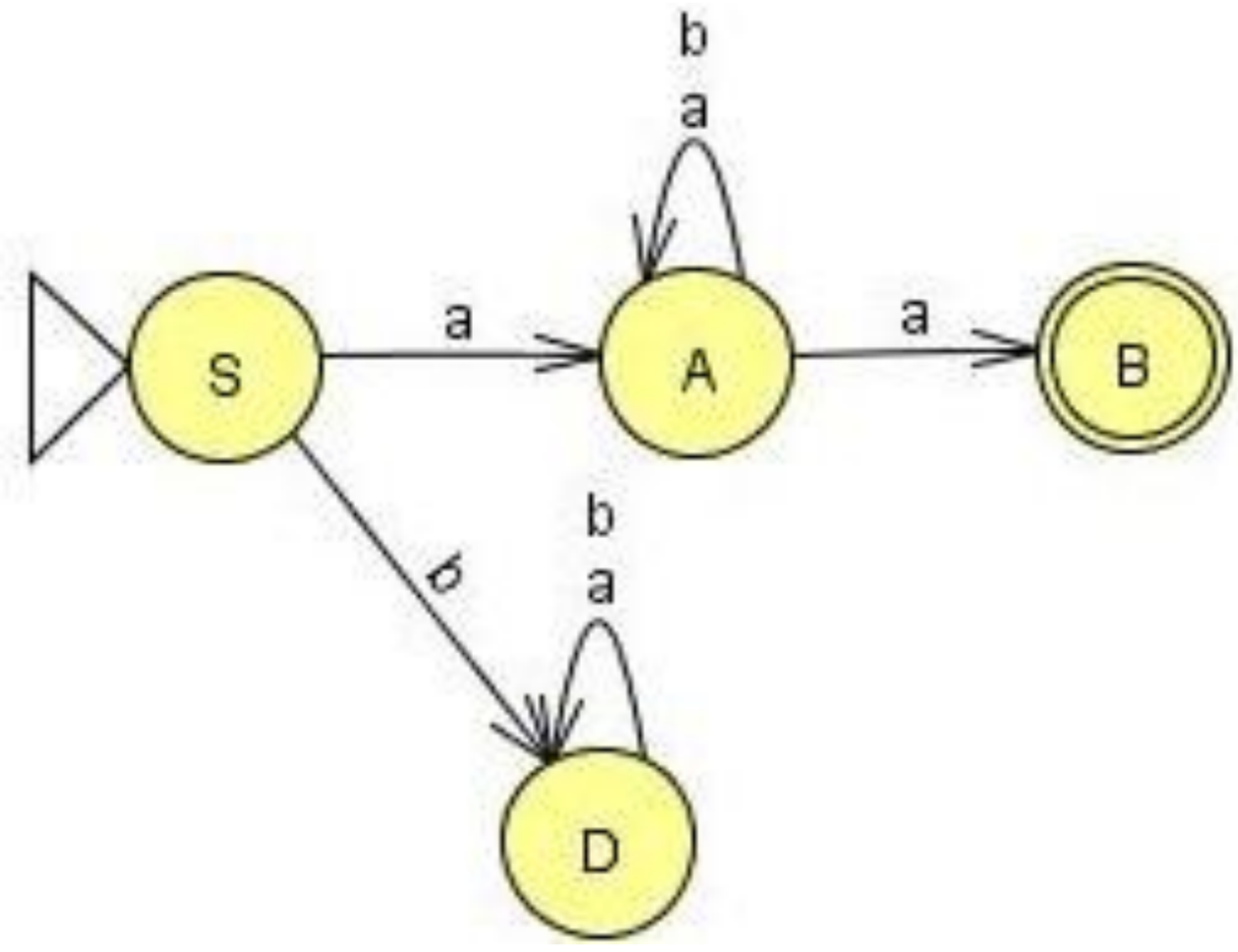
Grammar is,

$S \longrightarrow aA$

$A \longrightarrow bB$

$B \longrightarrow aB|bB|\lambda$

4. Converting given finite automata accepting L={awa,w∈{a,b}*} to regular grammar.



4. Converting given finite automata accepting L={awa,w∈{a,b}*} to regular grammar.

So the grammar is,

$S \longrightarrow aA$

$A \longrightarrow aA|bA|aB$

$B \longrightarrow \lambda$

Converting Regular grammar to finite automata

Example 1 :

$A \rightarrow aB|bA|b$

$B \rightarrow aC|bB$

$C \rightarrow aA|bC|a$

Variables->state

terminal symbols ->symbols of finite automata

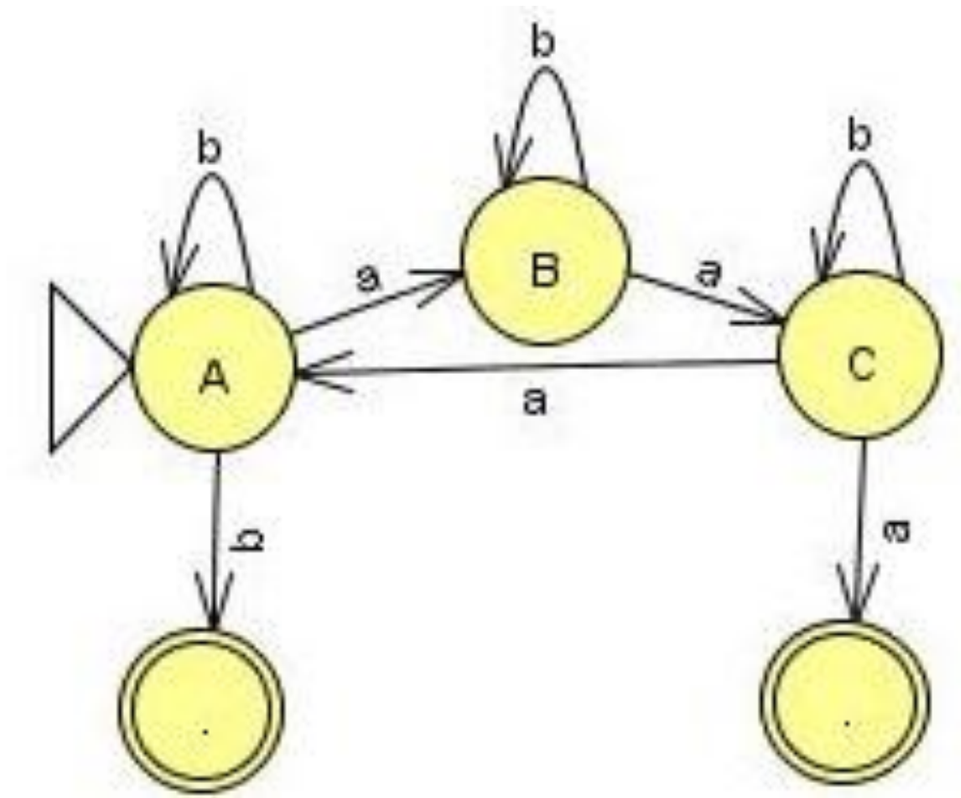
Example 1 :

$A \rightarrow aB|bA|b$

$B \rightarrow aC|bB$

$C \rightarrow aA|bC|a$

- Transition $C \rightarrow aA$ to $C \rightarrow a$ when $A \rightarrow \lambda$ indicates A is a final state.



