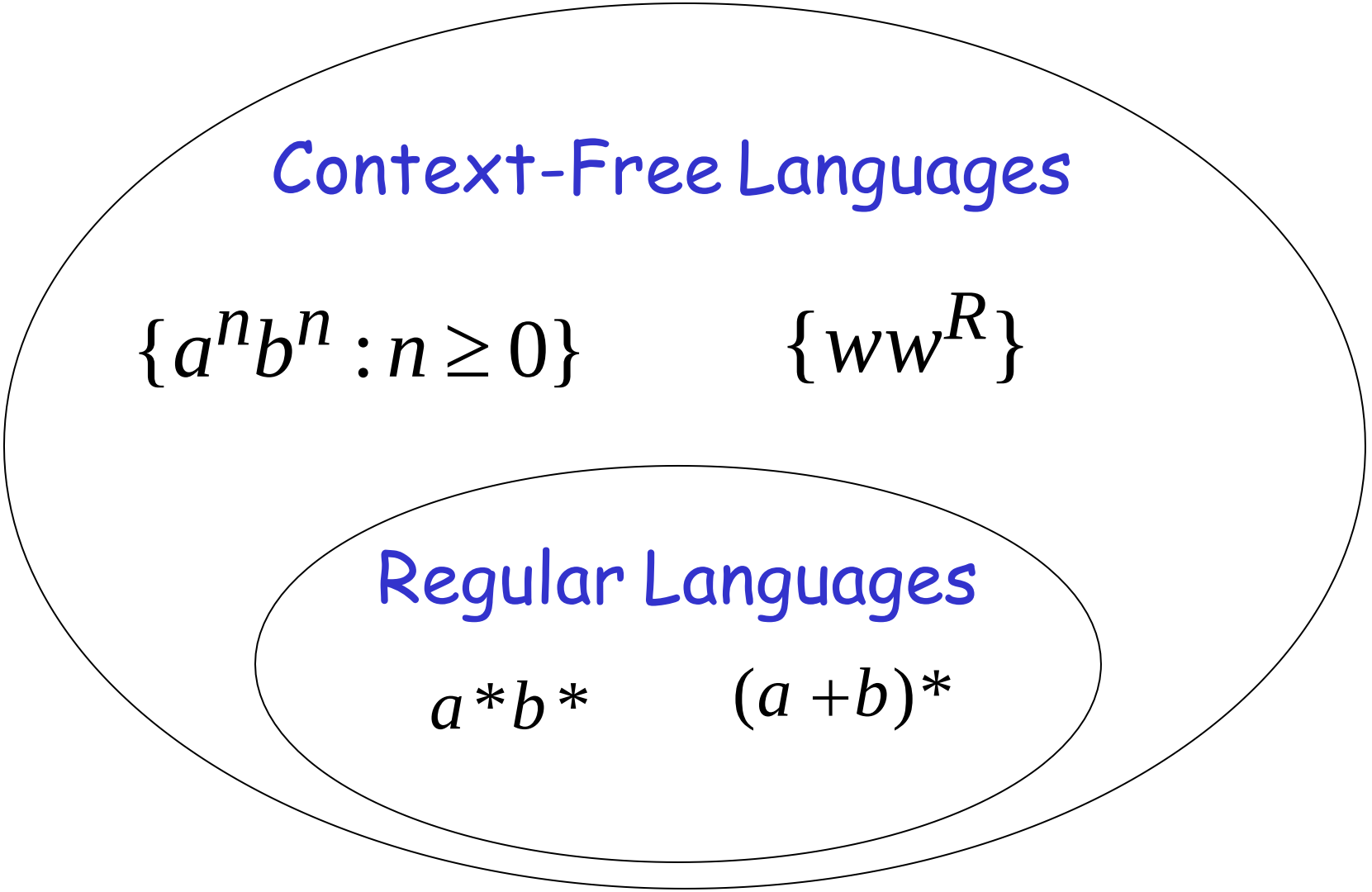
UNIT III:

CONTEXT-FREE GRAMMARAND LANGUAGES

Outline

- Introduction
- Formal Definition of Grammar, Notations,
- Derivation Process: Leftmost Derivation, Rightmost Derivation,
- derivation trees,
- Context Free Languages,
- Ambiguous CFG, Removal of ambiguity,
- Simplification of CFG, Normal Forms,
- Chomsky Hierarchy,
- Regular grammar, equivalence of RG(LRG and RLG) and FA.

Applications: Parse trees, Compilers, XML



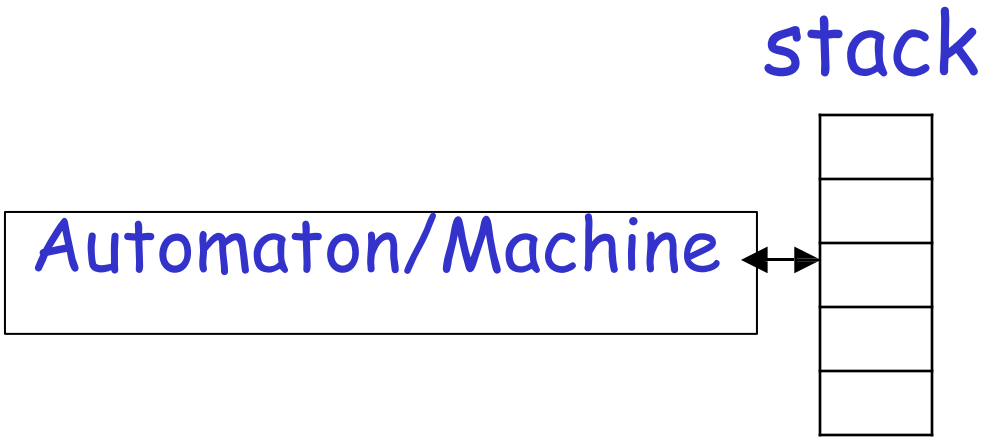
Context-Free Languages

Generator

Acceptor

Context-Free Grammars

Pushdown Automata



Context-Free Grammars

Grammars

Grammars express languages

Example: the English language grammar

<sentence> ❼ <noun_phrase> <predicate>

<noun_phrase> ❼ <article> <noun>

<predicate> ❼ <verb>

$$\langle article \rangle \rightarrow a$$

$$\langle article \rangle \rightarrow the$$

$$\langle noun \rangle \rightarrow cat$$

$$\langle noun \rangle \rightarrow dog$$

$$\langle verb \rangle \rightarrow runs$$

$$\langle verb \rangle \rightarrow sleeps$$

Generate the string "the dog sleeps":

$$\begin{aligned}
\langle sentence \rangle &\Rightarrow \langle noun_phrase \rangle \ \langle predicate \rangle \\
&\Rightarrow \langle noun_phrase \rangle \ \langle verb \rangle \\
&\Rightarrow \langle article \rangle \ \langle noun \rangle \ \langle verb \rangle \\
&\Rightarrow the \ \langle noun \rangle \ \langle verb \rangle \\
&\Rightarrow the \ dog \ \langle verb \rangle \\
&\Rightarrow the \ dog \ sleeps
\end{aligned}$$

Derivation

Derivation of string "a cat runs":

$$
\begin{aligned}
\langle sentence \rangle &\Rightarrow \langle noun\_phrase \rangle \ \langle predicate \rangle \\
&\Rightarrow \langle noun\_phrase \rangle \ \langle verb \rangle \\
&\Rightarrow \langle article \rangle \ \langle noun \rangle \ \langle verb \rangle \\
&\Rightarrow a \ \langle noun \rangle \ \langle verb \rangle \\
&\Rightarrow a \ cat \ \langle verb \rangle \\
&\Rightarrow a \ cat \ runs
\end{aligned}
$$

Language of the grammar:

L = { "a cat runs",
"a cat sleeps",
"the cat runs",
"the cat sleeps",
"a dog runs",
"a dog sleeps",
"the dog runs",
"the dog sleeps" }

the English language grammar

<sentence> ➔ <noun_phrase> <predicate>

<noun_phrase> ➔ <article> <noun>

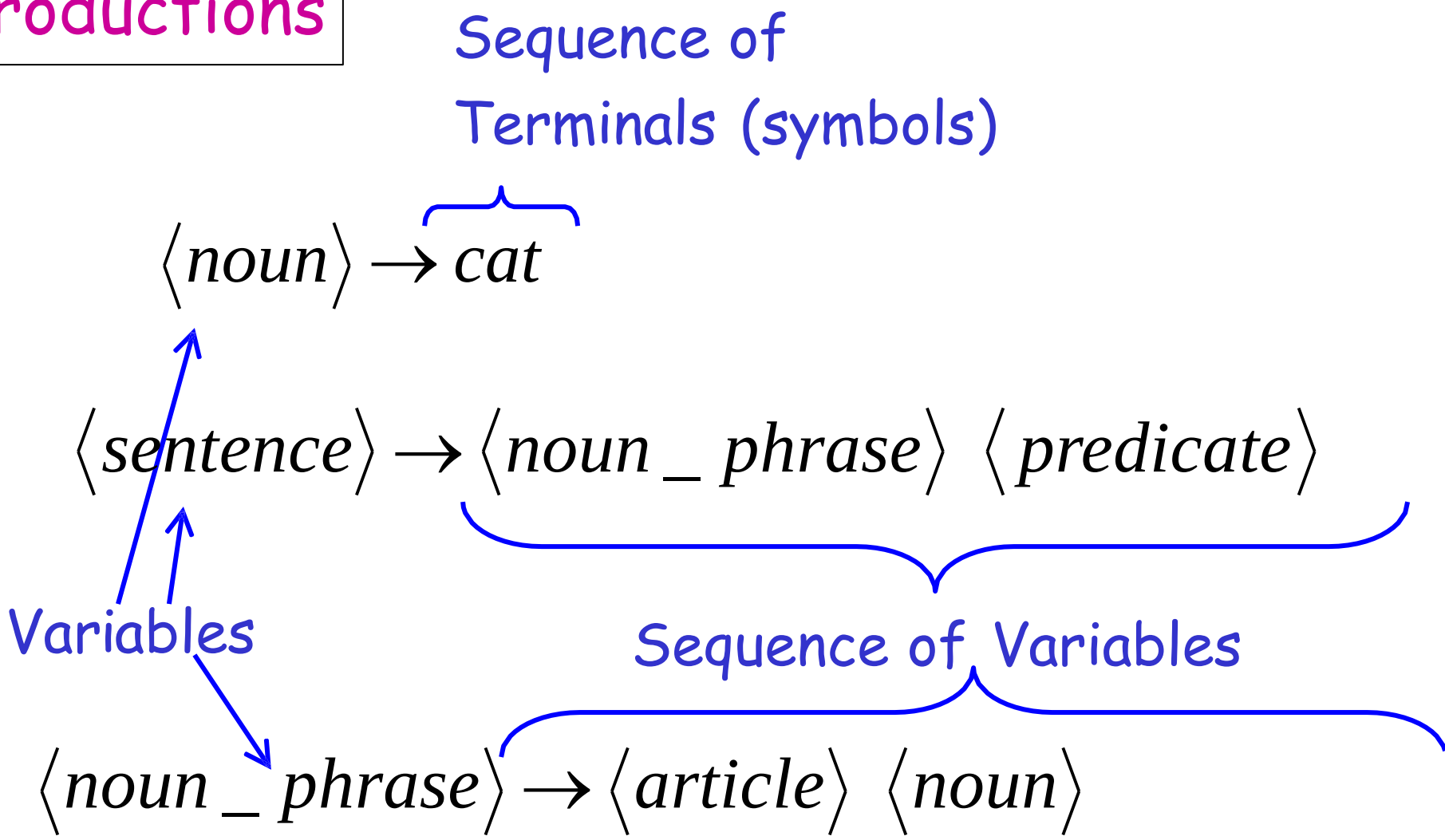
<predicate> ➔ <verb>

<noun> ➔ cat/dog

<article> ➔ a/an/the

<verb> ➔ sleeps/runs

Productions



Grammar for English Language

Start Variable

Productions

$$\langle sentence \rangle \rightarrow \langle noun\ \_phrase \rangle\ \langle predicate \rangle$$

$$\langle noun\ \_\ phrase \rangle \rightarrow \langle article \rangle\ \langle noun \rangle$$

$$\langle noun \rangle \rightarrow \text{"any noun e.g. cat or dog"}$$

$$\langle predicate \rangle \rightarrow \langle verb \rangle$$

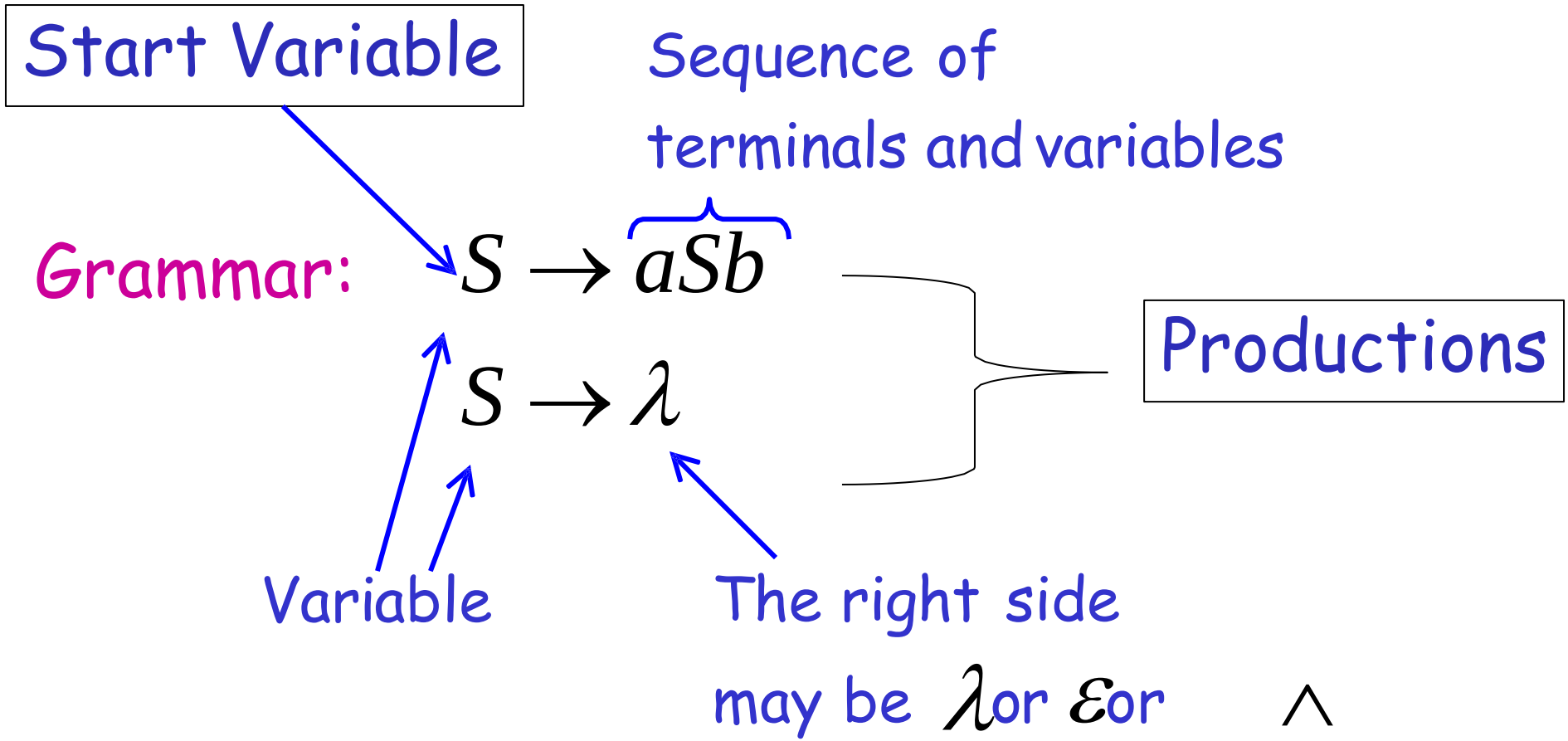
$$\langle article \rangle \rightarrow \text{"the, a, or an"}$$

$$\langle verb \rangle \rightarrow \text{"any verb e.g. runs or sleeps"}$$

Variables

Sequence of Variables and terminals

Another Example

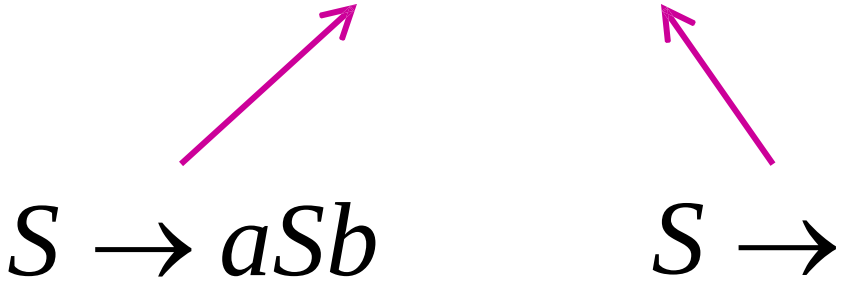


Grammar: $S \to aSb$

$S \to \lambda$

Derivation of string ab:

$$S \Rightarrow aSb \Rightarrow ab$$



Grammar: $S \to aSb$

$S \to \lambda$

Derivation of string $aabb$:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

$S \to aSb$ $\quad$ $S \to aSb$ $\quad$ $S \to \lambda$

Grammar: $S \to aSb$

$S \to \lambda$

Other derivations:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb$$

$$\Rightarrow aaaaSbbbb \Rightarrow aaaabbbb$$

Grammar: $S \to aSb$

$S \to \lambda$

Language of the grammar:

$$L = \{a^n b^n : n \geq 0\}$$

A Convenient Notation

We write: $S \overset{*}{\Rightarrow} aaabbb$

for zero or more derivationsteps

Instead of:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

In general we write: $w_1 \overset{*}{\Rightarrow} w_n$

If: $w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \square \Rightarrow w_n$

in zero or more derivation steps

Trivially: $w \overset{*}{\Rightarrow} w$

Example Grammar

$$S \to aSb$$

$$S \to \lambda$$

Possible Derivations

$$S \overset{*}{\Rightarrow} \lambda$$

$$S \overset{*}{\Rightarrow} ab$$

$$S \overset{*}{\Rightarrow} aaabbb$$

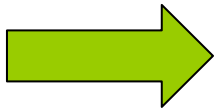
$$S \overset{*}{\Rightarrow} aaSbb \overset{*}{\Rightarrow} aaaaSbbbb$$