Example 2 :

S→01A

A→10B

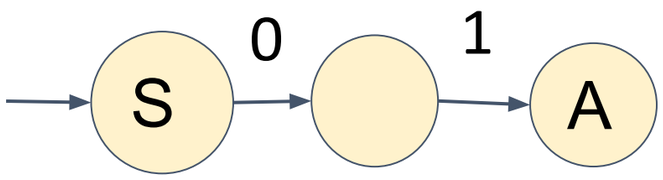
B→0A|11

Example 2 :

$S \rightarrow 01A$

$A \rightarrow 10B$

$B \rightarrow 0A|11$

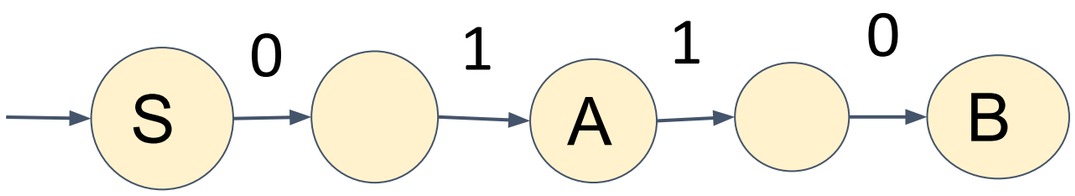


Example 2 :

$S \rightarrow 01A$

$A \rightarrow 10B$

$B \rightarrow 0A|11$



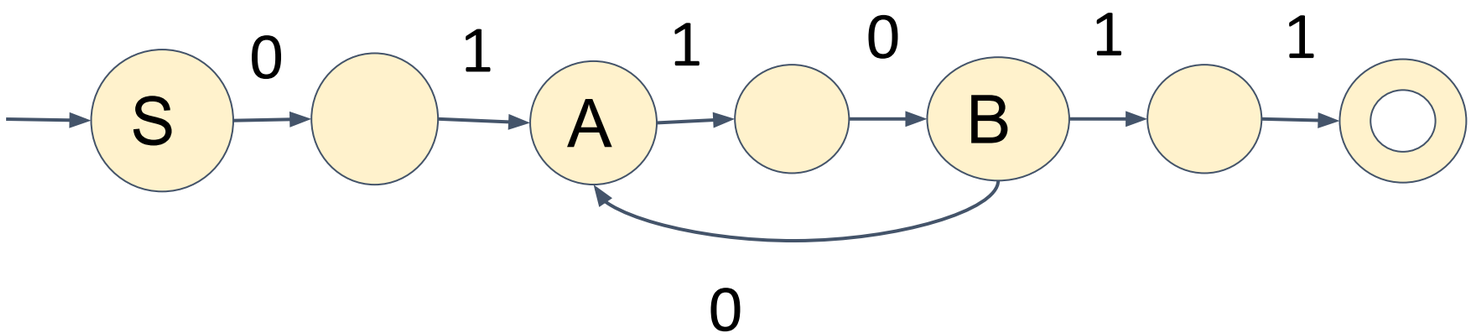
Example 2 :

S⟶01A

A⟶10B

B⟶0A|11

B⟶11 indicates on state B on consuming the input 11 we reach final state



Example 3 :

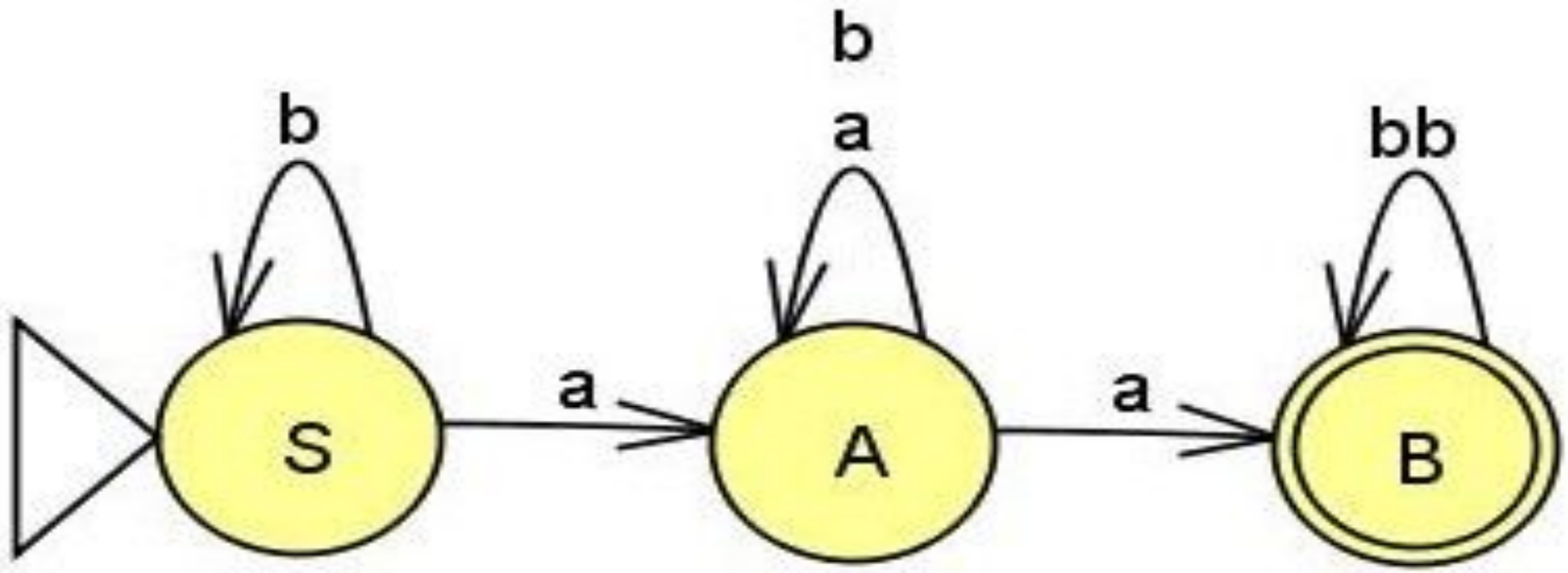
$S \rightarrow bS|aA$

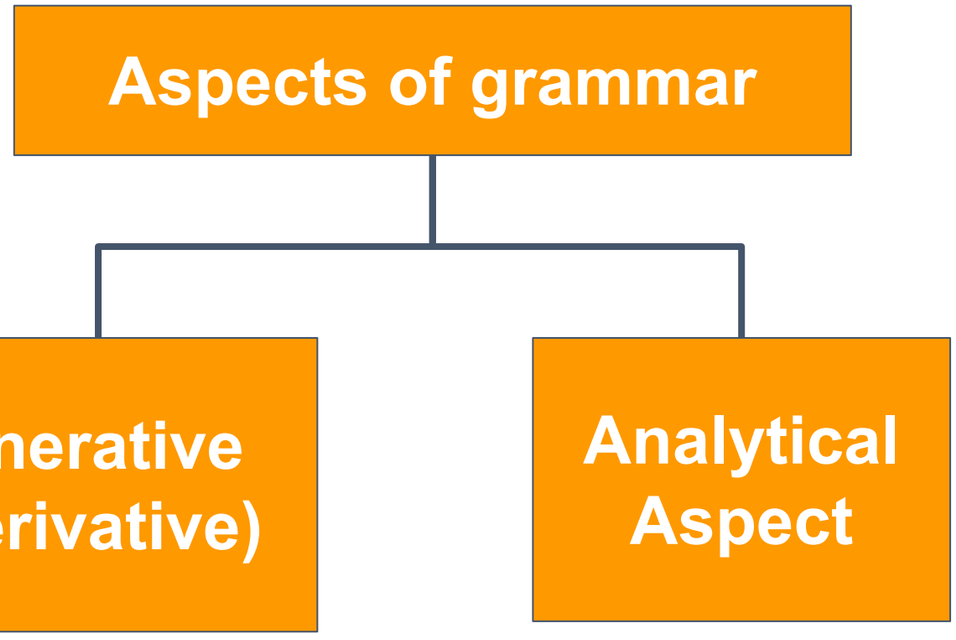
$A \rightarrow aA|bA$

$A \rightarrow aB$

$B \rightarrow bbB$

$B \rightarrow \lambda$




Aspects of grammar
Generative (Derivative)
Analytical Aspect
Generate All Strings w ϵ L(G)
Check whether w ϵ L(G) ??

Parse Tree

What is a Tree?

Leaves

Branches

Root

Technically we represent a Tree upside down

Root-

Branches

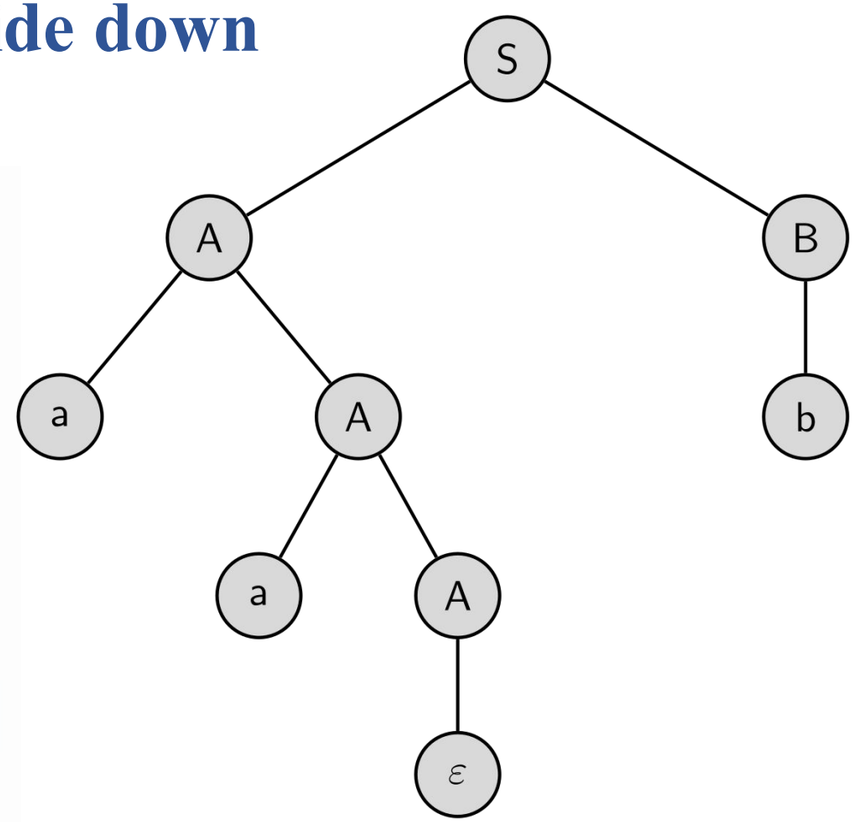
Leaves

Technically we represent a Tree upside down

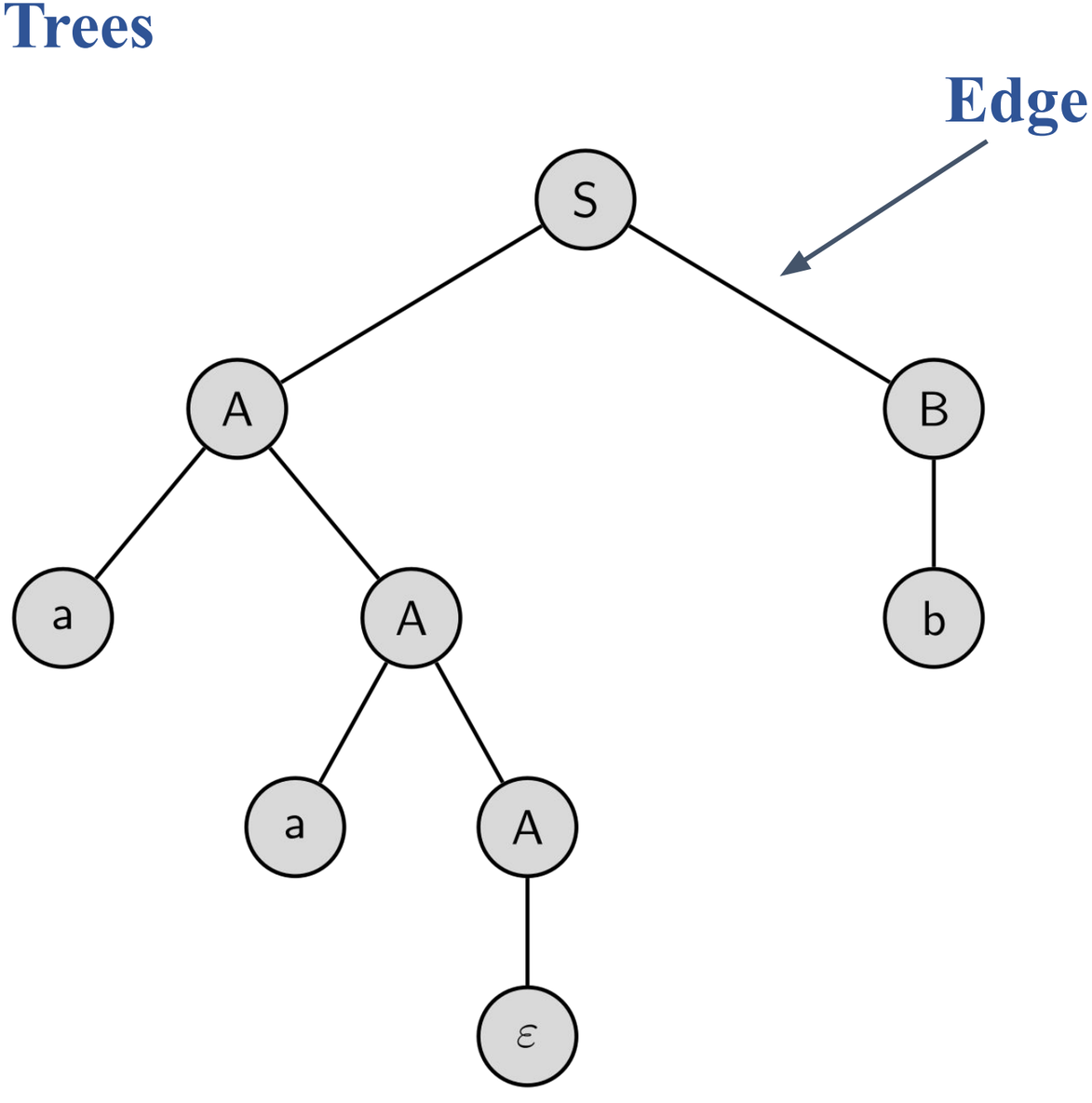
Root

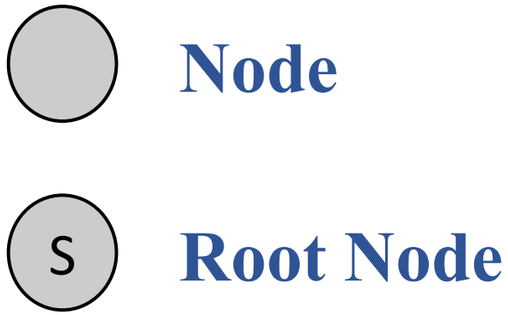
Branches

Leaves

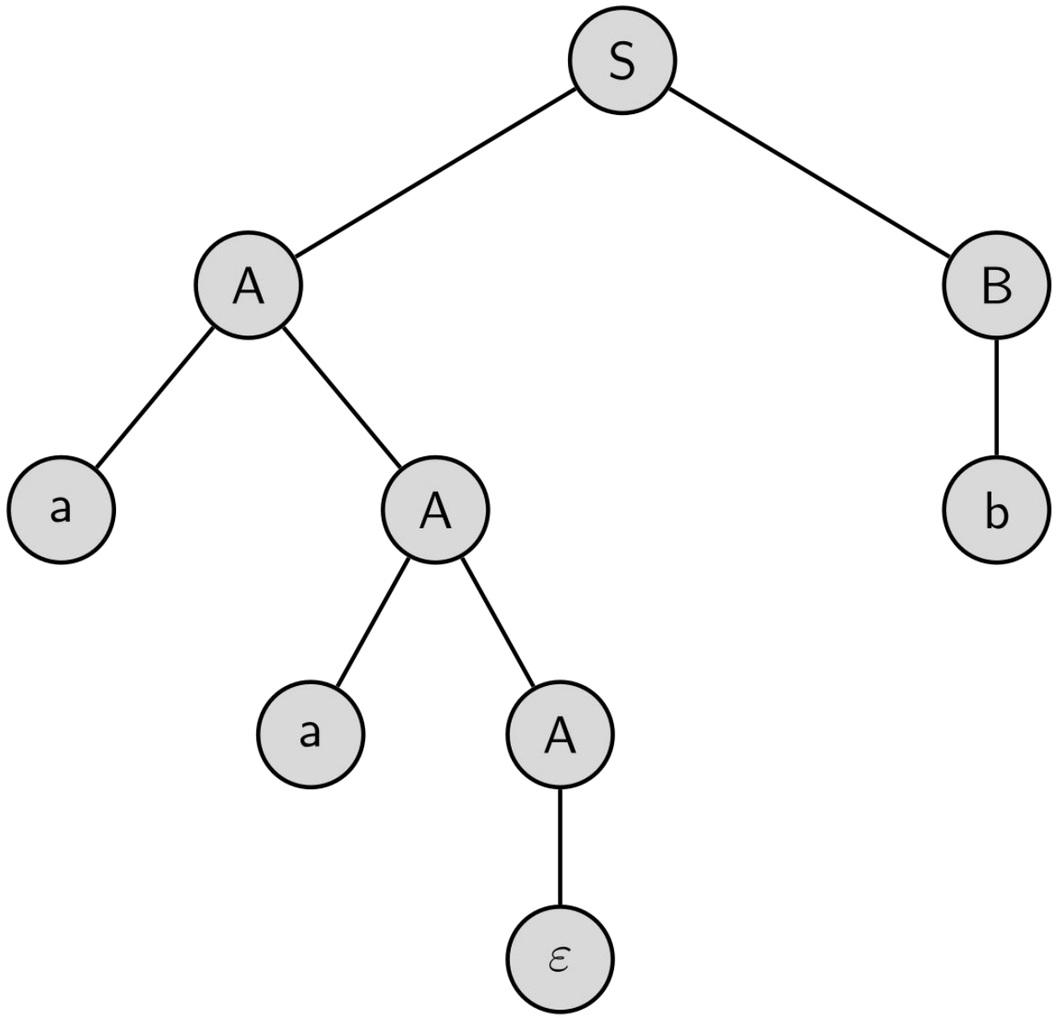