Another convenient notation:

$$\langle article \rangle \rightarrow a$$
$$\langle article \rangle \rightarrow the$$

$$\langle article \rangle \rightarrow a \mid the$$

$$S \rightarrow aSb$$
$$S \rightarrow \lambda$$



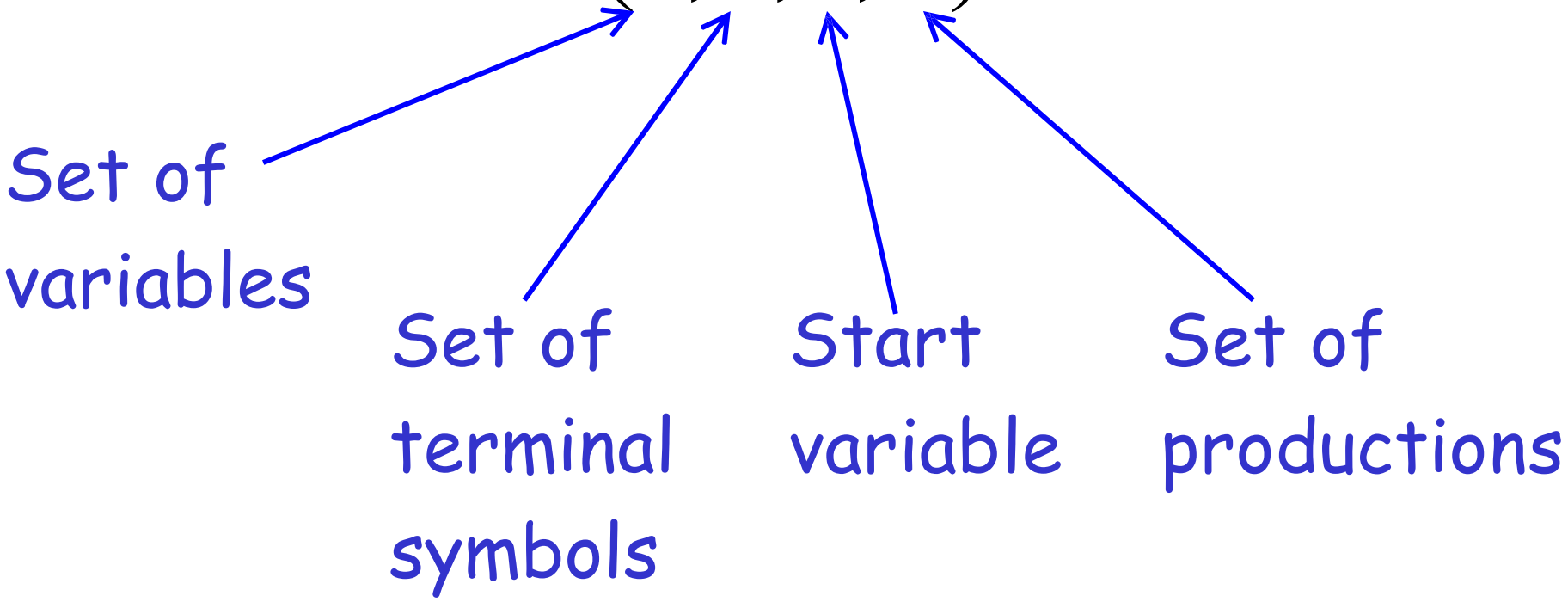
$$S \rightarrow aSb \mid \lambda$$

Recall

- Context Free Grammars
- Formal Definition
- Context Free Language
- Derivation

Formal Definitions

Grammar: $G = (V, T, S, P)$



Grammar for English Language

$$\langle sentence \rangle \rightarrow \langle noun\ \_\ phrase \rangle\ \langle predicate \rangle$$

$$\langle noun\ \_\ phrase \rangle \rightarrow \langle article \rangle\ \langle noun \rangle$$

$$\langle noun \rangle \rightarrow \text{"any noun e.g. cat or dog"}$$

$$\langle predicate \rangle \rightarrow \langle verb \rangle$$

$$\langle article \rangle \rightarrow \text{"the, a, or an"}$$

$$\langle noun \rangle \rightarrow \text{"any verb e.g. runs or sleeps"}$$

Context-Free Grammar: $G = (V, T, S, P)$

All productions in P are of the form

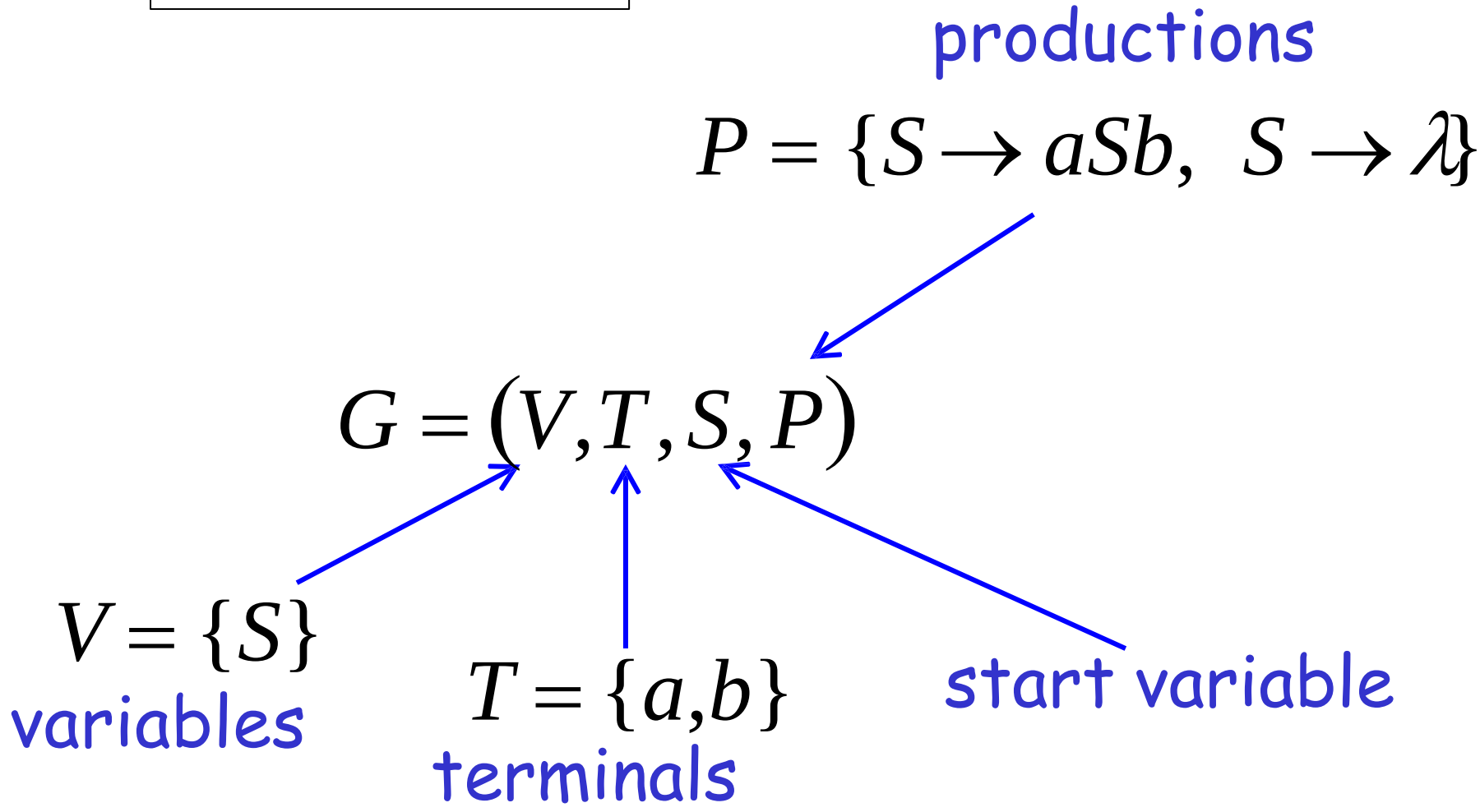
$$A \rightarrow s$$

Variable

String of variables and terminals

Example of Context-Free Grammar

$$S \to aSb \mid \lambda$$



Language of a Grammar:

For a grammar G with start variable S

$$L(G) = \{w: \quad S \overset{*}{\Rightarrow} w, \quad w \in T^*\}$$

String of terminals or λ

Example:

context-free grammar G : $\boxed{S \to aSb \mid \lambda}$

$$L(G) = \{a^n b^n : \quad n \geq 0\}$$

Since, there is derivation

$$S \overset{*}{\Rightarrow} a^n b^n \quad n \quad \text{for any} \quad n \geq 0$$

Context-Free Language:

A language L is context-free

if there is a context-free grammar G

with $L = L(G)$

Example:

$$L = \{a^n b^n : \quad n \geq 0\}$$

is a context-free language

since context-free grammar G:

$$S \rightarrow aSb \mid \lambda$$

generates $L(G) = L$

Another Example

Context-free grammar G:

$$S \rightarrow aSa|bSb|\lambda$$

Example derivations:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba$$

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaba$$

$$L(G) = \{ww^R : \quad w \in \{a,b\}^*\}$$

Palindromes of even length

Another Example

Context-free grammar :

G: S => (S) | SS | λ

- Language of balanced parenthesis.
- e g ()((((()))))((()))....

Example #3

A grammar for $L = \{0^m 1^n \mid m \geq n\}$

CFG G:

$S \Rightarrow 0S1 \mid A$

$A \Rightarrow 0A \mid \lambda$

How would you interpret thestring "00000111" using this grammar?

Terminal or Variable ?

- $S \rightarrow (S) \mid S + S \mid S \times S \mid A$
  $A \rightarrow 1 \mid 2 \mid 3$

  - **The terminal symbols are { (,), +, ×, 1, 2, 3}**
  - **The variable symbols are S and A**

Example CFG for ?:

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

P:

(1)	**S –> ABC**	
(2)	**A –> aA**	**A –> aA | ε**
(3)	**A –> ε**	
(4)	**B –> bB**	**B –> bB | ε**
(5)	**B –> ε**	
(6)	**C –> cC**	**C –> cC | ε**
(7)	**C –> ε**	

Example Derivations:

S => ABC	**(1)**
=> BC	**(3)**
=> C	**(5)**
=> ε	**(7)**

S => ABC	**(1)**
=> aABC	**(2)**
=> aaABC	**(2)**
=> aaBC	**(3)**
=> aabBC	**(4)**
=> aabC	**(5)**
=> aabcC	**(6)**
=> aabc	**(7)**

Note that G generates the language a*b*c*

CFG – Generating strings

- E.g. : $G = (\{S\}, \{a,b\}, P, S)$

  $P: \quad S \rightarrow aS \mid bS \mid \lambda$

- The language above can also be defined using regular expression:

  $L(G) = (a + b)^*$

Recall

- Context Free Grammer
- Context Free Language
- Construction of CFL from CFG
- Construction of CFG from CFL

Example

- Let G be the grammar given by the production

$$S \rightarrow aSa \mid aBa$$

$$B \rightarrow bB \mid b$$

- Then $L(G) = \{a^n b^m a^n \mid n > 0, m > 0\}$

Example

- Let $L(G) = \{a^n b^m c^m d^{2n} \mid n \geq 0, m > 0\}$
- Then the production rules forthis grammar is:

$$S \rightarrow aSdd \mid A$$

$$A \rightarrow bAc \mid bc$$

Example

- A string **w** is a palindrome if $w = w^R$
- The set of palindrome over **{a, b}** can be derived using rules:

$$S \rightarrow a \mid b \mid \lambda$$

$$S \rightarrow aSa \mid bSb$$

Example

$$\{a^m b^n c^{m+n} \mid m,n \geq 0\}$$

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

$S \rightarrow S' \mid a\ S\ c$

$S' \rightarrow \varepsilon \mid b\ S'\ c$

- S ❼ S +T
  | T T
  ❼
  T*F | F F
  ❼ (S) | a
- Give a derivation for the string (a+a)*a.
- S=>T
  =>T * F
  =>F * F
  =>(S) * F
  =>(S + T) * F
  =>(T + T) * F
  =>(F + F) * F

How to convert RE to CFG

- **General rules:**
  - **a + b** $S \rightarrow a \mid b$
  - **ab** $S \rightarrow aA$, $A \rightarrow b$
  - **a*** $S \rightarrow aS \mid \lambda$

More Examples

- What is the language of this grammar ?
- $S \rightarrow aA$ — $a(a + b)^*b$

  $A \rightarrow aA \mid bA \mid b$
- $S \rightarrow aA$ — aa^*bb^*

  $A \rightarrow aA \mid bB$

  $B \rightarrow bB \mid \lambda$
- $S \rightarrow aS \mid bA$ — a^*bba^*

  $A \rightarrow bB$

  $B \rightarrow aB \mid \lambda$

Example

- Consider Grammer

  S ➆A 1 B

  A ➆0 A | ε

  B ➆0 B | 1B | ε

CFL:-

$$0^*1(0+1)^*$$

Example

- $S \rightarrow aaB \mid Abb$
  $A \rightarrow a \mid aA$
  $B \rightarrow b \mid bB$

- What is $L(G)$?

- $L(G) = \{aab^n \mid n > 1\} \cup \{a^nbb \mid n >= 1\}$

Example

- Write CFL for the following CFG

  S ➆ a S c | A | ε

  A ➆ a A b | ε

$CFL = a^{n+m} b^n c^m \mid n,m >= 0$

Example

- Write CFL for the following CFG

  S → 0 S 0 | A | ε

  A → 1 A 0 | ε

CFL =

$$0^n 1^m 0^{n+m}$$

Recall

- Construction of CFL from CFG
- Construction of CFG from CFL

Example

- S ➐ S S | aSaSb | aSbSa | bSaSa | ε

- CFL :-
- all string generated by the grammar have twice many *a* as *b* .

Example

- S ➔ 1S ➔ 0A0S ➔ ε
  A ➔ 1A ➔ ε
- CFL:-
- Generates all string which contain even number of 0's

Example

- Find CFL for
  - S ➆ aB | bA
    - A ➆ a | a S | b A A
    - B ➆ b | b S | a B B
- CFL
- Language which contains strings having equal number of a's and equal number of b's.

Example

- Let $A=\{ a^n b^n b^m c^m \mid n; m \geq 0 \}$
- Grammar for above language is:

  $S \rightarrow S1S2$

  $S1 \rightarrow aS1b \mid \varepsilon$

  $S2 \rightarrow bS2c \mid \varepsilon$
- Give a context-free grammar for the language,

  $L = \{0^n 1^n 2^m 3^m \mid n \geq 1 \text{ and } m \geq 1\}$

Example

- Write context-free grammar for L = {w | w starts and ends with the same symbol}
- CFG: -
- S ➆ 0T0 | 1T1
  T ➆ 0T | 1T
  | ε

Example

- Write context-free grammar L = {w | the length of w is odd}:
- CFG: -
-

Example

- Write context-free grammar L = {w | the length of w is odd}:
- CFG: -
- S ➆0 S 0 | 0S1 |1S0 | 1S1| 0 | 1
- S ➆ 0T | 1T
  T ➆ 0S | 1S | ε
-

Example

- Write context-free grammar L = {w | the length of w is odd and its middle is 0}:
- CFG: -

Example

- Write context-free grammar L = {w | the length of w is odd and its middle is 0}:
- CFG: -
- S ➐ 0S0 | 1S0 | 0S1 | 1S1 | 0

Example

- Write context-free grammar for
- $L = \{ a^i b^j c^k \mid i, j, k \geq 0, \text{ and } i = j \text{ or } i = k \}$

- CFG: -
- $S \rightarrow XY \mid W$
  $X \rightarrow aXb \mid \varepsilon$
  $Y \rightarrow cY \mid \varepsilon$
  $W \rightarrow aW\,c \mid Z$
  $Z \rightarrow bZ \mid \varepsilon$

Example

- Write CFG for

(letter)(letter + digit)*

CFG =

S → letter S1

S1 → letter S1 | digit S1 |

ε

Problem

Write a CFG which generates the language L denoted by

- $(a+b)^*bbb(a+b)^*$
- $\{0^m|^n\ 0^{m+n} | m,n \geq 0\}$

(Nov-2017, 6 Marks)

Construct contextfree grammar corresponding to regular expression.

$(0 + l)\ l^*\ (l + (0l)^*)$

(Nov-2017, 6 Marks)

Recall

- Construction of CFL from CFG
- Construction of CFG from CFL
- Exercise from University Papers

Derivation Order
and
Derivation Trees

- Strings α yields string β, written , $\alpha \overset{*}{\Longrightarrow} \beta$ if it is possible to get from α to β using the productions.

- A derivation of β is the sequence of steps that gets to β (sentential form).

- A leftmost derivation is where at each stage one replaces the leftmost variable.

- A rightmost derivation is defined similarly where at each stage one replaces the rightmost variable.