Trees





Parse Tree / Derivation Tree



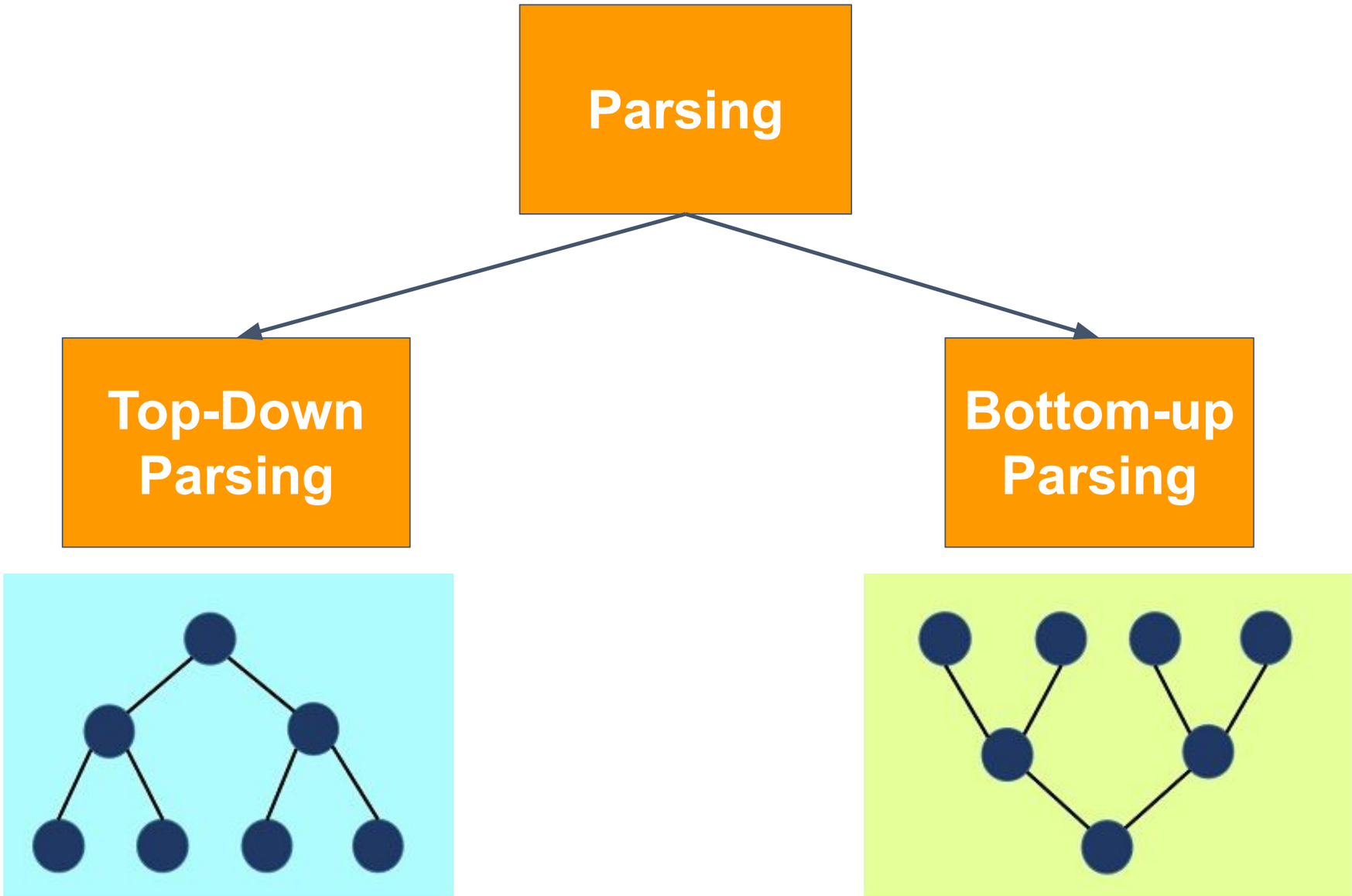
S **Root Node (Start Symbol)**

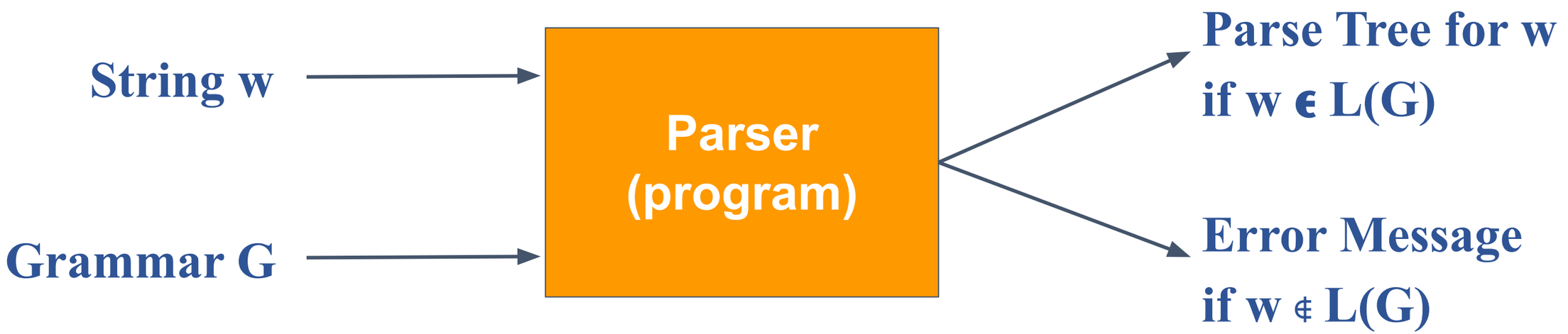
V **Non-Terminals - Interior Nodes**

T **Terminal Symbols - Leaf nodes**

Yield of Parse Tree - aab

Parsing is the process of determining whether a String w ∈ L(G)??


Parsing
Top-Down Parsing
Bottom-up Parsing

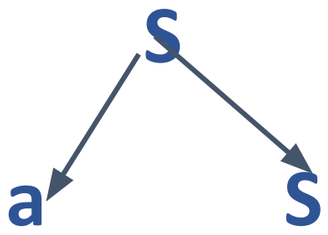

String w
Grammar G
Parser
(program)
Parse Tree for w
if w ϵ L(G)
Error Message
if w ∉ L(G)

Let us look at few examples and generate parse tree for a given String w and Grammar G.

Parse Tree

S Root node

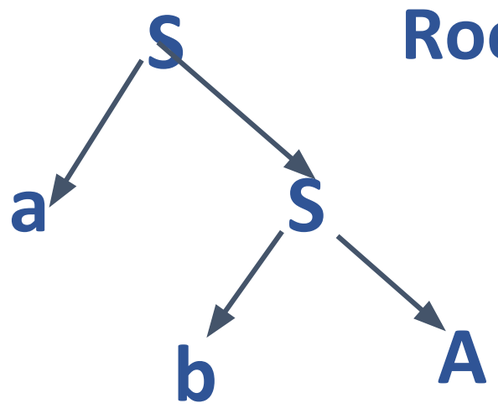
Parse Tree



Root node

$S \rightarrow aS$

Parse Tree

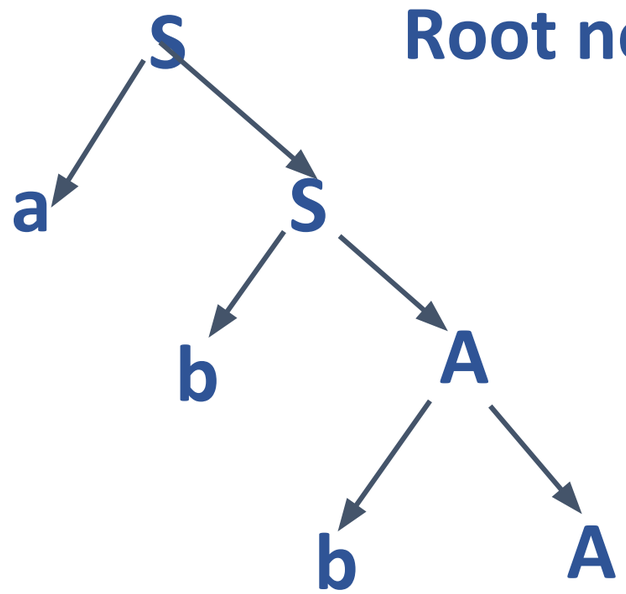


Root node

$S \rightarrow aS$

$S \rightarrow bA$

Parse Tree



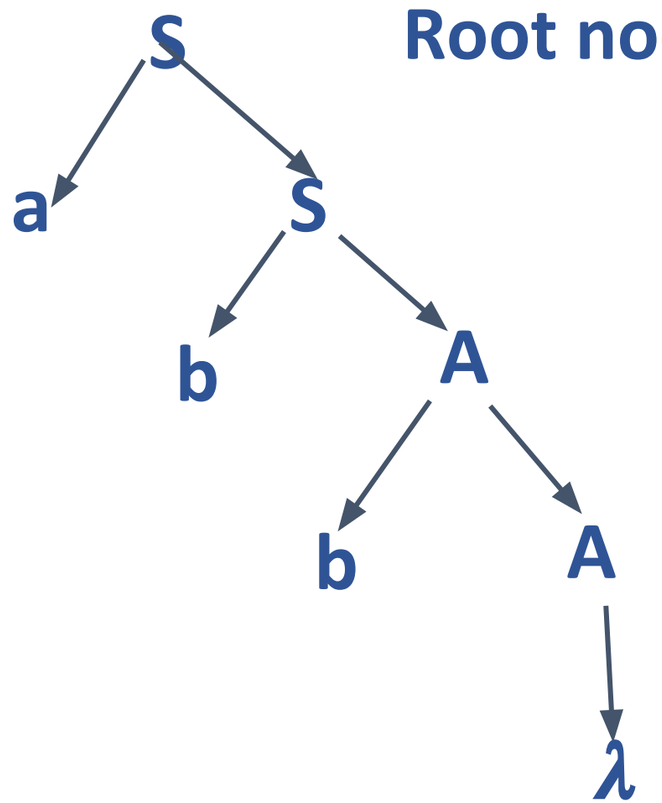
Root node

S⟶aS

S⟶bA

A⟶bA

Parse Tree



Root node

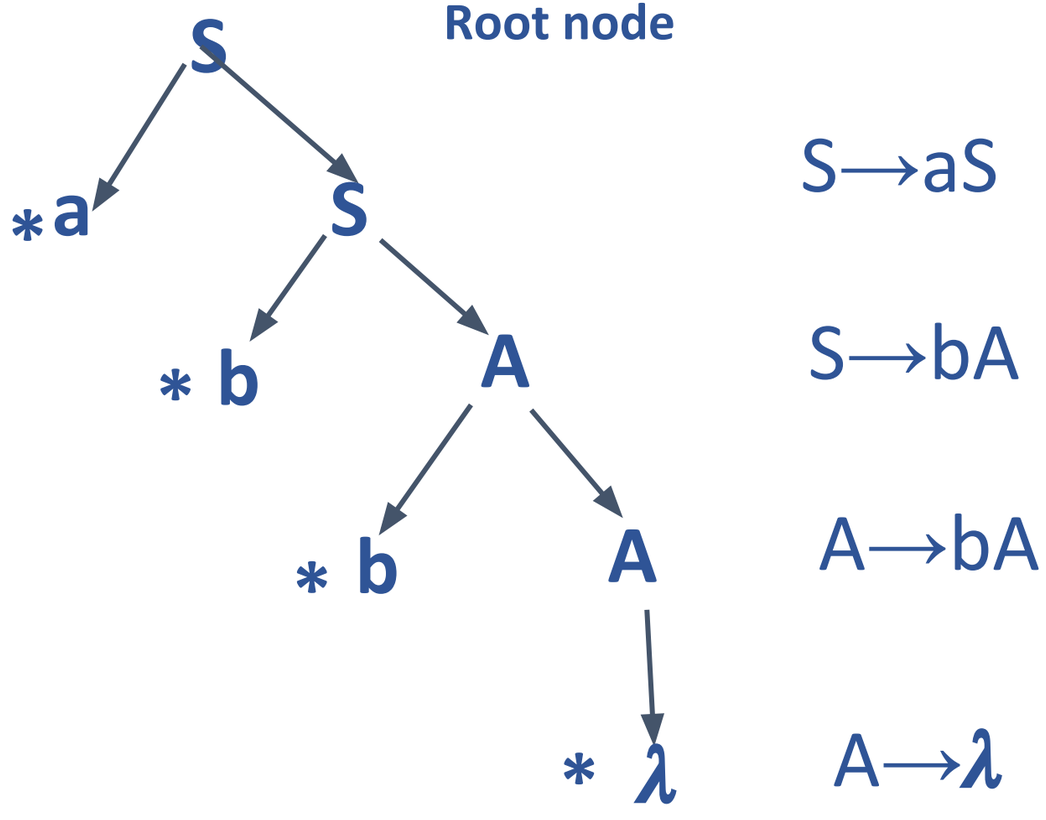
S⟶aS

S⟶bA

A⟶bA

A⟶λ

Parse Tree



Root node

S⟶aS

S⟶bA

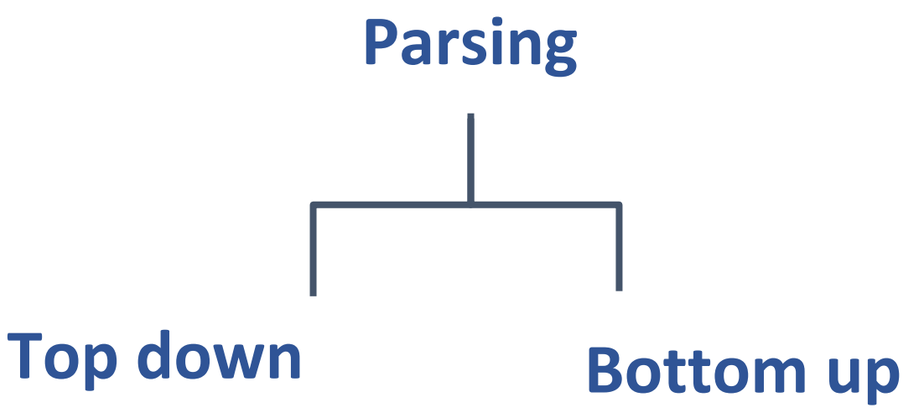
A⟶bA

A⟶λ

Leaf Node:*

Yield of the tree:abb

2. Analytical aspect



2. Analytical aspect

$S \rightarrow aS$