Derivation Order

Consider the following example grammar with 5 productions:

1. $S \rightarrow AB$
2. $A \rightarrow aaA$
3. $A \rightarrow \lambda$
4. $B \rightarrow Bb$
5. $B \rightarrow \lambda$

$$S \rightarrow A\,B$$
$$A \rightarrow a\,a\,A$$
$$A \rightarrow \lambda$$
$$B \rightarrow B\,b$$
$$B \rightarrow \lambda$$

$$S \rightarrow A\,B$$
$$A \rightarrow a\,a\,A \mid \lambda$$
$$B \rightarrow B\,b \mid \lambda$$

1. $S \to AB$ 2. $A \to aaA$ 4. $B \to Bb$

3. $A \to \lambda$ 5. $B \to \lambda$

Leftmost derivation order of string aab:

$$S \overset{1}{\Rightarrow} AB$$

At each step, we substitute the leftmost variable

1. $S \to AB$

2. $A \to aaA$

3. $A \to \lambda$

4. $B \to Bb$

5. $B \to \lambda$

Leftmost derivation order of string aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB$$

At each step, we substitute the leftmost variable

1. $S \to AB$ 2. $A \to aaA$ 4. $B \to Bb$

3. $A \to \lambda$ 5. $B \to \lambda$

Leftmost derivation order of string aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB$$

At each step, we substitute the leftmost variable

1. $S \to AB$
2. $A \to aaA$
3. $A \to \lambda$
4. $B \to Bb$
5. $B \to \lambda$

Leftmost derivation order of string aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{3}{\Rightarrow} aaB$$

At each step, we substitute the leftmost variable

1. $S \to AB$ 2. $A \to aaA$ 4. $B \to Bb$

3. $A \to \lambda$ 5. $B \to \lambda$

Leftmost derivation order of string aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{3}{\Rightarrow} aaB$$

At each step, we substitute the leftmost variable

1. $S \to AB$ 2. $A \to aaA$ 4. $B \to Bb$

3. $A \to \lambda$ 5. $B \to \lambda$

Leftmost derivation order of string aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{3}{\Rightarrow} aaB \overset{4}{\Rightarrow} aaBb$$

At each step, we substitute the leftmost variable

1. $S \to AB$ 2. $A \to aaA$ 4. $B \to Bb$

3. $A \to \lambda$ 5. $B \to \lambda$

Leftmost derivation order of string aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{3}{\Rightarrow} aaB \overset{4}{\Rightarrow} aaBb \overset{5}{\Rightarrow} aab$$

At each step, we substitute the leftmost variable

1. $S \to AB$ 2. $A \to aaA$ 4. $B \to Bb$

3. $A \to \lambda$ 5. $B \to \lambda$

Rightmost derivation order of string aab:

$$S \overset{1}{\Rightarrow} AB$$

At each step, we substitute the rightmost variable

1. $S \to AB$ 2. $A \to aaA$ 4. $B \to Bb$

3. $A \to \lambda$ 5. $B \to \lambda$

Rightmost derivation order of string aab :

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} ABb$$

At each step, we substitute the rightmost variable

1. $S \to AB$ 2. $A \to aaA$ 4. $B \to Bb$

3. $A \to \lambda$ 5. $B \to \lambda$

Rightmost derivation order of string aab :

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} ABb \overset{3}{\Rightarrow} Ab$$

At each step, we substitute the rightmost variable

1. $S \to AB$ 2. $A \to aaA$ 4. $B \to Bb$

3. $A \to \lambda$ 5. $B \to \lambda$

Rightmost derivation order of string aab :

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} ABb \overset{3}{\Rightarrow} Ab \overset{4}{\Rightarrow} aaAb$$

At each step, we substitute the rightmost variable

1. $S \to AB$ 2. $A \to aaA$ 4. $B \to Bb$

3. $A \to \lambda$ 5. $B \to \lambda$

Rightmost derivation order of string aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} ABb \overset{3}{\Rightarrow} Ab \overset{4}{\Rightarrow} aaAb \overset{5}{\Rightarrow} aab$$

At each step, we substitute the rightmost variable

1. $S \to AB$ 2. $A \to aaA$ 4. $B \to Bb$

3. $A \to \lambda$ 5. $B \to \lambda$

Leftmost derivation of aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{3}{\Rightarrow} aaB \overset{4}{\Rightarrow} aaBb \overset{5}{\Rightarrow} aab$$

Rightmost derivation of aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} ABb \overset{3}{\Rightarrow} Ab \overset{4}{\Rightarrow} aaAb \overset{5}{\Rightarrow} aab$$

Derivation Trees

- In a derivation tree
- Also called Parse Tree
- the root is the start variable of grammar,
- all internal nodes are labeled with variables,
- while all leaves are labeled with terminals.
- The children of an internal node are labeled from left to right with the right-hand side of the production used.

Derivation Trees

Consider the same example grammar:

$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

And a derivation of aab:

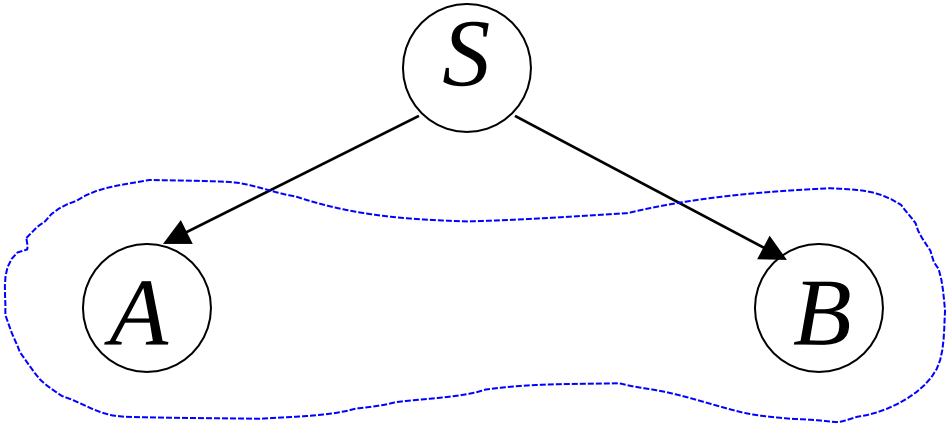
$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaBb \Rightarrow aab$$

$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

S

$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

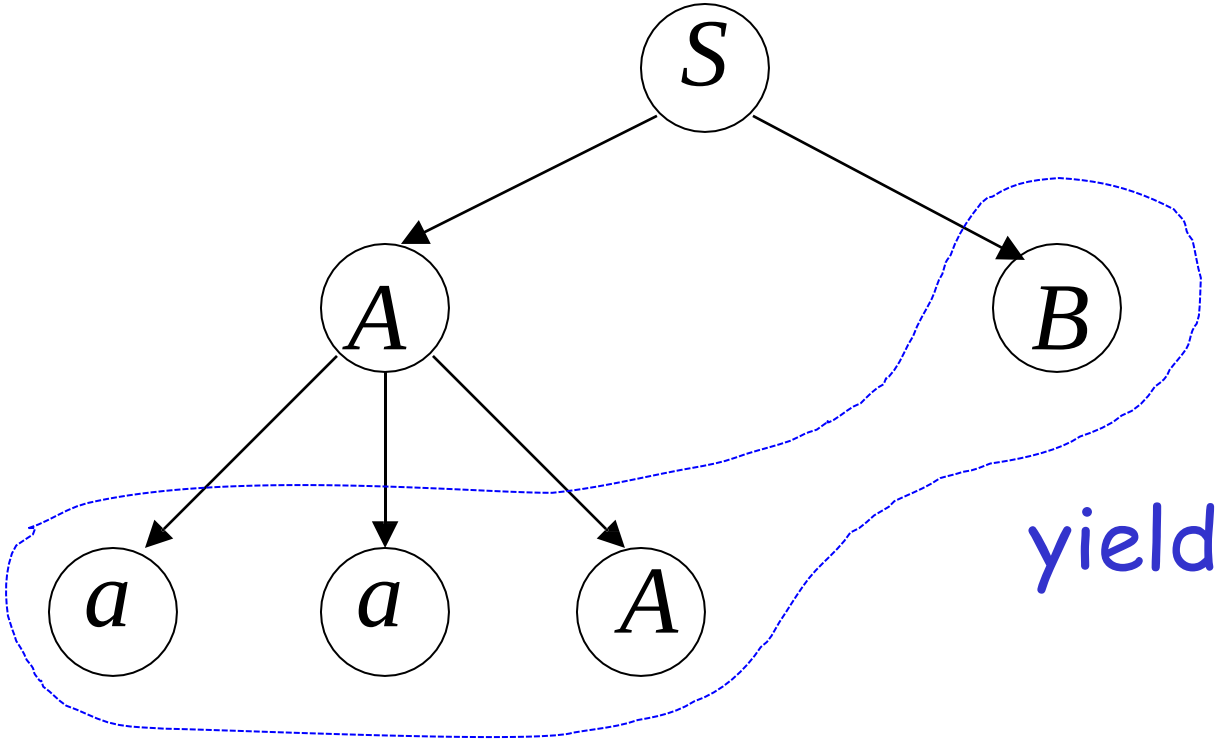
$$S \Rightarrow AB$$



yield AB

$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

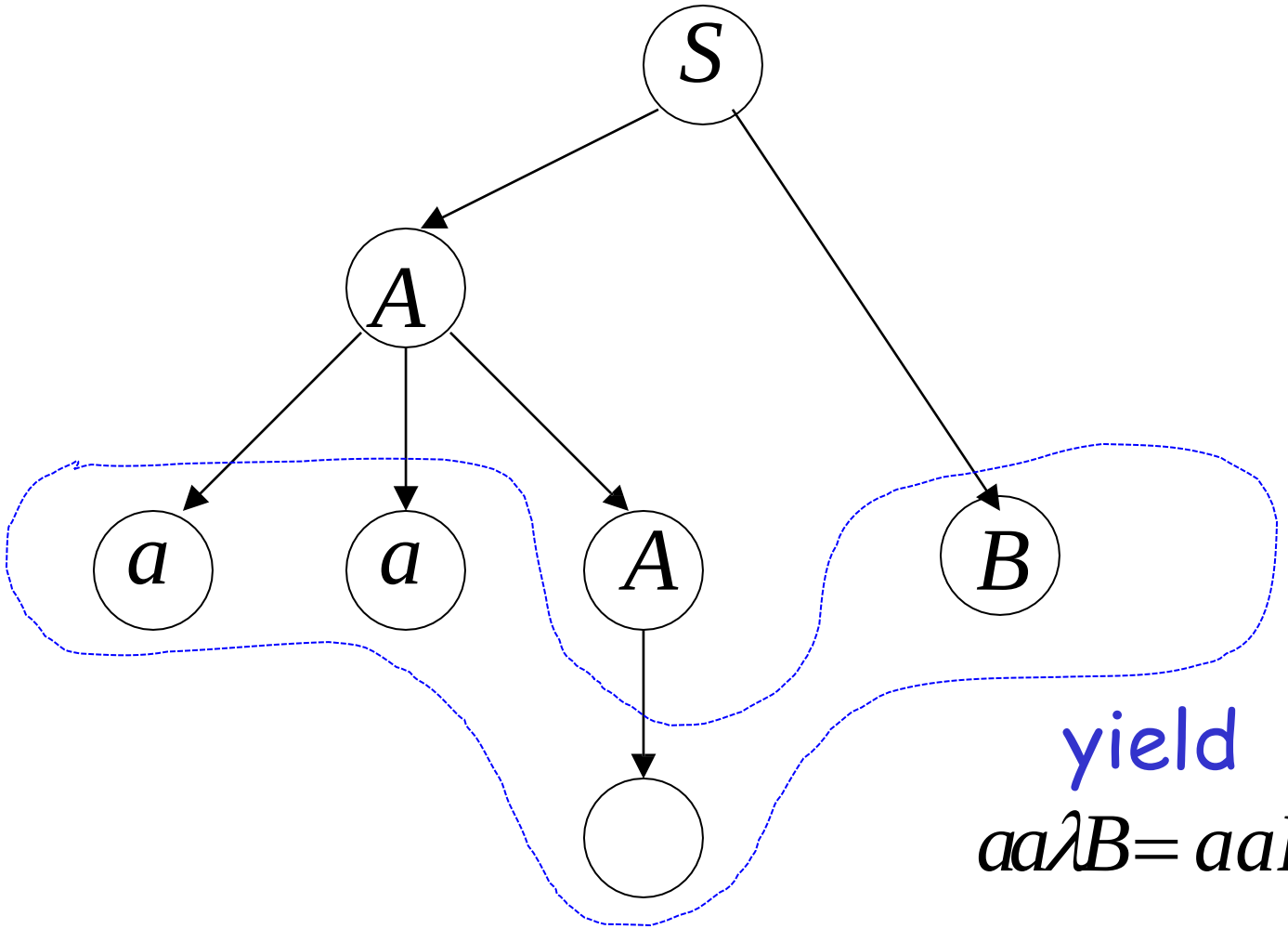
$$S \Rightarrow AB \Rightarrow aaAB$$



yield aaAB

$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB$

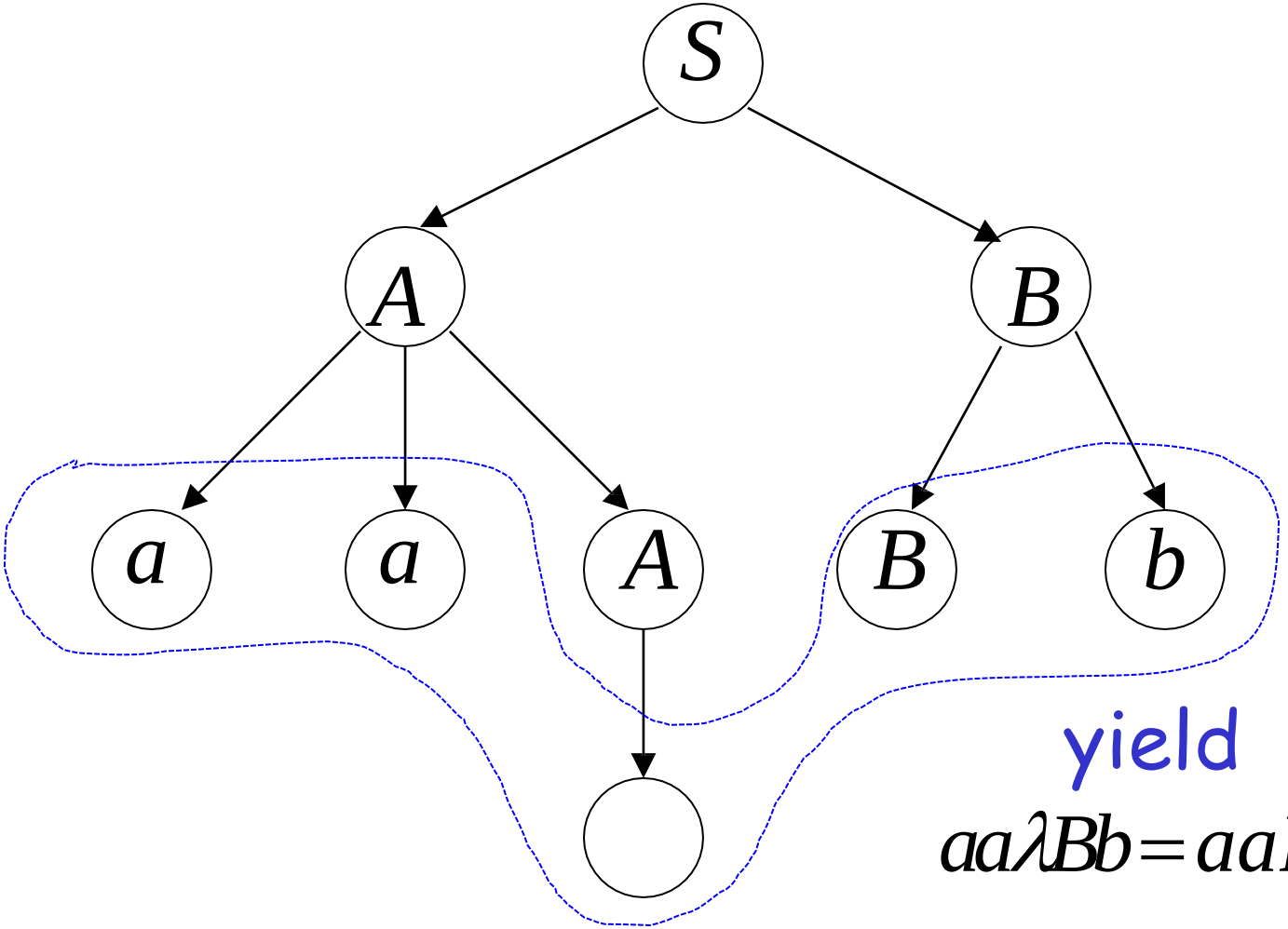


yield

$aa\lambda B = aaB$

$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb$



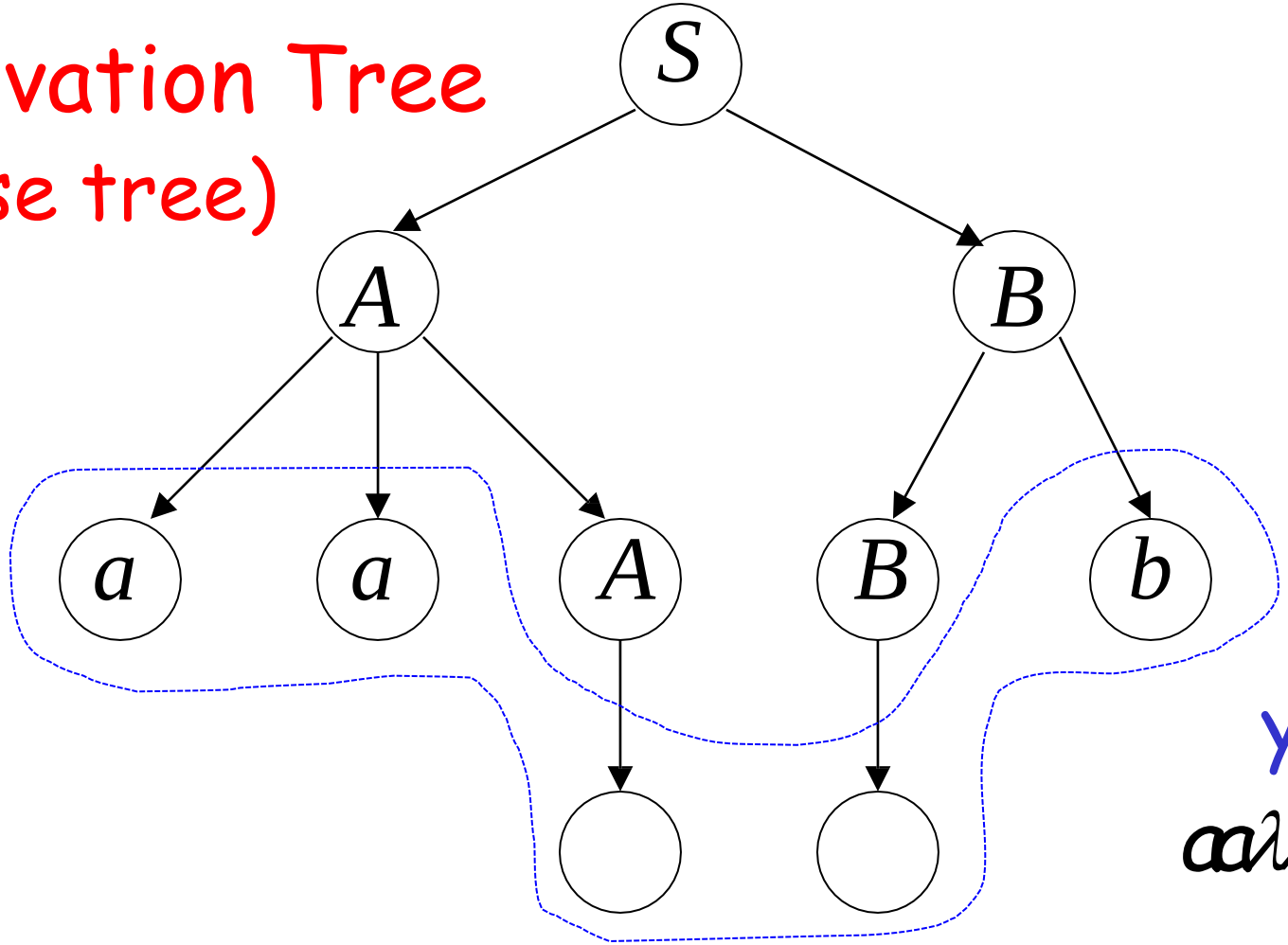
yield

$aa\lambda Bb = aaBb$

$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aab$$

Derivation Tree
(parse tree)



yield

$aa\lambda\lambda b = aab$

$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

Sometimes, derivation order doesn't matter

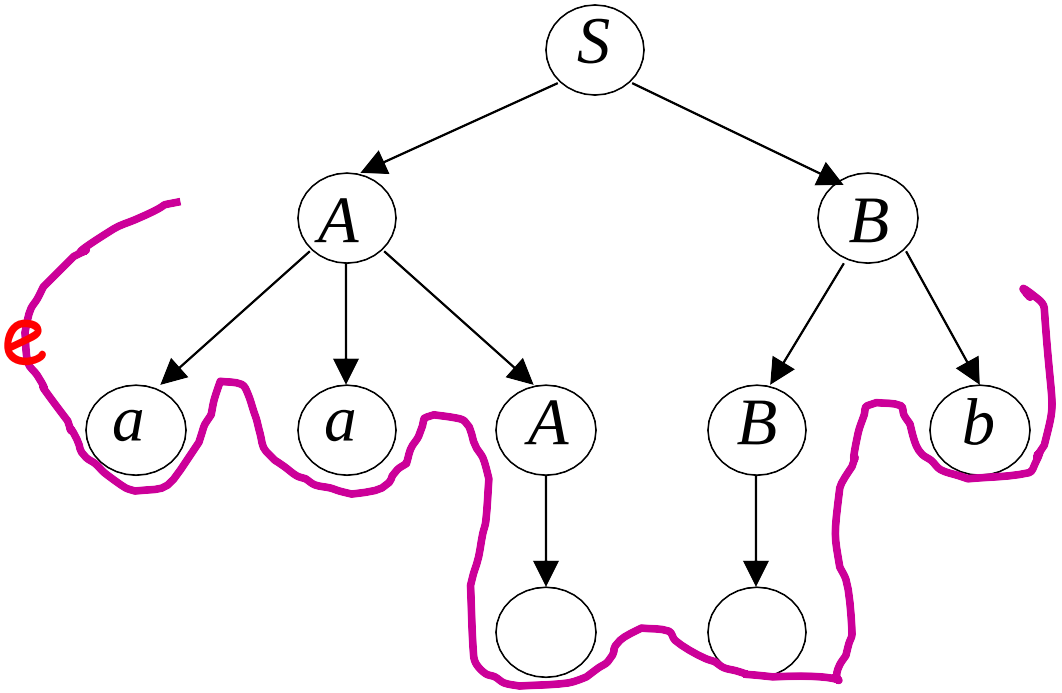
Leftmost derivation:

$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aab$$

Rightmost derivation:

$$S \Rightarrow AB \Rightarrow ABb \Rightarrow Ab \Rightarrow aaAb \Rightarrow aab$$

Give same derivation tree



CFG – Generating strings

$G = (V, \Sigma, P, S)$, $V = \{S, A\}$, $\Sigma = \{a, b\}$,

P: $S \rightarrow AA$

$A \rightarrow AAA \mid bA \mid Ab \mid a$

- **Four distinct derivations of ababaa in G:**

(a) $S \Rightarrow AA$
$\Rightarrow aA$
$\Rightarrow aAAA$
$\Rightarrow abAAA$
$\Rightarrow abaAA$
$\Rightarrow ababAA$
$\Rightarrow ababaA$
$\Rightarrow ababaa$

(b) $S \Rightarrow AA$
$\Rightarrow AAAA$
$\Rightarrow aAAA$
$\Rightarrow abAAA$
$\Rightarrow abaAA$
$\Rightarrow ababAA$
$\Rightarrow ababaA$
$\Rightarrow ababaa$

(c) $S \Rightarrow AA$
$\Rightarrow Aa$
$\Rightarrow AAAa$
$\Rightarrow AAbAa$
$\Rightarrow AAbaa$
$\Rightarrow AbAbaa$
$\Rightarrow Ababaa$
$\Rightarrow ababaa$

(d) $S \Rightarrow AA$
$\Rightarrow aA$
$\Rightarrow aAAA$
$\Rightarrow aAAa$
$\Rightarrow abAAa$
$\Rightarrow abAbAa$
$\Rightarrow ababAa$
$\Rightarrow ababaa$

(a) & (b) leftmost, (c) rightmost, (d) arbitrary

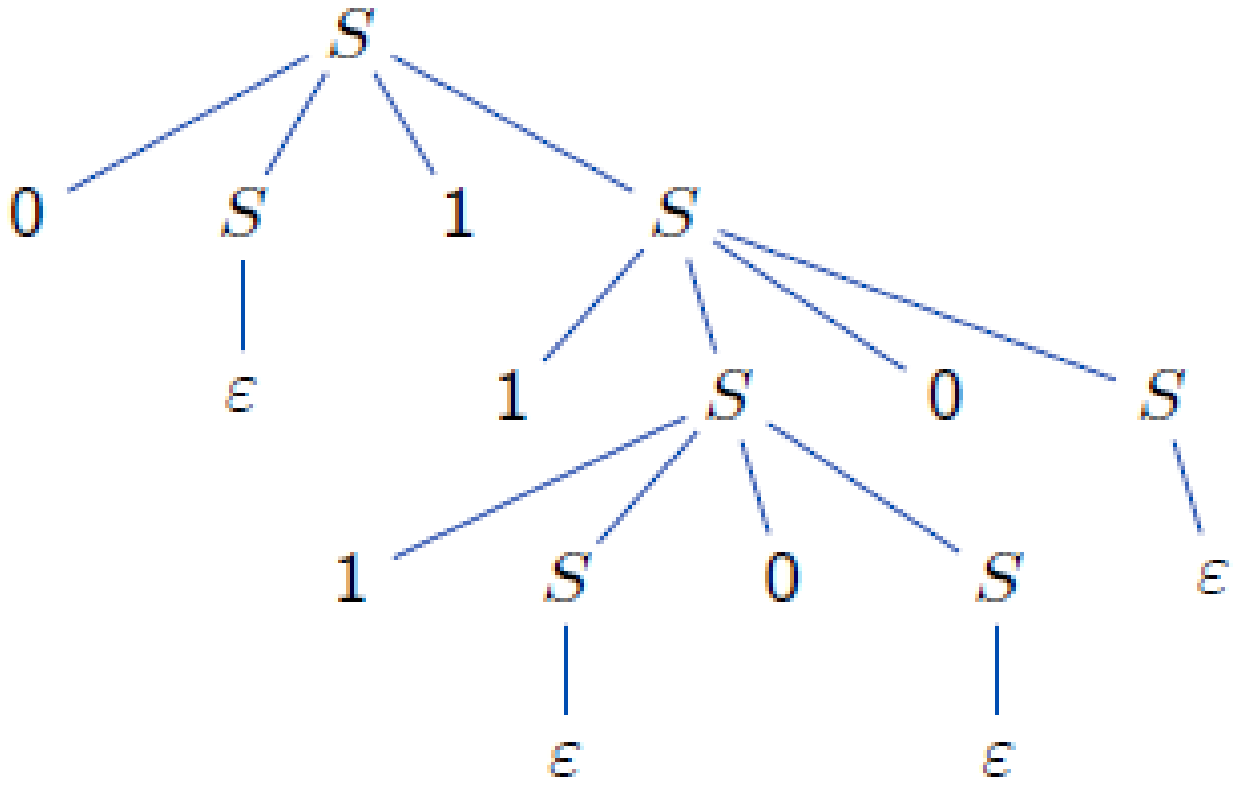
the CFG for equal 0's and 1's

S ➐ 0S1S | 1S0S |ε

- The derivation for 011100:

S => 0S1S => 01S => 011S0S => 0111S0S0S
=> 01110S0S => 011100S => 011100

- Here is derivation tree for 011100 in the above grammar



Problem

- Consider the grammar $G = \{(S, A), (0, 1), P, S\}$, where P consists of:
- $S \rightarrow 0AS \mid 0$
- $A \rightarrow S1A \mid SS \mid 10$
- Show the leftmost derivation and rightmost derivation for the input string '001100'.

Problem

- Write a CFG that will generate the language:
- $L = \{WCW^R \mid W \in (a, b)^*$, and W^R is the reverse of $W\}$
- Using this grammar, show the leftmost derivation, rightmost derivation, and derivation tree for the input string '*ababCbaba*'.

Problem

- Consider CFG with productions
- S -> aB/bA
- A -> a/aS/bAA
- B -> b/bS/aBB
- For string aaabbabbba
- Find leftmost and rightmost derivation.

(May-2018,4 Marks)

Problem

- Consider the following CFG:
- G = {(S, A), (a, b), P, S}
- Where P consists of :
- S→ aAs|a
- A→SbA|ss|ba
- Derive string 'aabbaa' using leftmost & right most derivation

(Aug-2017,4 Marks)

Recall

- Leftmost Derivation
- Rightmost Derivation
- Derivation Tree/Parse Tree
-

Ambiguity

Grammar for mathematical expressions

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

Example strings:

$$(a + a) + (a + a * (a + a))$$

Denotes any number

$$(a+a)+(a+a*(a+a))$$

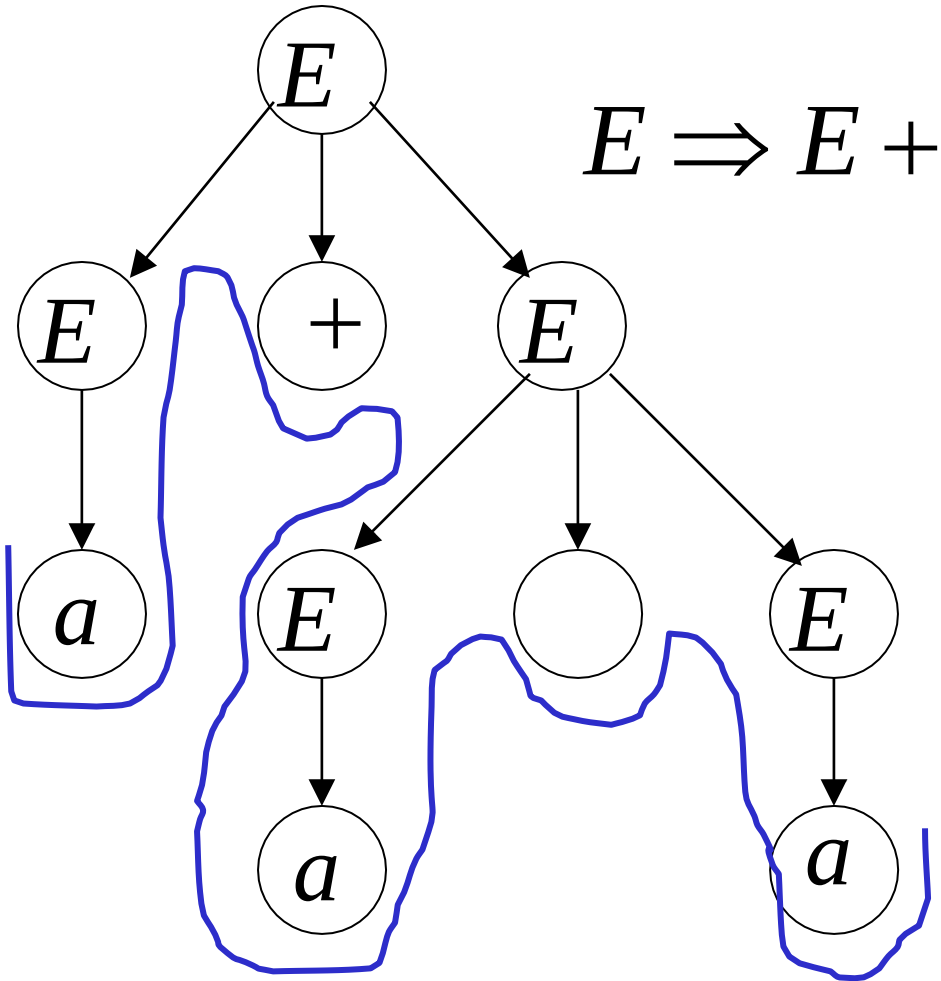
$$E \rightarrow E+E \mid (E)$$

$$E \rightarrow E*E \mid a$$

$$S \Rightarrow E + E$$
$$S \Rightarrow (E + E) + E$$
$$S \Rightarrow (a + a) + E$$
$$S \Rightarrow (a + a) + (E)$$
$$S \Rightarrow (a + a) + (E + E)$$
$$S \Rightarrow (a + a) + (a + E)$$
$$S \Rightarrow (a + a) + (a + E * E)$$
$$S \Rightarrow (a + a) + (a + a * E)$$
$$S \Rightarrow (a + a) + (a + a * (E))$$
$$S \Rightarrow (a + a) + (a + a * (E + E))$$
$$S \Rightarrow (a + a) + (a + a * (a + a))$$

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

A leftmost derivation for $a+a*a$



$$E \Rightarrow E+E \Rightarrow a+E \Rightarrow a+E*E$$

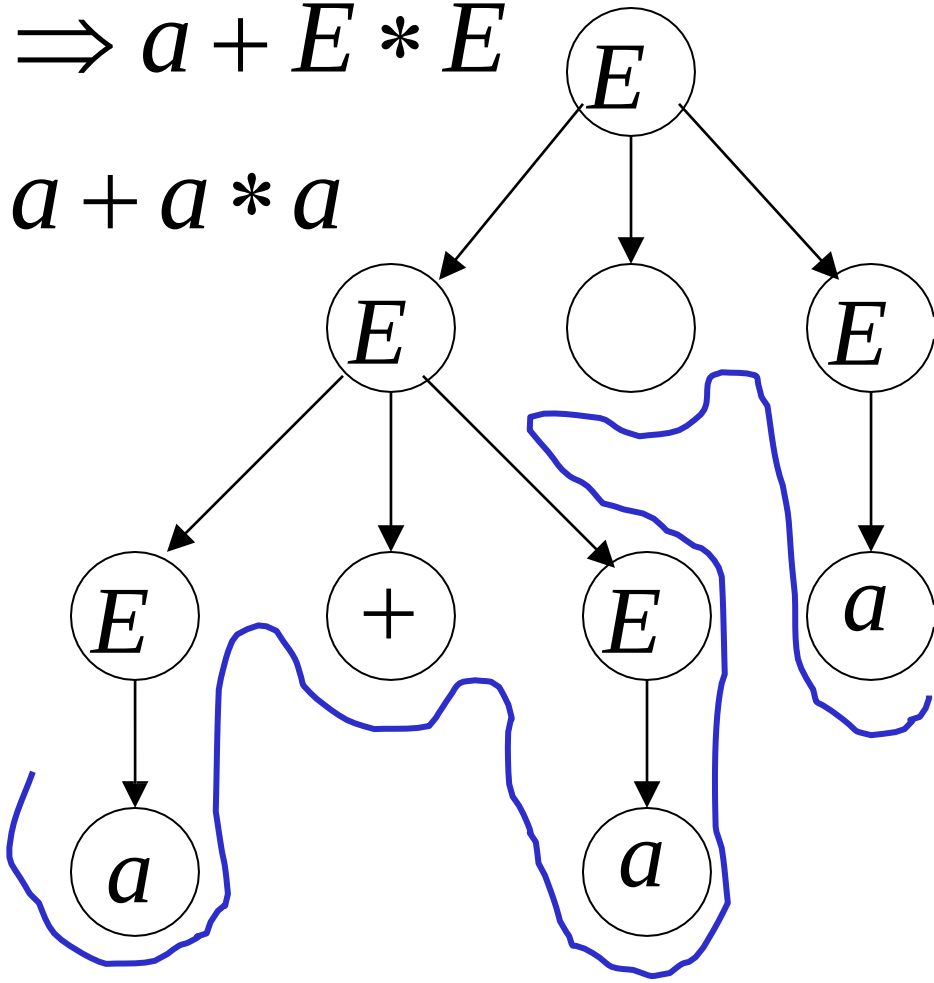
$$\Rightarrow a+a*E \Rightarrow a+a*a$$

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

Another leftmost derivation for $a+a*a$

$$E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow a+E*E$$

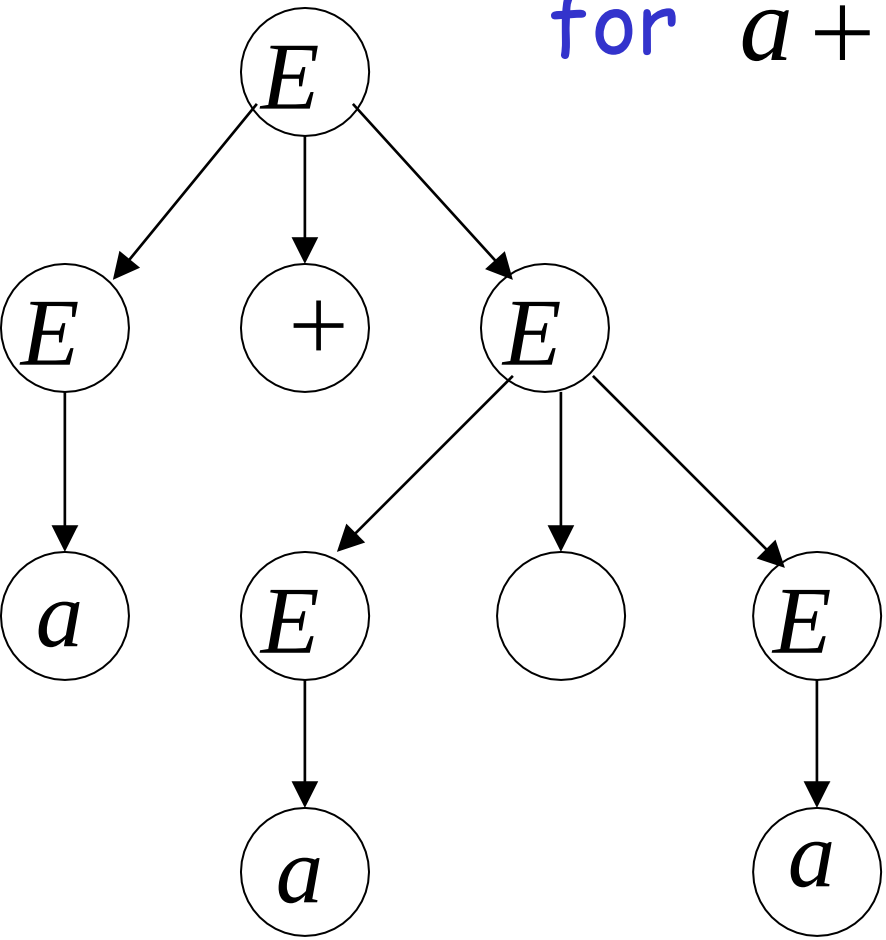
$$\Rightarrow a+a*E \Rightarrow a+a*a$$

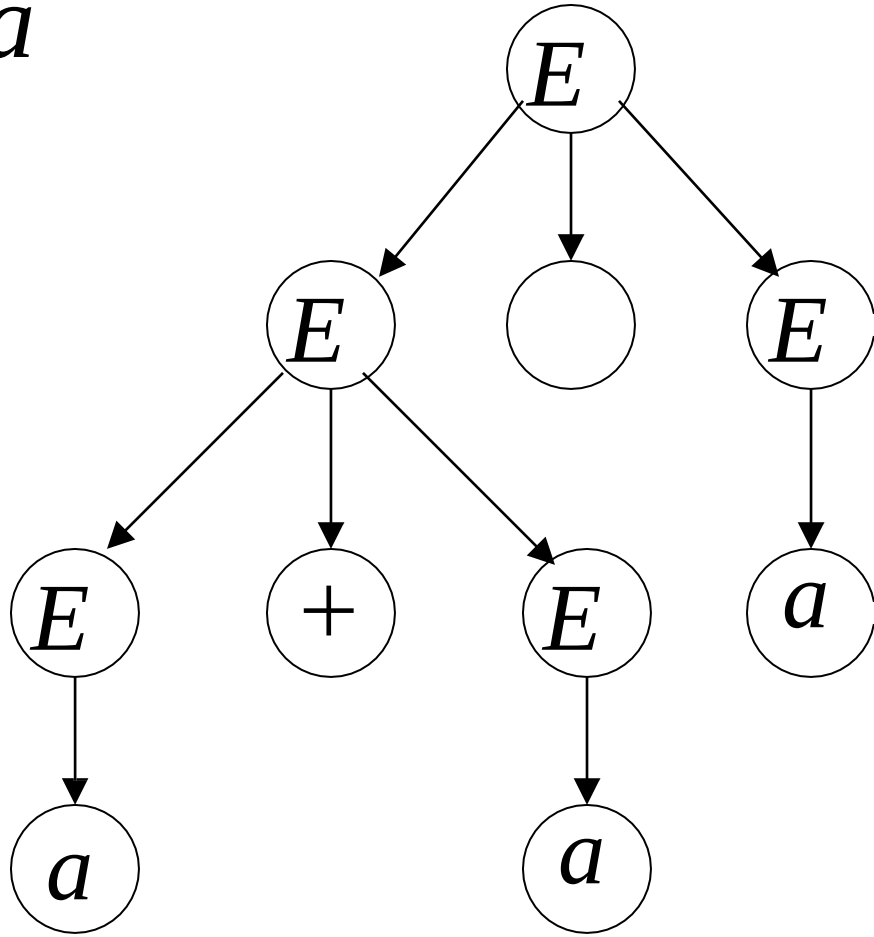


$$E \rightarrow E+E \mid E*E \mid (E) \mid a$$

Two derivation trees

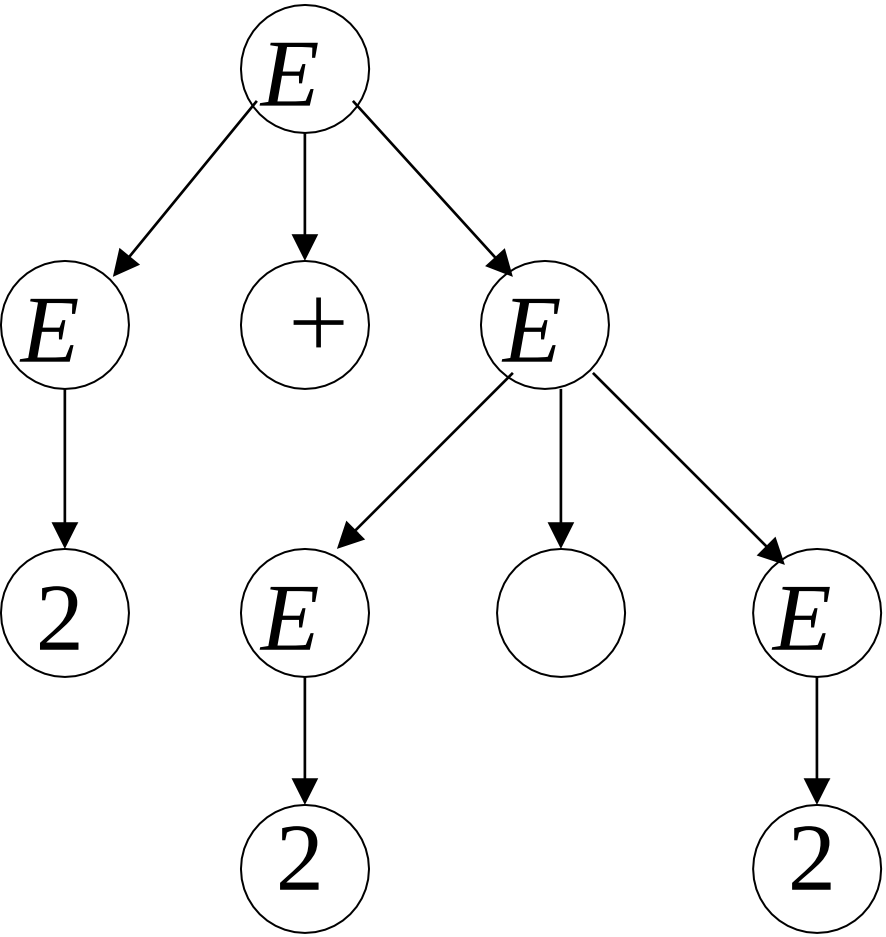
for $a+a*a$

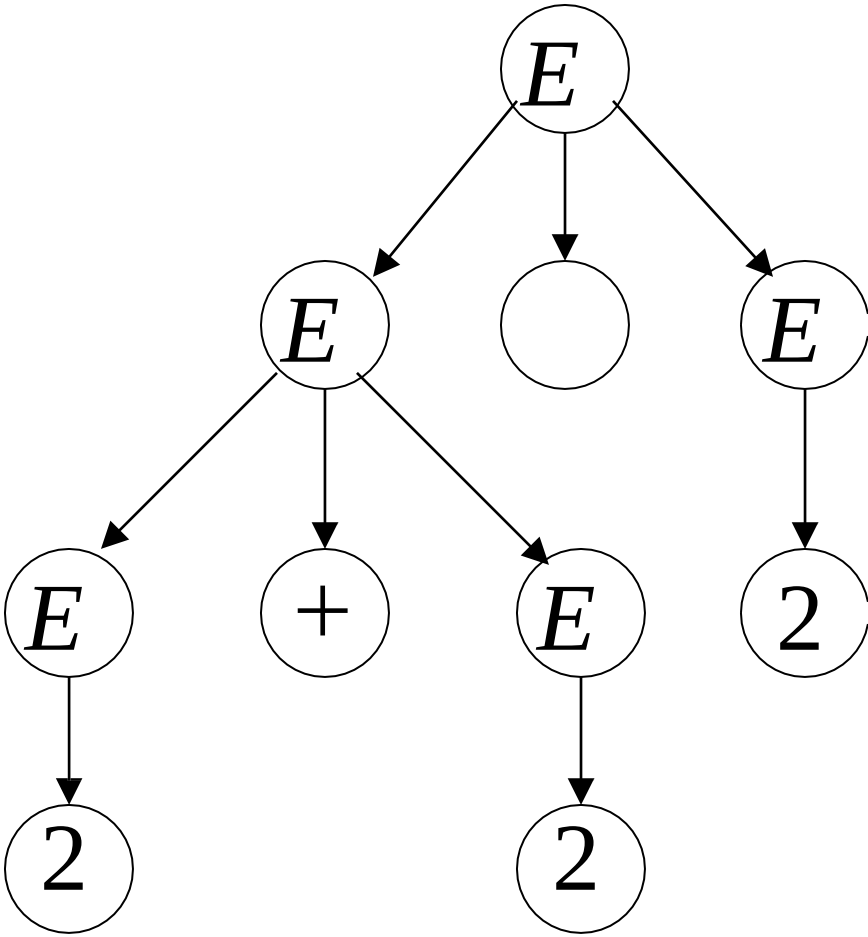




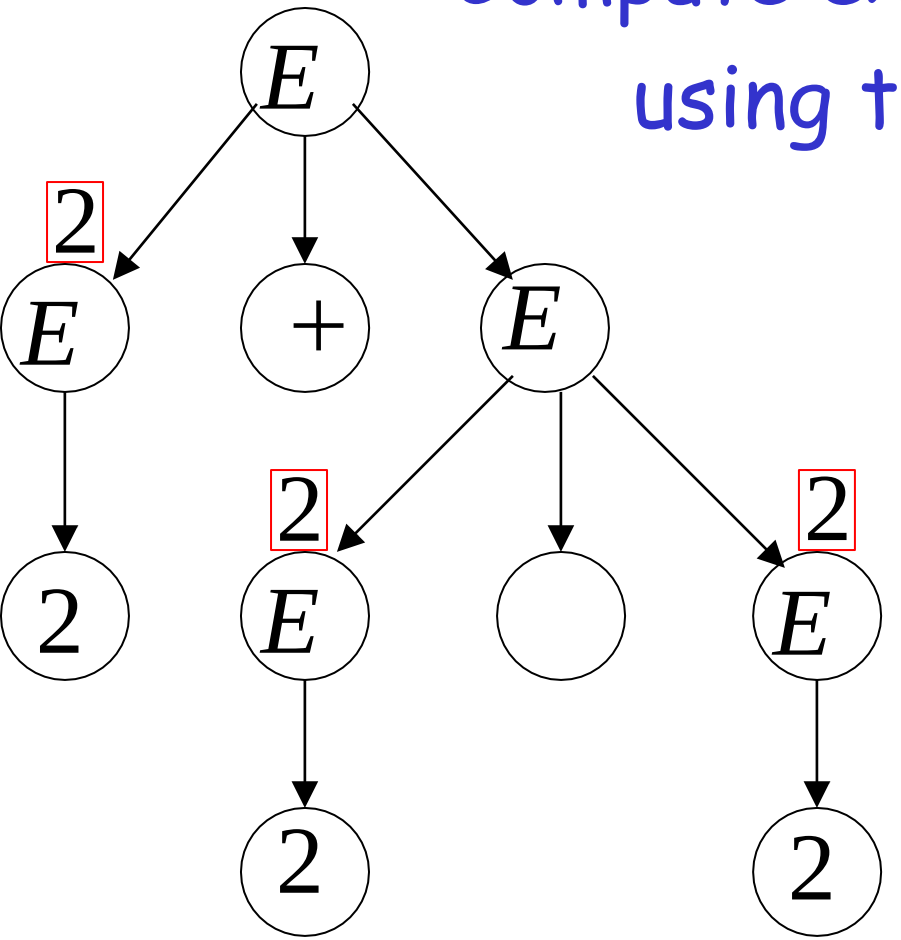
take $a = 2$

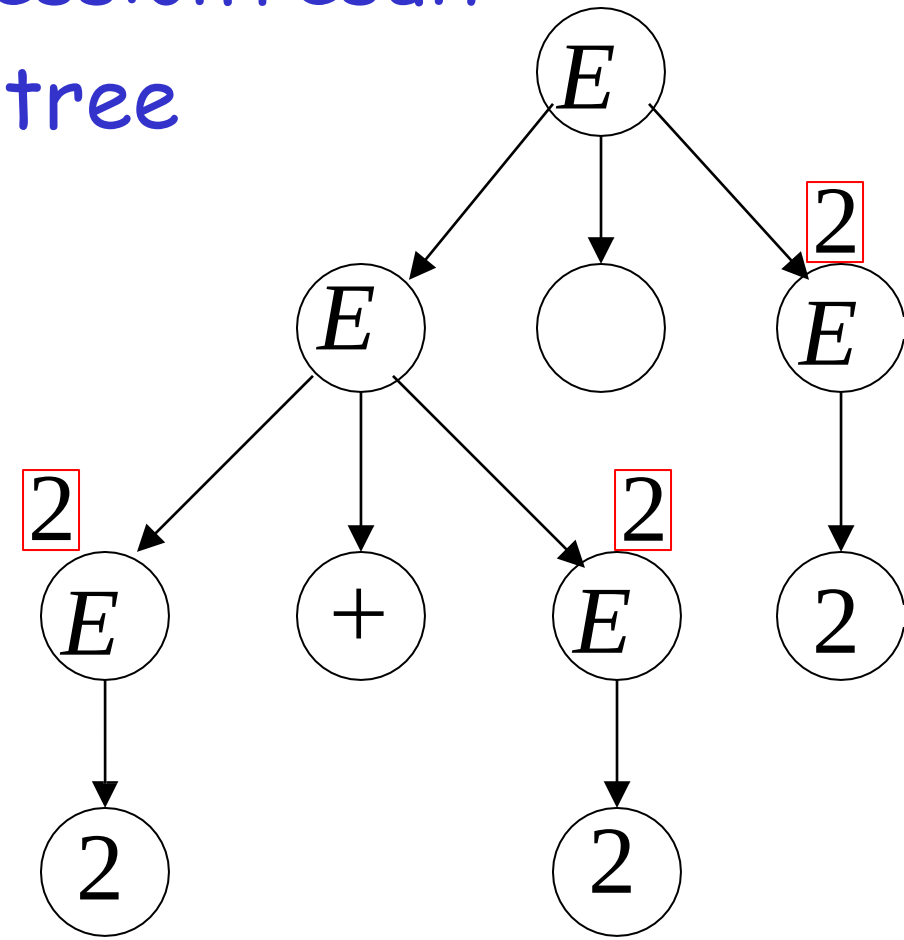
$a + a * a = 2 + 2 * 2$



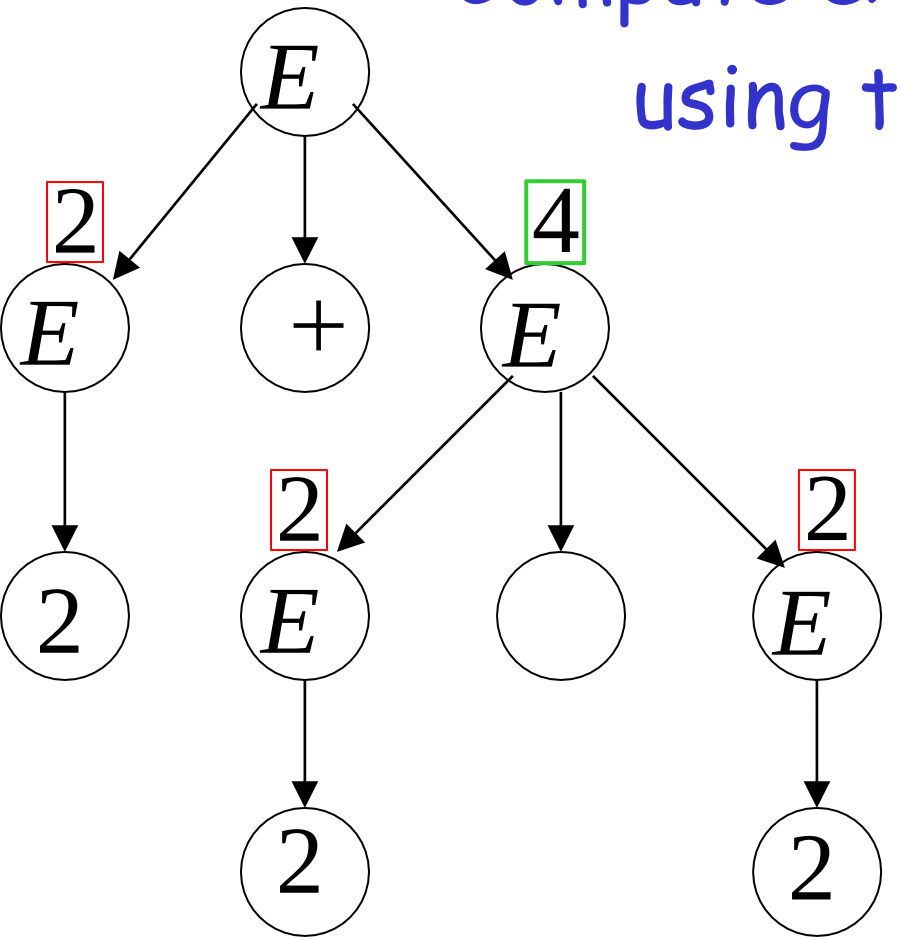


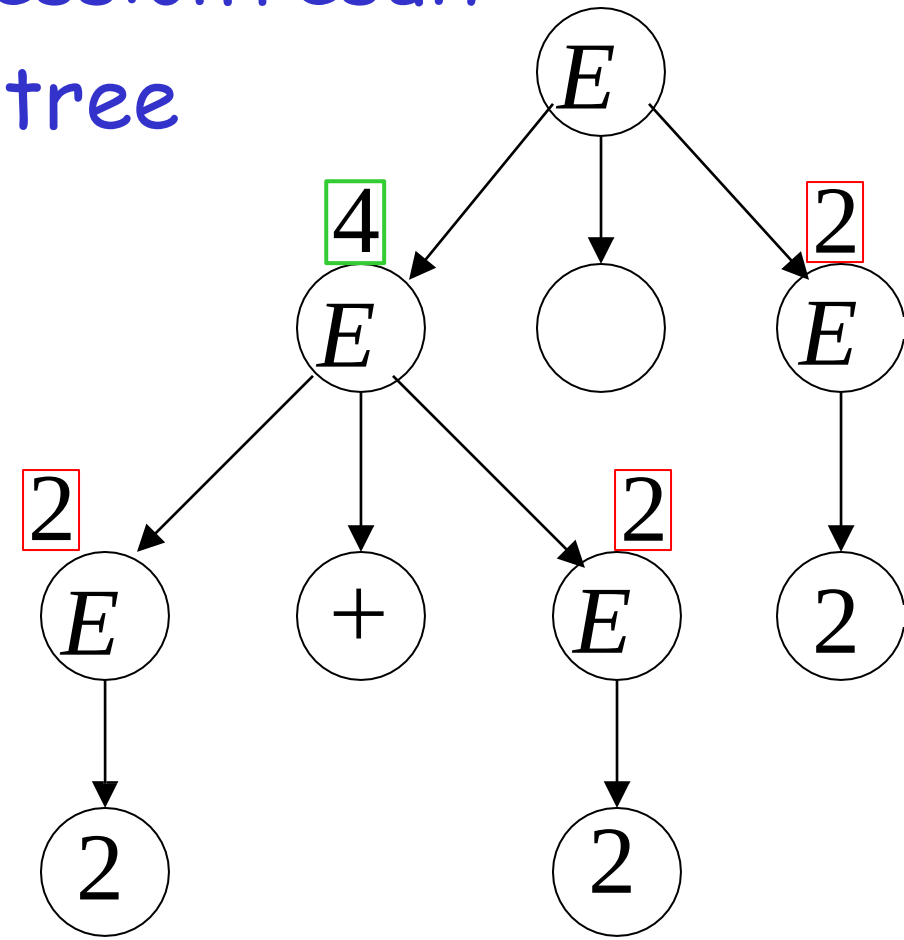
Compute expression result using the tree