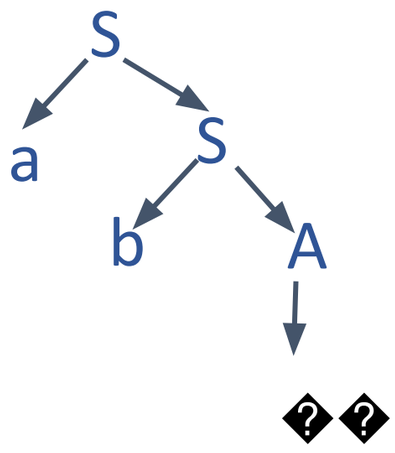
$S \rightarrow bA$

$A \rightarrow bA$

$A \rightarrow \lambda$

Parsing

Top down



Bottom up

2. Analytical aspect

$S \rightarrow aS$

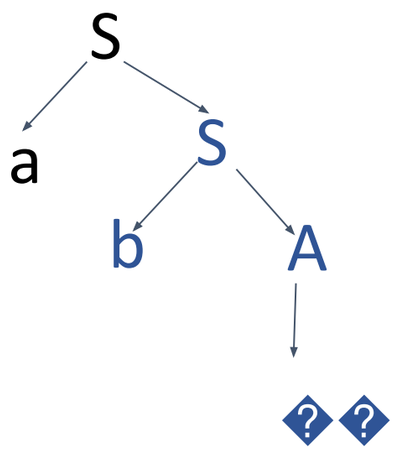
$S \rightarrow bA$

$A \rightarrow bA$

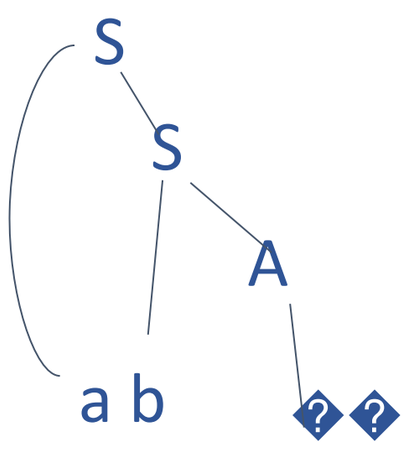
$A \rightarrow \lambda$

Parsing

Top down



Bottom up



All about Left Linear Grammar

Automata Formal Languages and Logic

Unit 2 - Constructing a Left Linear Grammar

Right Linear

$A \rightarrow aB$

$B \rightarrow aB|bB|\lambda$

L={starting with ‘a’}

Left Linear

??

https://www.youtube.com/watch?v=8a2jBm5Syjs

Right Linear

$A \rightarrow aB$

$B \rightarrow aB \mid bB \mid \lambda$

L={starting with ‘a’}

Reverse the Symbols on RHS

Left Linear

$A \rightarrow Ba$

$B \rightarrow Ba \mid Bb \mid \lambda$

L=?