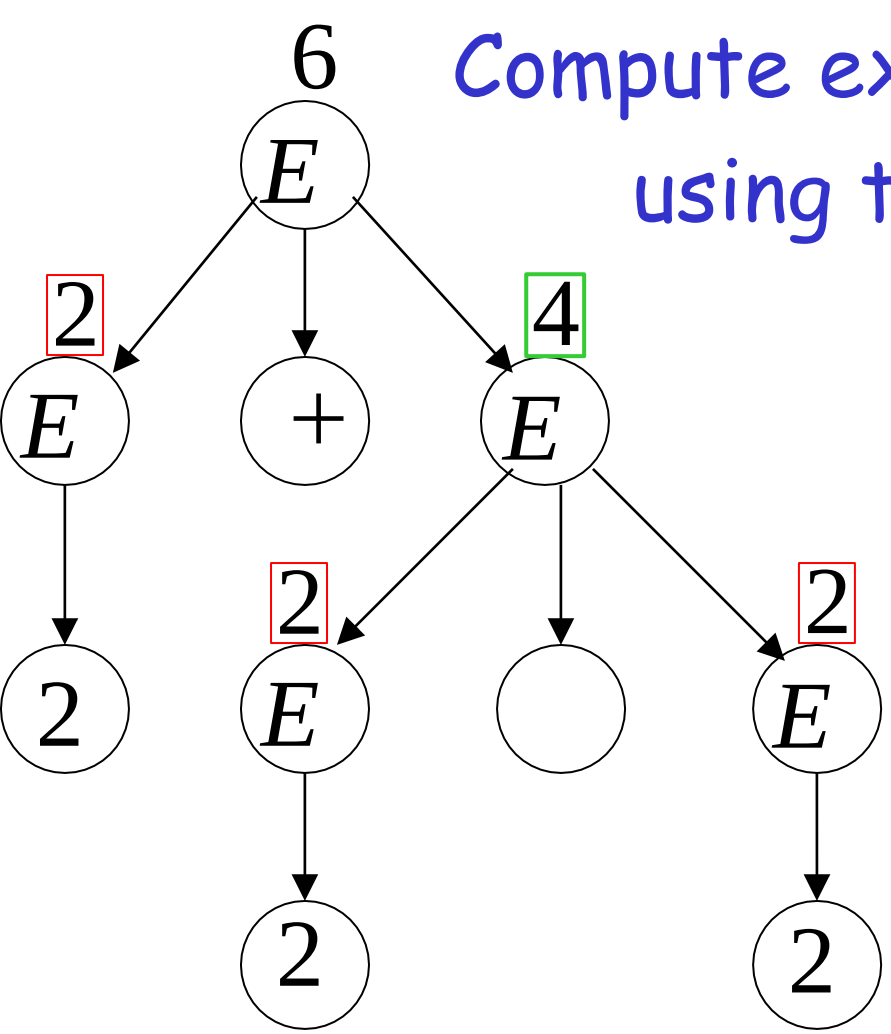


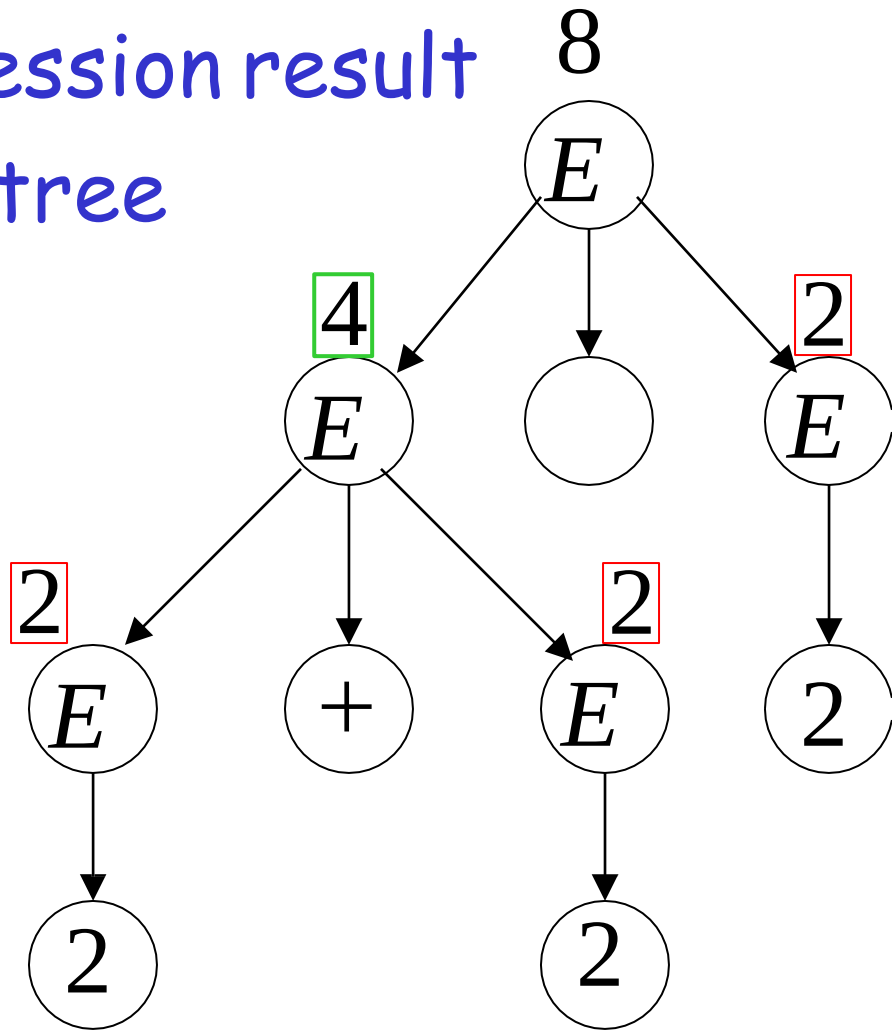
Compute expression result using the tree





Compute expression result using the tree





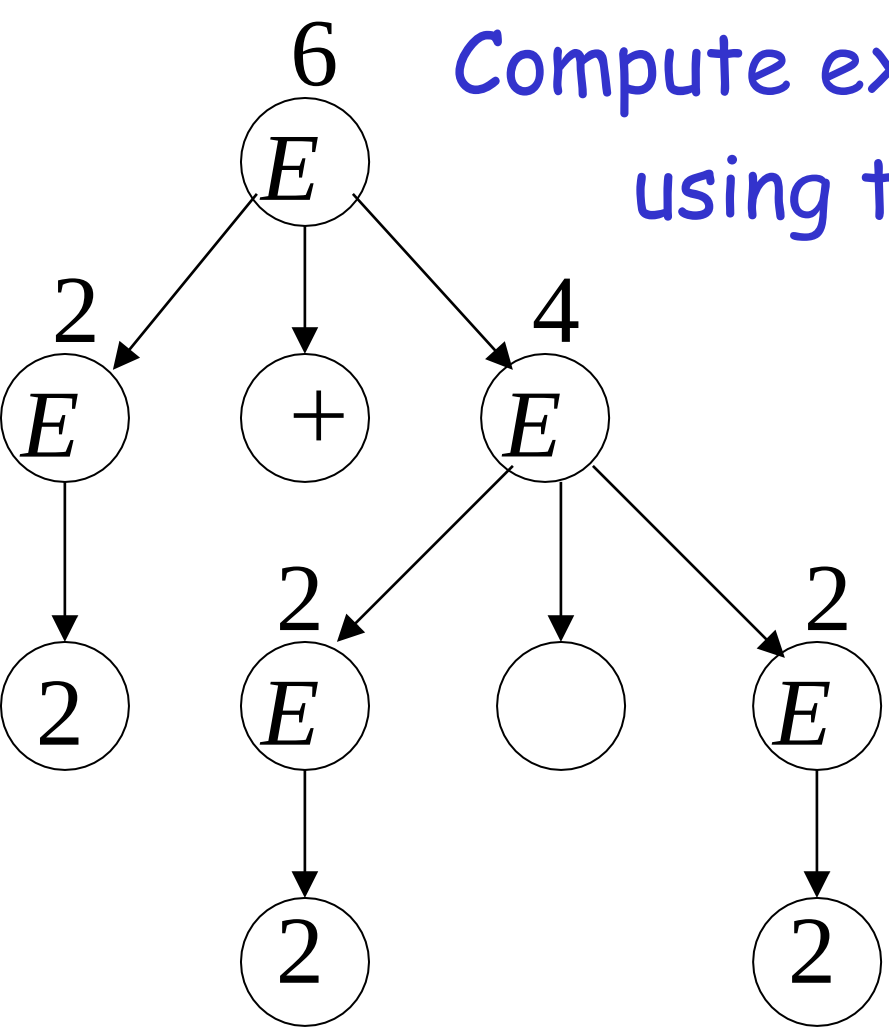
Good Tree

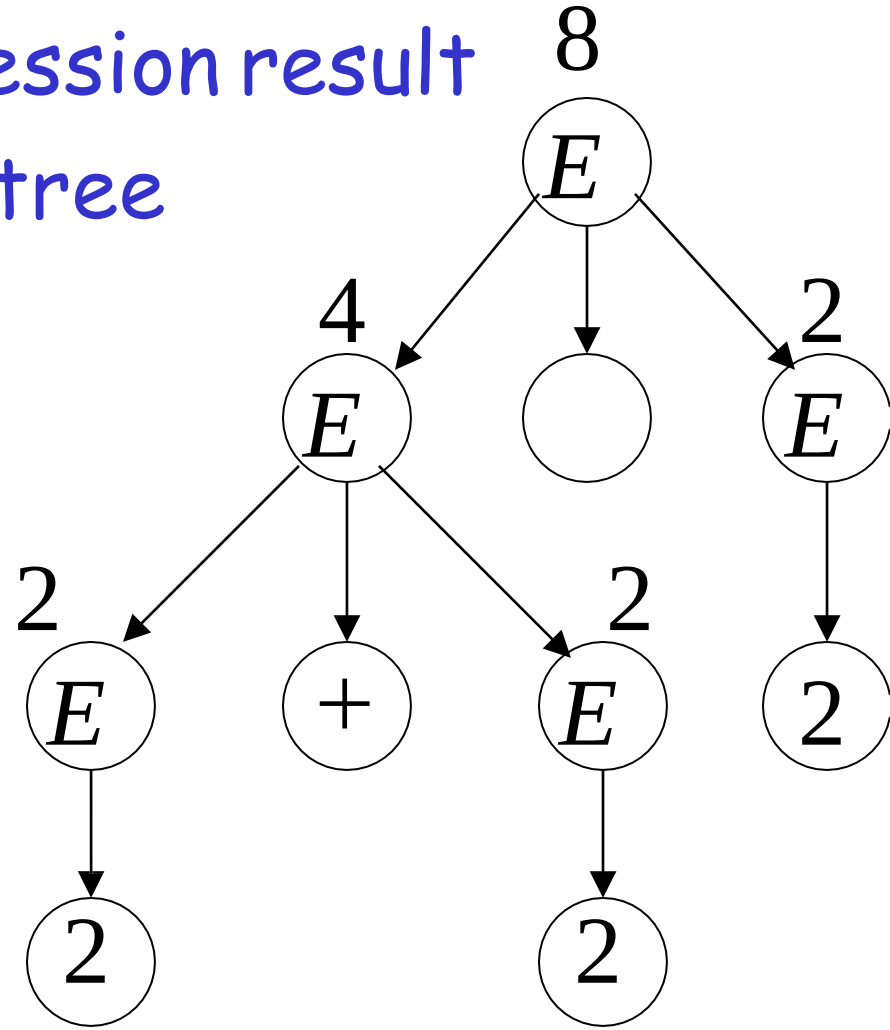
$2+2*2=6$

Bad Tree

$2+2*2=8$

Compute expression result using the tree





Q. Show that the following grammar is ambiguous:

$S \rightarrow a \mid a\ b\ S\ b \mid a\ A\ b$

$A \rightarrow b\ S \mid a\ A\ A\ b$

Two different derivation trees may cause problems in applications which use the derivation trees:

- Evaluating expressions
- In general, in compilers for programming languages

- Ambiguous grammars should be avoided
  - Do not guarantee unique parsing and translation
  - Expression evaluation is not clearly defined in the above grammar

Removal of Ambiguity

Ambiguous Grammar:

A context-free grammar G is ambiguous if there is a string $w \in L(G)$ which has:

two different derivation trees

or

two leftmost derivations

or

two rightmost derivations

(Two different derivation trees give two different leftmost derivations and vice-versa)

In general, ambiguity is bad
and we want to remove it

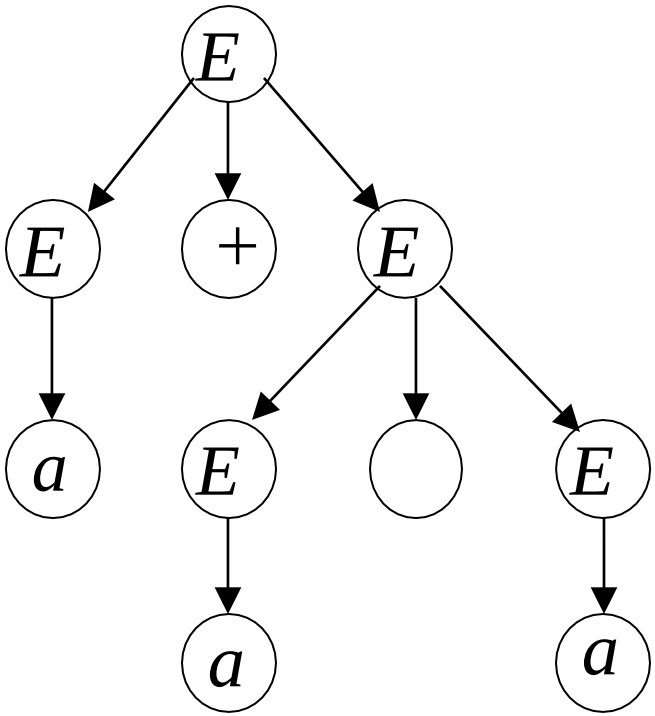
Sometimes it is possible to find
a non-ambiguous grammar for a language

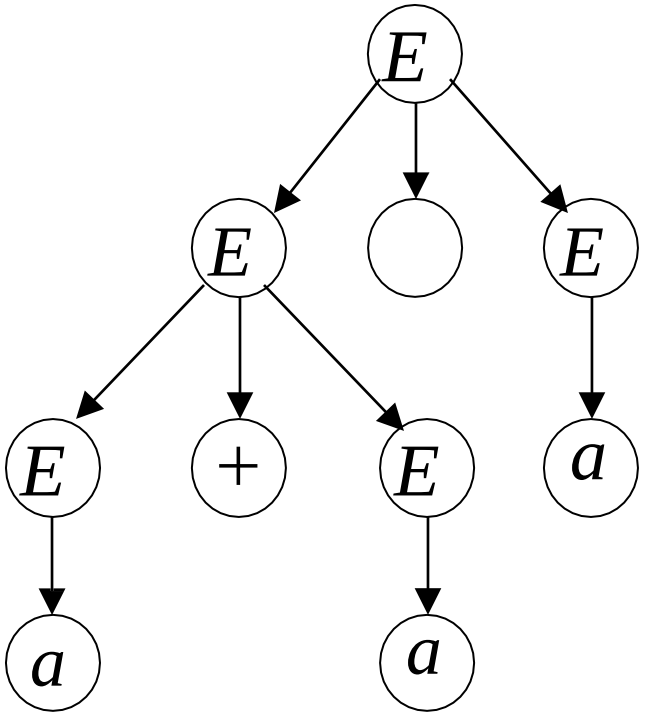
But, in general we cannot do so

Example: $E \rightarrow E+E \mid E*E \mid (E) \mid a$

this grammar is ambiguous since

string $a+a*a$ has two derivation trees





$$E \to E + E \mid E * E \mid (E) \mid a$$

this grammar is ambiguous also because

string $a + a * a$ has two leftmost derivations

$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E$$
$$\Rightarrow a + a * E \Rightarrow a + a * a$$

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E$$
$$\Rightarrow a + a * E \Rightarrow a + a * a$$

Recall

- Leftmost Derivation
- Rightmost Derivation
- Derivation Tree/Parse Tree
- Ambiguity

Eliminating Ambiguity in Expressions

- To guarantee unique translation, ambiguity should be eliminated
  - Unfortunately, there is NO algorithm that detects ambiguity in any CFG
  - However, some classes of grammars can be shown to be unambiguous
- To handle ambiguity in expressions …
  - The **precedence** and **associativity** of operators specify order of evaluation
  - Higher precedence operators are evaluated first
  - Equal precedence operators are evaluated according to associativity
    - Left-to-right or Right-to-left
- To handle precedence of operators …
  - We divide operators into groups of equal precedence
  - For each precedence level, we introduce a nonterminal and grammar rules
- To handle associativity of operators …
  - We design grammar rules to be either left or right recursive

Removing Ambiguity From Grammars

- Good news: Sometimes we can remove ambiguity "by hand"
- Bad news: There is no algorithm to do it. Solve by case to case

- There are two problems with above example:

1. There is no precedence between * and +

2. There is no grouping of sequences of operators, e.g. is E + E + E meant to be E +(E +E) or (E +E)+E.

Example: $E \rightarrow E+E \mid E*E \mid (E) \mid a$

- Solution:-

1.Precedenceis introduced by adding new non-terminal symbols.

2. Cascade symbols with lowestprecedence closest to the start symbol.

- E⓻ E+T|T
- T⓻ T*F|F
- F⓻ (E)|a

E ⓻ E +E

E ⓻ E

*E E

⓻ (E)

E ⓻ a

A successful example:

Ambiguous Grammar

$$E \rightarrow E + E$$
$$E \rightarrow E * E \quad E$$
$$\rightarrow (E)$$
$$E \rightarrow a$$

Equivalent Non-Ambiguous Grammar

$$E \rightarrow E + T \mid T \quad T$$
$$\rightarrow T * F \mid F \quad F \rightarrow$$
$$(E) \mid a$$

generates the same language

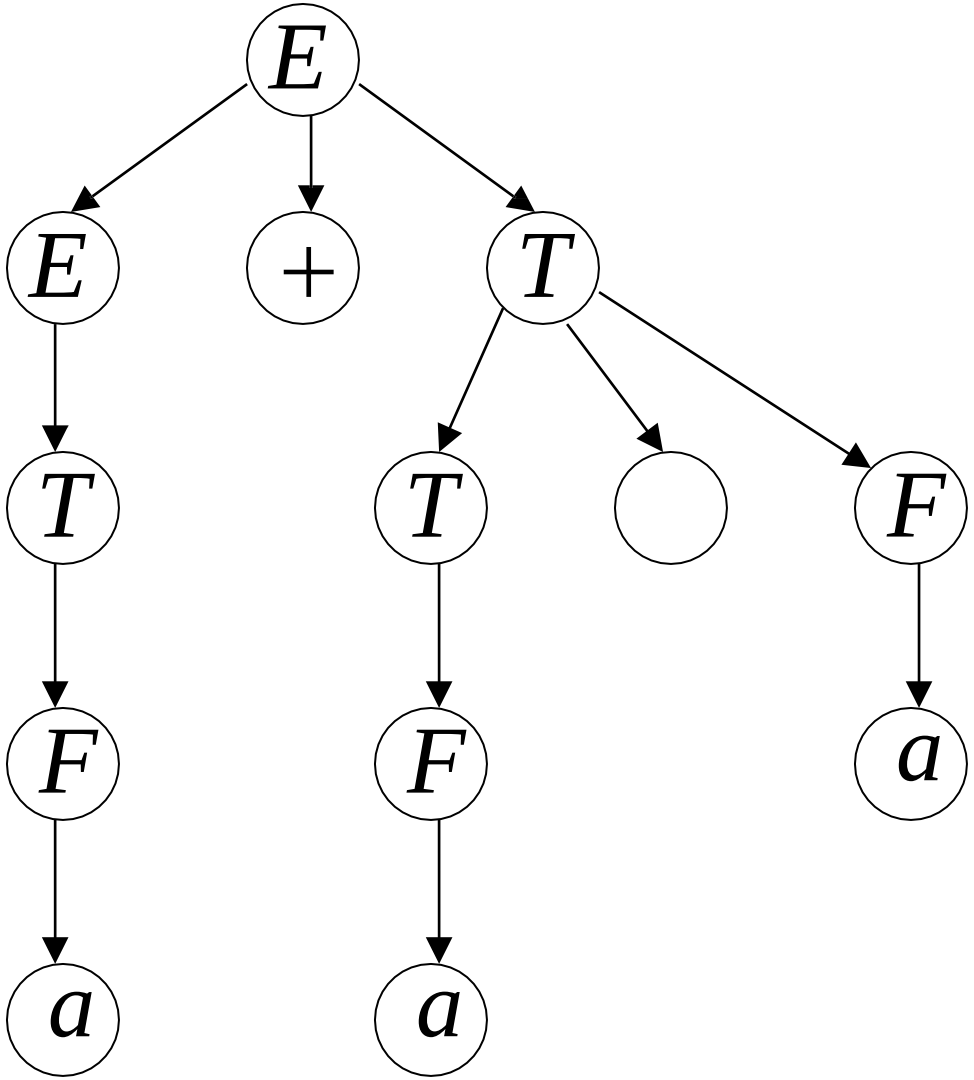
$$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow a+T \Rightarrow a+T*F$$

$$\Rightarrow a+F*F \Rightarrow a+a*F \Rightarrow a+a*a$$

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

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

$$F \rightarrow (E) \mid a$$



Unique derivation tree for $a+a*a$

Problem

Check whether or not the following grammar is ambiguous : if it is ambiguous, remove the ambiguity and write an equivalent unambiguous grammar

$$E \rightarrow E + E | E - E \mid E * E \mid E / E \mid (E) | id$$

(May-2018, 6 marks)

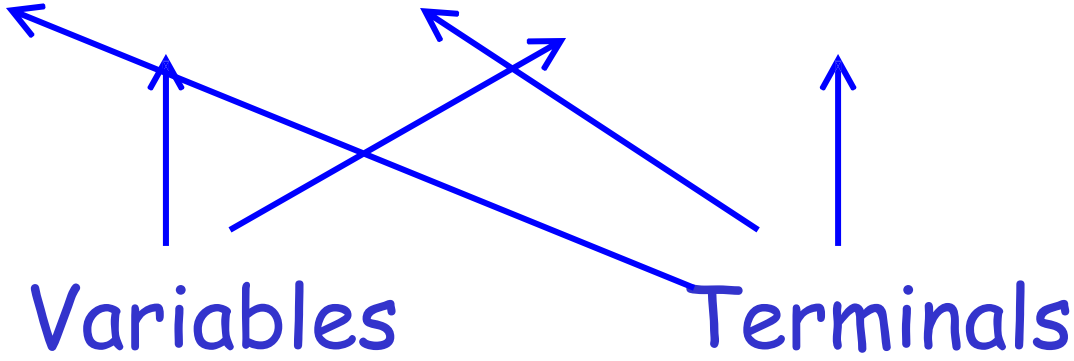
Recall

- Ambiguity
- Removal of Ambiguity

Another grammar:

STMT → if EXPR then STMT

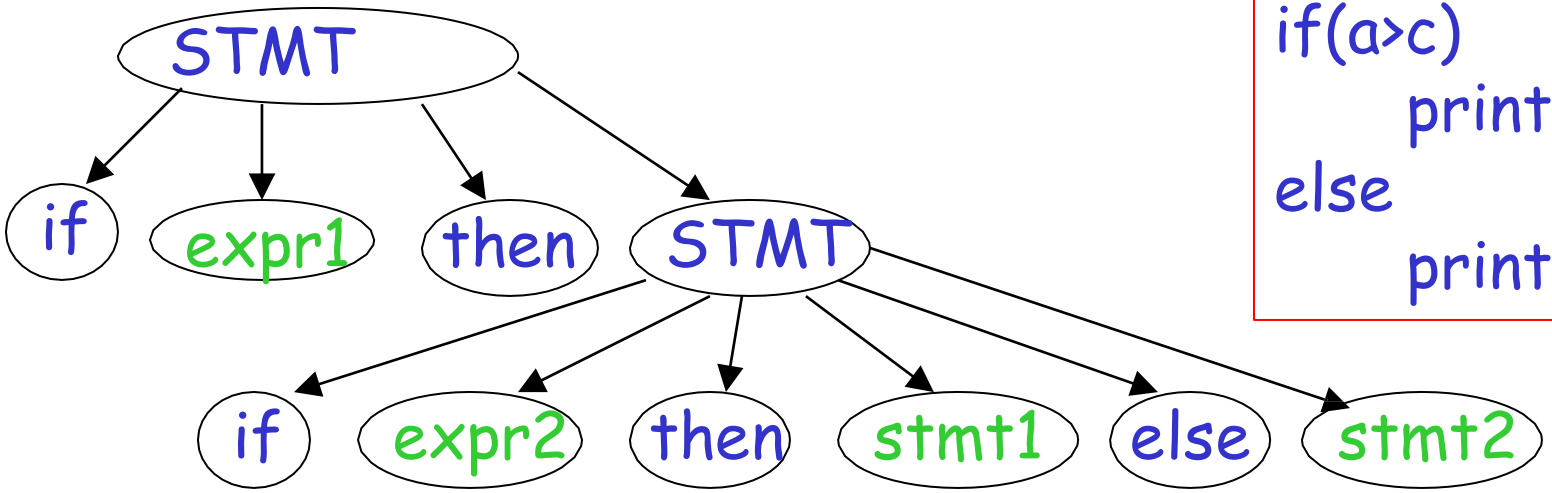
| if EXPR then STMT else STMT



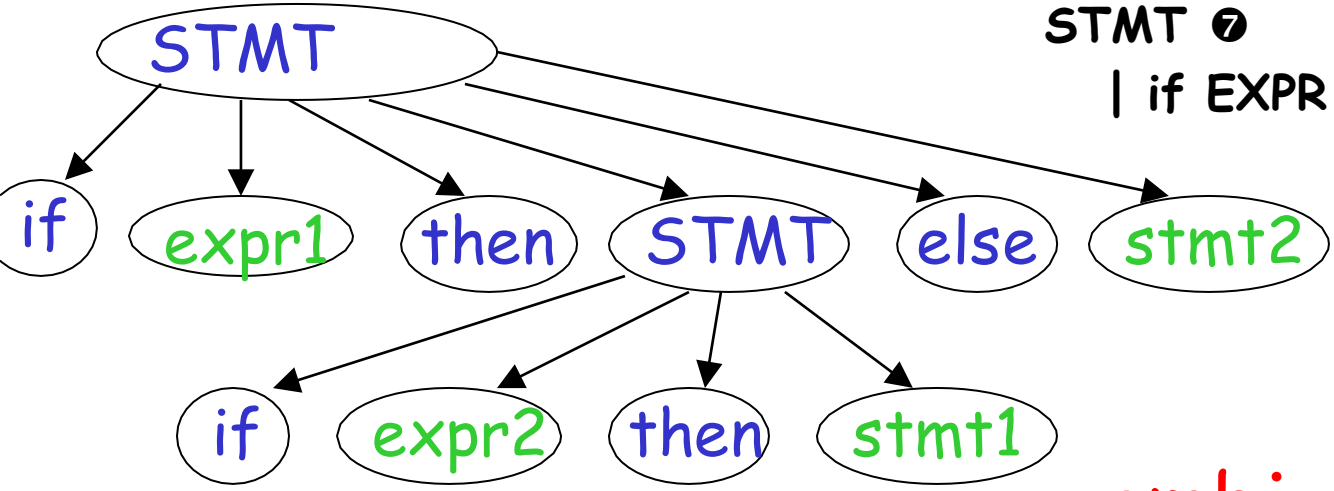
Very common/popular piece of grammar in programming languages

If expr1 then if expr2 then stmt1 else stmt2

```
if(a>b)
if(a>c)
        print a is large
else
        print c is large
```



Two derivation trees



STMT ❼ if EXPR then STMT
| if EXPR then STMT else STMT

ambiguous Grammer

Problem:

S➐ iEtS|iEtSe

S|a E➐ b

- Solution:-
- S ➐ S1 | S2
- S 1 ➐ iEtS1eS1
- S 2 ➐ iEtS1eS2|iEtS

- Solution:-
- S ➐ S1 | S2
- S 1 ➐ iEtS1eS1 | Astmt
- S 2 ➐ iEtS1eS2|iEtS
- Astmt-> any sentence

Problem

Check whether or not the following grammar is ambiguous; if it is ambiguous, remove the ambiguity and write an equivalent unambiguous grammar.

$S \rightarrow aS \mid aSbS \mid \varepsilon$

(Nov-2017, 6 marks)

S -> a S | S1 S | epsilon
S1 -> a S1 S1 b | epsilon

Simplifications of Context-Free Grammars

CFG Simplification

Can't always eliminate ambiguity.

But, CFG simplification & restriction still useful theoretically & pragmatically.

- Simpler grammars are easier to understand.
- Simpler grammars can lead to faster parsing.
- Restricted forms useful for some parsing algorithms.
- Restricted forms can give you more knowledge about derivations.

Motivations for Transforming CFG

- In many instances it is desirable to place restrictions on the grammars - transform the grammar in certain form
  - **Convenience of making arguments in theorem proof**
  - **Easier applications to parsing**

Recall

- Ambiguity and Removal of Ambiguity
- Simplifications of Context Free Grammar

Simplifications of Context-Free Grammars

Methods for Transforming CFG

- Simplification = removal of certain types of undesirable productions
- Restrict our discussion to ε-free languages – make our discussion easier
- Generating equivalent grammars = applying substitutions

A Substitution Rule

Equivalent grammar

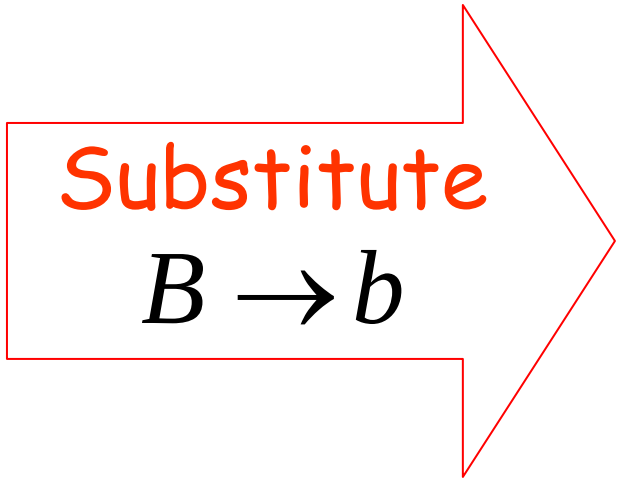
$S \to aB$

$A \to aaA$

$A \to abBc$

$B \to aA$

$B \to b$



$S \to aB \mid ab$

$A \to aaA$

$A \to abBc \mid abbc$

$B \to aA$

$$S \to aB \mid ab$$

$$A \to aaA$$

$$A \to abBc \mid abbc$$

$$B \to aA$$

Substitute

$$B \to aA$$

$$S \to aB \mid ab \mid aaA$$

$$A \to aaA$$

$$A \to abBc \mid abbc \mid abaAc$$

Equivalent grammar

In general: $A \to xBz$

$B \to y_1$

Substitute

$B \to y_1$

$A \to xBz \mid xy_1z$

equivalent grammar

Nullable Variables/Null production

λ–production: $X \to \lambda$

Nullable Variable: $Y \Rightarrow \square \Rightarrow \lambda$

Example: $S \to aMb$

$M \to aMb$

$M \to \lambda$

Nullable variable

λ–production

Removing $\lambda-$productions

$S \to aMb$

$M \to aMb$

~~$M \to \lambda$~~

Substitute $M \to \lambda$

$S \to aMb \mid ab$

$M \to aMb \mid ab$

After we remove all the $\lambda-$productions

all the nullable variables disappear

(except for the start variable)

Problem for Removing λ – productions

$S \rightarrow aA$

$A \rightarrow \varepsilon$

S ❼ AB

aC, A

❼ BC

B ❼ b

ε, C

❼

S ❼ ASA |

aB A

❼

B | S

B ❼ b | ε

S ❼ *A* |

BA ❼

aBa |

ε

B ❼

Unit-Productions

Unit Production: $X \to Y$

(a single variable in both sides)

Example:

$S \to aA$

$A \to a$

$A \to B$

$B \to A$

Unit Productions

$B \to bb$

Eliminate Variable Unit Productions

- A unit production is where RHS has only one symbol.
- Consider production A ➧ B . Then for every production B ➧ α , add the production A ➧ α.
- Repeat until done (but don't re-create a unit production already deleted)

Removal of unit productions:

$S \rightarrow aA$

$A \rightarrow a$

~~$A \rightarrow B$~~

$B \rightarrow A$

$B \rightarrow bb$

Substitute

$A \rightarrow B$

$S \rightarrow aA$

$A \rightarrow a \mid A \mid bb$

$B \rightarrow A$

$B \rightarrow bb$

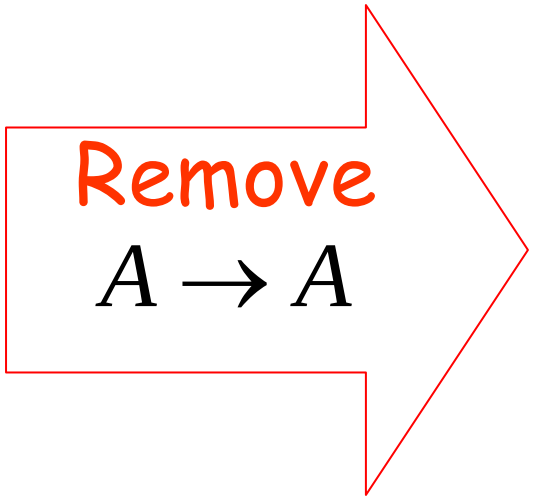
Unit productions of form $X \to X$ can be removed immediately

$S \to aA$

$A \to a \mid A \mid bb$

$B \to A$

$B \to bb$



$S \to aA$

$A \to a \mid bb$

$B \to A$

$B \to bb$

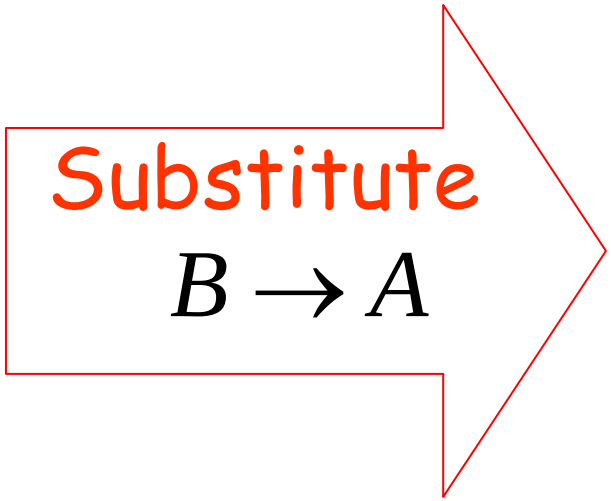
Removal of unit productions:

$S \rightarrow aA$

$A \rightarrow a \mid bb$

~~$B \rightarrow A$~~

$B \rightarrow bb$



$S \rightarrow aA$

$A \rightarrow a \mid bb$

$B \rightarrow a \mid bb$

Removal of unit productions:

S ➆ aXbX | abX | aXb |

ab X ➆ a Y | bY | a | b

Y ➆ X | c Y → X

S ➆ aXbX | abX | aXb |

ab X ➆ a Y | bY | a | b

Y ➆ c | aY | bY | a | b

Problem for Removing unit production

$S \rightarrow A \mid b$

$A \rightarrow B \mid b$

$B \rightarrow bB \mid a$

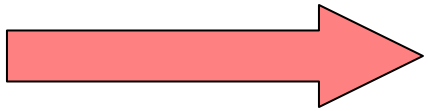
S ➐ ASA | aB | a | SA |

~~S~~A ➐ B | S

B ➐ b

Remove Useless productions

$$S \rightarrow aA$$
$$A \rightarrow a \mid bb$$
$$B \rightarrow a \mid bb$$



Final grammar

$$S \rightarrow aA$$
$$A \rightarrow a \mid bb$$

Useless Productions

$$S \to aSb$$

$$S \to \lambda S$$

$$\to A$$

$$A \to aA$$ Useless Production

Some derivations never terminate...