https://www.youtube.com/watch?v=8a2jBm5Syjs

Unit 2 - Constructing a Left Linear Grammar

Right Linear

$A \rightarrow aB$

$B \rightarrow aB \mid bB \mid \lambda$

L={starting with ‘a’}

Reverse the Symbols on RHS

Left Linear

$A \rightarrow Ba$

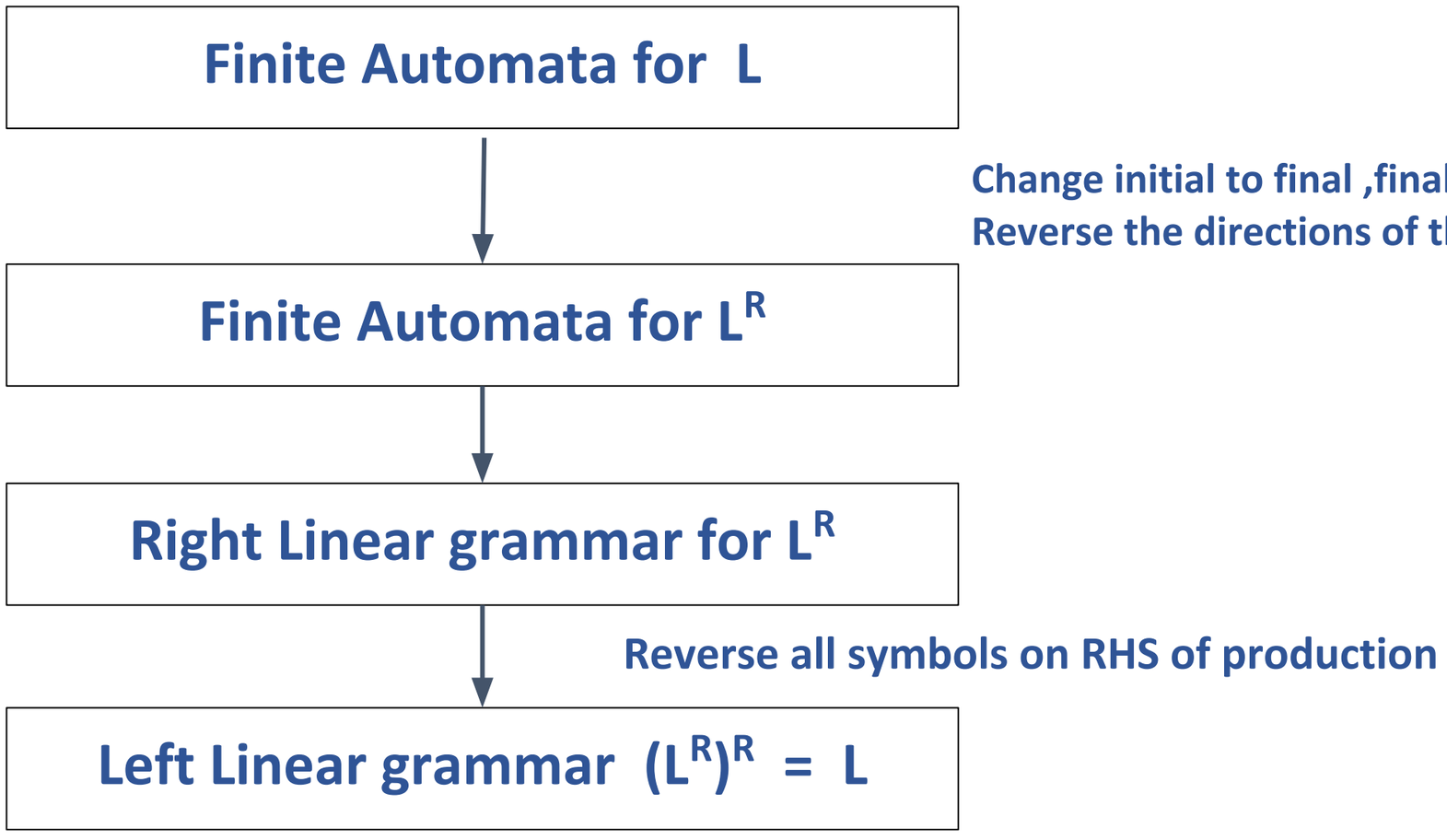
$B \rightarrow Ba \mid Bb \mid \lambda$

L^R**={ending with ‘a’}**

When we reverse the RHS in every production in the right linear grammar for “ L” we get a left linear grammar which will represent L^R.

https://www.youtube.com/watch?v=8a2jBm5Syjs

Converting Finite Automata to Left linear grammar

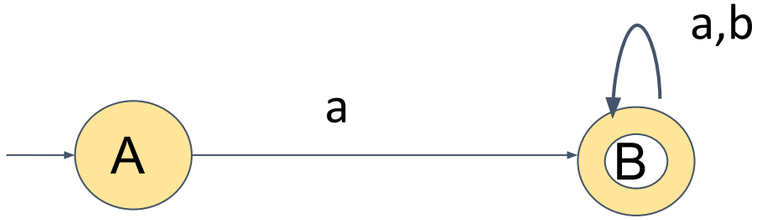


https://www.youtube.com/watch?v=8a2jBm5Syjs

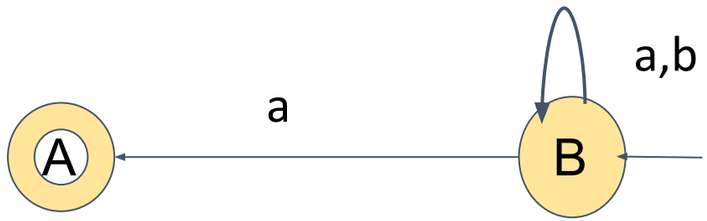
Automata Formal Languages and Logic

Unit 2 - Constructing a Left Linear Grammar

FA for L = {aw, w ϵ {a,b}*}



FA for L^R = {wa, w ϵ {a,b}*}



RLG for L^R

Right Linear Grammar

$B \rightarrow aB \mid bB \mid aA$

$A \rightarrow \lambda$

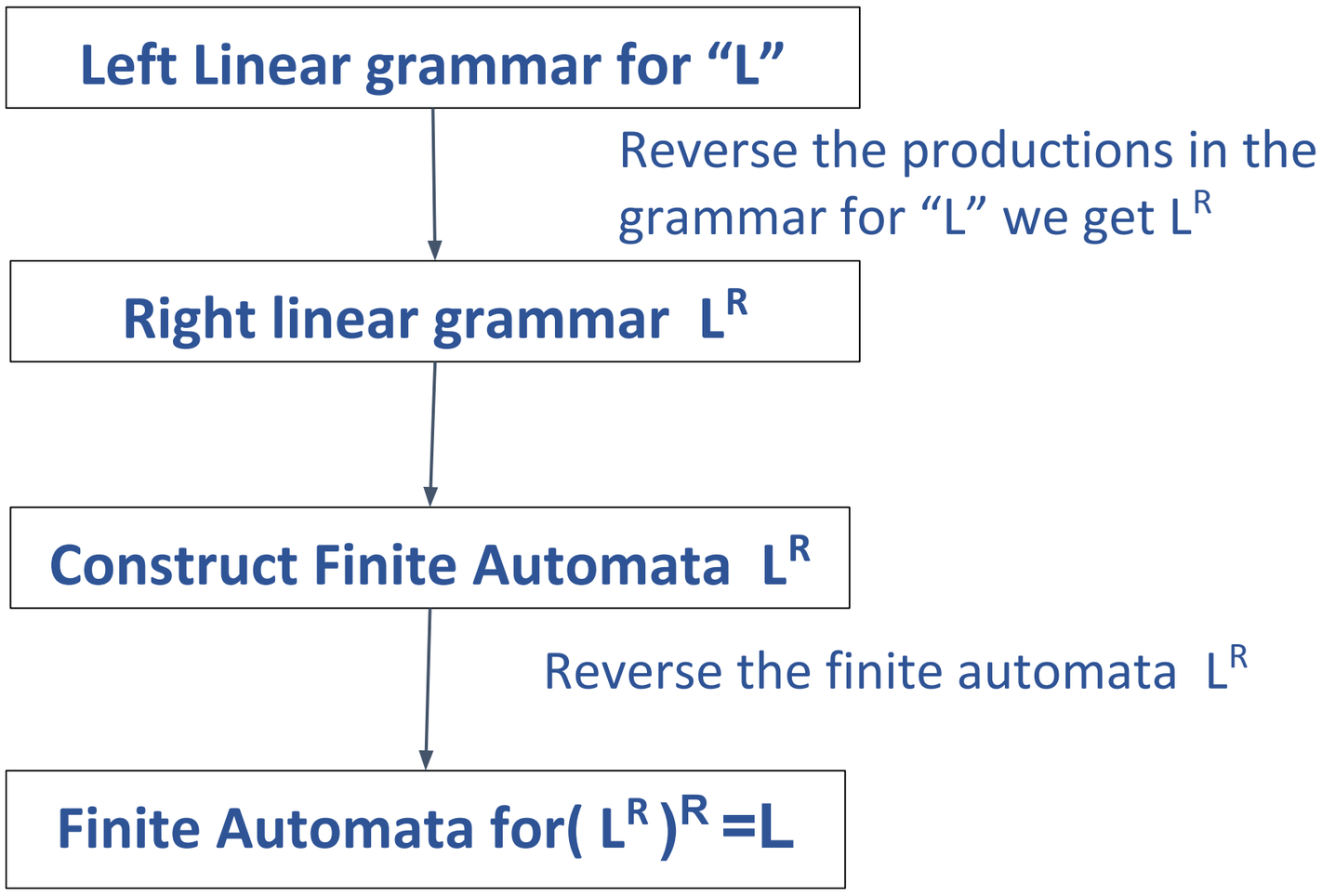
LLG for $(L^R)^R = L$

Left Linear Grammar

$B \rightarrow Ba \mid Bb \mid Aa$

$A \rightarrow \lambda$

Converting Left linear grammar to Finite automata


Left Linear grammar for “L”
Reverse the productions in the grammar for “L” we get L^R
Right linear grammar L^R
Construct Finite Automata L^R
Reverse the finite automata L^R
Finite Automata for(L^R)^R =L

Automata Formal Languages and Logic

Unit 2 - Constructing a Left Linear Grammar

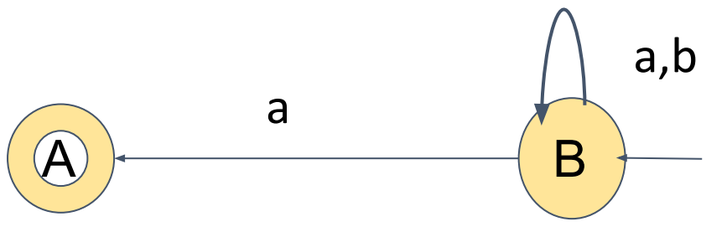
LLG for L

Left Linear Grammar
$B \rightarrow Ba \mid Bb \mid Aa$
$A \rightarrow \lambda$

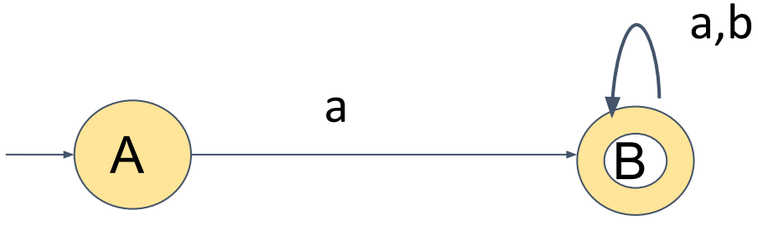
RLG for L^R

Right Linear Grammar
$B \rightarrow aB \mid bB \mid aA$
$A \rightarrow \lambda$

FA for L^R = {wa, w ϵ {a,b}*}



FA for $(L^R)^R = L$ = {aw, w ϵ {a,b}*}



Which one is easier LLG or RLG??

Automata Formal Languages and Logic

Unit 2 - Which one is easier ,Left or Right Linear Grammar?

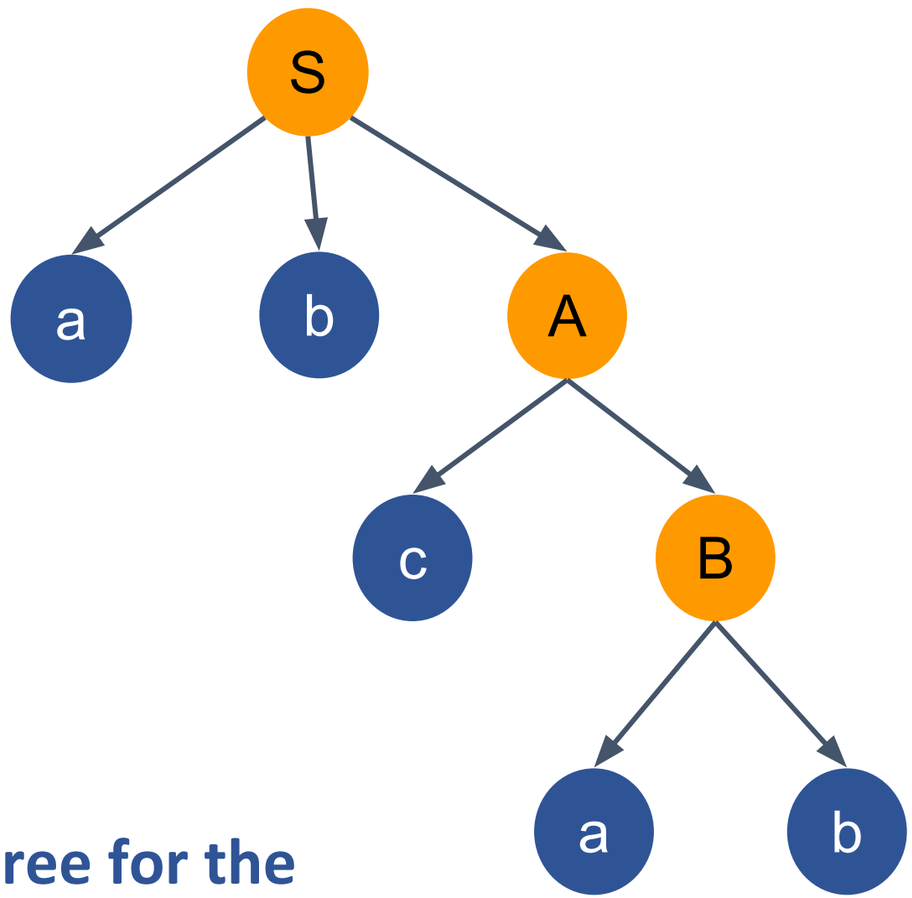
Right Linear

S→abA

A→cB|aC

B→ab

C→b



Construct Parse tree for the String “abcab”

https://www.slideserve.com/kaloni/how-to-convert-a-left-linear-grammar-to-a-right-linear-grammar

Unit 2 - Which one is easier,Left or Right Linear Grammar?

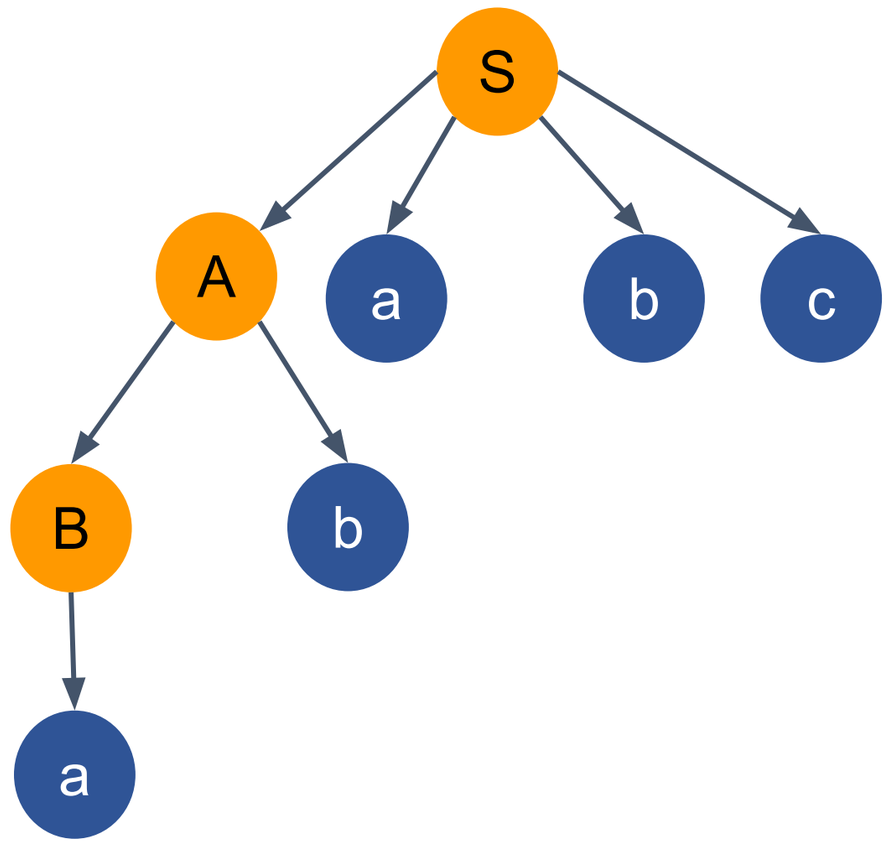
Left Linear

$S \rightarrow Aabc$

$A \rightarrow Bb|C$

$B \rightarrow a$

$C \rightarrow b$



Construct Parse tree for the String “ababc”

https://www.slideserve.com/kaloni/how-to-convert-a-left-linear-grammar-to-a-right-linear-grammar

PES UNIVERSITY ONLINE

THANK YOU

Preet Kanwal

Department of Computer Science & Engineering

preetkanwal@pes.edu

+91 80 6666 3333 Extn 724