$$S \Rightarrow A \Rightarrow aA \Rightarrow aaA \Rightarrow \square \Rightarrow aa\square aA \Rightarrow [$$

Another grammar:

$$S \to A$$

$$A \to aA$$

$$A \to \lambda$$

$$B \to bA$$

Useless Production

Not reachable from S

In general:

If there is a derivation

$$S \Rightarrow \square \Rightarrow xAy \Rightarrow \square \Rightarrow w \in L(G)$$

consists of terminals

Then variable A is useful

Otherwise, variable A is useless

Recall

- Simplifications of Context Free Grammar
- Null variable and Null Production
- Unit production
- Useless Variable

A production $A \to x$ is useless
if any of its variables is useless

$S \to aSb$

$S \to \lambda$

Productions

Variables

$S \to A$ useless

useless $A \to aA$ useless

useless $B \to C$ useless

useless $C \to D$ useless

Removing Useless Variables and Productions

Example Grammar:

$$S \to aS \mid A \mid C$$

$$A \to a$$

$$B \to aa$$

$$C \to aCb$$

First: find all variables that can produce strings with only terminals or (possible useful variables)

$S \to aS \mid A \mid C$

$A \to a$

$B \to aa$

$C \to aCb$

Round 1: $\{A, B\}$

(the right hand side of production that has only terminals)

Round 2: $\{A, B, S\}$

(the right hand side of a production has terminals and variables of previous round)

This process can be generalized

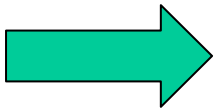
Then, remove productions that use variables other than $\{A, B, S\}$

$S \to aS \mid A \mid C$

$A \to a$

$B \to aa$

$C \to aCb$



$S \to aS \mid A$

$A \to a$

$B \to aa$

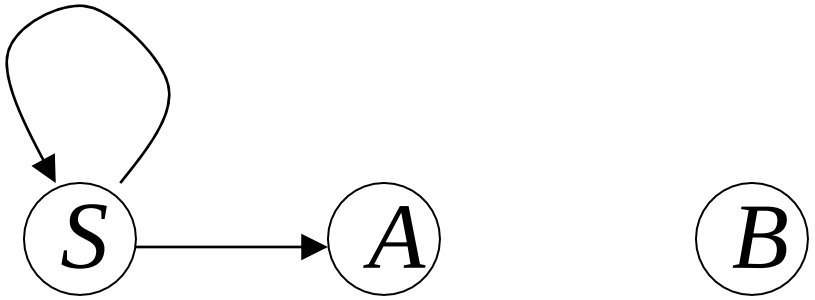
Second: Find all variables reachable from S

Use a Dependency Graph where nodes are variables

$$S \to aS \mid A$$

$$A \to a$$

$$B \to aa$$



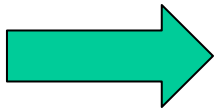
unreachable

Keep only the variables reachable from S

$S \rightarrow aS \mid A$

$A \rightarrow a$

~~$B \rightarrow aa$~~



Final Grammar

$S \rightarrow aS \mid A$

$A \rightarrow a$

Contains only useful variables

Removing All

Step 1: Remove Nullable Variables

Step 2: Remove Unit-Productions

Step 3: Remove Useless Variables

This sequence guarantees that unwanted variables and productions are removed

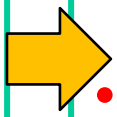
Simplification of grammers

Eliminating Null ε-Productions

- A ➔ C D
- B ➔ C b
- C ➔ a | ε
- D ➔ b D | ε

- A ➔ C D
- B ➔ C b
- C ➔ a | ε
- D ➔ bD | ε

- A ➔ CD | C | D
- B ➔ C b
- C ➔ a | ε
- D ➔ bD | ε



- A ➔ CD | C | D | ε
- B ➔ C b | b
- C ➔ a | ε
- D ➔ b D | ε

- A ➔ CD | C | D | ε
- B ➔ C b | b
- C ➔ a
- D ➔ b D | b

Eliminating Unit Productions

- S → S+T | T
- T → T*F | F
- F → (S) | a

S → S+T | T

T → T*

F | F

F → (S)

S → S+T | T*F | (S) | a

F

T → T*

F | F

S → S+T | T*F | (S) | a

T → T*F | (S) | a

F → (S) | a

S → S+T | T*F | (S) | a

T → T*F | (S) | a

F → (S) | a

Eliminating Unit-Productions

- A ➡ CD|C|D
- B ➡ C b | b
- C ➡ a
- D ➡ bD|b

- A ➡ CD|C|D
- B ➡ C b | b
- C ➡ a
- D ➡ bD|b

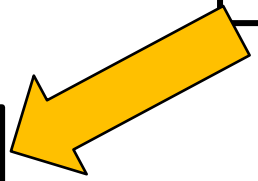
- A ➡ CD|a|D
- B ➡ Cb|b
- C ➡ a
- D ➡ bD|b

- A ➡ CD|a|bD|b
- B ➡ C b | b
- C ➡ a
- D ➡ b D | b

Eliminating Null ε-Productions

- $S \rightarrow ASA \mid aB$
- $A \rightarrow B \mid S$
- $B \rightarrow b \mid \varepsilon$

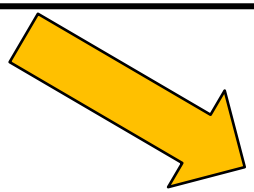
- $S \rightarrow ASA \mid aB$
- $A \rightarrow B \mid S$
- $B \rightarrow b \mid \varepsilon$



$S \rightarrow ASA \mid aB \mid a$

$A \rightarrow B|S|\varepsilon$

$B \rightarrow b$



$S \rightarrow ASA \mid aB|a|AS|SA|S$

$A \rightarrow B|S$

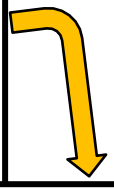
$B \rightarrow b$

Eliminating Unit-Productions

$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid S$

$A \rightarrow B \mid S$

$B \rightarrow b$



$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

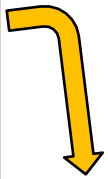
$A \rightarrow B \mid S$

$B \rightarrow b$

$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$A \rightarrow b \mid S$

$B \rightarrow b$



$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$A \rightarrow b \mid ASA \mid aB \mid a \mid AS \mid SA$

$B \rightarrow b$

Recall

- Simplifications of Context Free Grammar
- Null variable and Null Production
- Unit production
- Useless Variable

Noam Chomsky (from Wikipedia)

An American linguist, philosopher, cognitive scientist, political activist, author, and lecturer. He is an Institute Professor and professor emeritus of linguistics at the Massachusetts Institute of Technology.

Chomsky is well known in the academic and scientific community as one of the fathers of modern linguistics. Since the 1960s, he has become known more widely as a political dissident, an anarchist, and a libertarian socialist intellectual. Chomsky is often viewed as a notable figure in contemporary philosophy.

Normal Forms
for
Context-free Grammars

Normal Forms

- It is Convenient to simplify CFG to remove redundant variables without changing the language of grammars.

- Chomsky Normal Form

- Greibach Normal Form

Chomsky Normal Form

Each productions has form:

$A \to BC$ or $A \to a$

Examples:

$$S \to AS$$
$$S \to a$$
$$A \to SA$$
$$A \to b$$

Chomsky Normal Form

$$S \to AS$$
$$S \to AAS$$
$$A \to SA$$
$$A \to aa$$

Not Chomsky Normal Form

Conversion to Chomsky Normal Form

Example:

$$S \rightarrow ABa$$

$$A \rightarrow aab$$

$$B \rightarrow Ac$$

Not Chomsky Normal Form

We will convert it to Chomsky Normal Form

Introduce new variables for the terminals:

$$T_a, T_b, T_c$$

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

$$S \to ABT_a$$

$$A \to T_a T_a T_b$$

$$B \to AT_c$$

$$T_a \to a$$

$$T_b \to b$$

$$T_c \to c$$

Introduce new intermediate variable V_1 to break first production:

$$S \to ABT_a$$
$$A \to T_a T_a T_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_a T_a T_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Introduce intermediate variable: V_2

$S \to AV_1$

$V_1 \to BT_a$

$A \to T_a T_a T_b$

$B \to AT_c$

$T_a \to a$

$T_b \to b$

$T_c \to c$

$S \to AV_1$

$V_1 \to BT_a$

$A \to T_a V_2$

$V_2 \to T_a T_b$

$B \to AT_c$

$T_a \to a$

$T_b \to b$

$T_c \to c$

Final grammar in Chomsky Normal Form:

Initial grammar

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

$$S \to AV_1$$

$$V_1 \to BT_a$$

$$A \to T_aV_2$$

$$V_2 \to T_aT_b$$

$$B \to AT_c$$

$$T_a \to a$$

$$T_b \to b$$

$$T_c \to c$$

Each productions has form:

$A \to BC$ or $A \to a$

variable variable terminal

In general:

From any context-free grammar
(which doesn't produce)
not in Chomsky Normal Form

we can obtain:

an equivalent grammar
in Chomsky Normal Form

The Procedure

First remove:

Nullable variables

Unit productions

(Useless variables optional)

Then, for every symbol/terminal a :

New variable: T_a

Add production $T_a \to a$

In productions with length at least 2
replace a with T_a

Productions of form $A \to a$
do not need to change!

Replace any production $A \to C_1 C_2 \square C_n$

with $A \to C_1 V_1$

$V_1 \to C_2 V_2$

$\square$

$V_{n-2} \to C_{n-1} C_n$

New intermediate variables: $V_1, V_2, \square, V_{n-2}$

Observations

- Chomsky normal forms are good for parsing and proving theorems

- It is easy to find the Chomsky normal form for any context-free grammar

Converting Grammer to the CNF

- S ➆A A C D
- A ➆a A b | ε
- C ➆a C | a
- D ➆aDa|bDb|
  ε

1.Eliminating Null Productions:-

The Null Variables are A and D

Converting Grammer to the CNF

- S ➆A A C D
- A ➆a A b | ε
- C ➆a C | a
- D ➆a D a | b D b | ε

- S ➆A A C D
- A ➆a A b | ε
- C ➆a C | a
- D➆ aDa|bDb|
  ε

- S ➆ A A C D
- A ➆ a A b | ε
- C ➆ a C | a
- D ➆ aDa|bDb|ε

- S➆AACD|ACD|AAC|CD|AC|C
- A ➆a A b | ε
- C ➆a C | a
- D➆ aDa|bDb|ε

S ➆ AACD|ACD|AAC|CD|AC|C
A ➆ aAb|ab
C ➆ aC|a
D ➆ aDa|bDb
|ε

- S ➆ AACD|ACD|AAC|CD|AC|C
- A ➆a A b | a b
- C ➆a C | a
- D ➆aDa|bDb|aa|bb

2.Eliminating Unit Productions:-

- S ➐ AACD|ACD|AAC|CD|AC|C
- A ➐a A b | a b
- C ➐a C | a
- D ➐a D a | b D b | a a | b b

- S ➐ AACD|ACD|AAC|CD|AC|aC|a
- A ➐a A b | a b
- C ➐a C | a
- D ➐a D a | b D b | a a | b b

3.Restrict RHS by single terminal or Two Variables only

- S ➔ AACD|ACD|AAC|CD|AC|aC|a
- A ➔a A b | a b
- C ➔a C | a
- D ➔aDa|bDb|aa|bb

3.1.Add New Variable for Each terminal

- S ➔ AACD|ACD|AAC|CD|AC|aC|a
- A ➔a A b | a b
- C ➔a C | a
- D ➔aDa|bDb|aa|bb
- X ➔ a
- Y ➔ b

2. In productions with length at least 2 replace terminal with corresponding variable and productions of the form $A \to a$ leave as it is.

- $S \to AACD|ACD|AAC|CD|AC|aC|a$
- $A \to aAb|ab$
- $C \to aC|a$
- $D \to aDa|bDb|aa|bb$
- $X \to a$
- $Y \to b$

- $S \to AACD|ACD|AAC|CD|AC|XC|a$
- $A \to XAY|XY$
- $C \to XC|a$
- $D \to XDX|YDY|XX|YY$
- $X \to a$
- $Y \to b$

4. The final Step:-Covert it into the form of Two variables on RHS (Already we got single terminal on RHS)

- S ➔ AACD|ACD|AAC|CD|AC|XC|a
- A ➔X A Y | X Y
- C ➔X C | a
- D ➔X D X | Y D Y | X X | Y Y
- X ➔ a
- Y ➔ b

- S ➆A V 1 , V1➆AV2,V2➆D
- S ➆A T 1 , T 1 ➆C D
- S ➆A U 1 , U 1 ➆A C
- S ➆C D | A C | X C | C
- A ➆X Z 1 , Z 1 ➆A Y
- A ➆X Y
- C ➆X C | a
- D ➆X W 1 , W 1 ➆D X
- D ➆Y M 1 , M 1 ➆D y
- D ➆X X | Y Y
- X ➆ a
- Y ➆ b

Recall

- Normal Forms
- CNF (Chomsky Normal Form)

Converting Grammer to the CNF

- $S \rightarrow ASA \mid aB$
- $A \rightarrow B \mid S$
- $B \rightarrow b \mid \varepsilon$

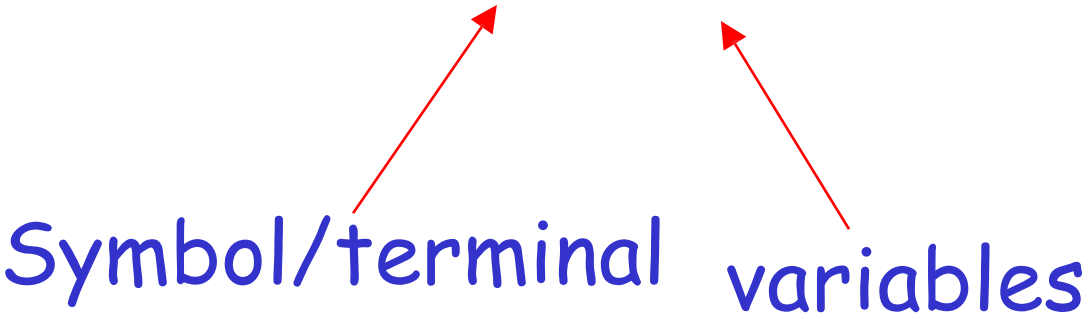
Recall

- Normal Forms
- CNF (Chomsky Normal Form)

Greinbach Normal Form

All productions have form:

$$A \to a\, V_1 V_2 \square V_k \qquad k \geq 0$$



- In Chomsky's Normal Form (CNF), restrictions are put on the length of right sides of a production,
- whereas in Greibach Normal Form (GNF), restrictions are put on the positions in which terminals and variables can appear.

Examples:

$$S \to cAB$$
$$A \to aA \mid bB \mid b$$
$$B \to b$$

Greinbach
Normal Form

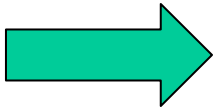
$$S \to abSb$$
$$S \to aa$$

Not Greinbach
Normal Form

Conversion to Greibach Normal Form:

$$S \to abSb$$

$$S \to aa$$



$$S \to aT_bST_b$$

$$S \to aT_a$$

$$T_a \to a$$

$$T_b \to b$$

Greibach Normal Form

Observations

- Greibach normal forms are very good for parsing strings (better than Chomsky Normal Forms)

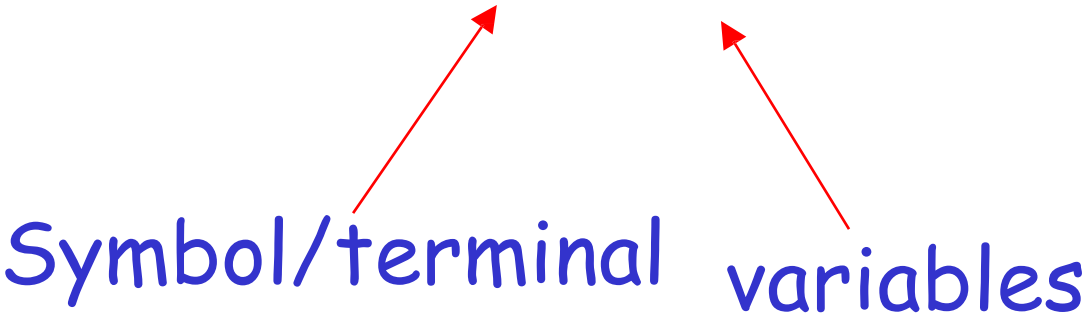
- However, it is difficult to find the Greibach normal of a grammar

- GNF is useful in simplifying some proofs and making constructions such as Push Down Automaton (PDA) accepting a CFG.

Greinbach Normal Form

All productions have form:

$$A \to a\, V_1 V_2 \square V_k \qquad k \geq 0$$



Procedure for conversion of CFG to GNF

- Firstly, we convert the given grammar into Chomsky Normal Form:
  - Start with a grammar G = (V, T, P, S)
  - Eliminate useless variables that cannot become terminals.

    Eliminate useless variables that cannot be reached.
  - Eliminate λ-productions. Eliminate unit productions.
  - Convert grammar to Chomsky Normal Form.
- Then, we convert this grammar into an equivalent grammar in Greibach NF.

Convert a CNF grammar into Greibach Normal Form:

1. Re-label all variables such that the names are $A_1, A_2, \ldots, A_n$.
2. We want to, give order(index) the productions, which are not terminal but contain variables. We use this for the purpose of indexing of the variables, so that

$$A_i \rightarrow A_j \alpha \text{ with } i<j \text{ for all } i = 1,\ldots,m \text{ and } j = 2, \ldots, m.$$

- We perform the ordering process by substitution of the first variable on the PRODUCTION, if the production violates the condition (see 2 and 3).

3. We start the ordering with A_1

A_1-productions can have only a higher numbered variable as first variable on the PRODUCTION, or a single terminal.

4. We assume now that all rules are okay up to A_{k-1}. The next rule we encounter, with A_k on the production, is the first one, which is not okay: $A_K \rightarrow A_l\ \alpha$ with $k>l$

We resolve this problem by substituting A_l. Since $l<k$, the A_l rules have already gone through the sorting process and are in the proper format:

- $A_l \rightarrow A_j\ \alpha$ with $l<j$

- Now, we substitute the PRODUCTIONs of A_l in the A_k-rule, and come up with:

  - $A_k \rightarrow A_l\ \alpha$ or $A_k \rightarrow a$ for some a

- If l is still less than k, we substitute again. And again and again, until we get at least A_k on the PRODUCTION - all rules up to A_{k-1} are already sorted and in proper form, and the first variable on the PRODUCTION of the A_{k-1} - production must be therefore at least A_k.
- If their present Left Recursion Remove the Left Recursion.

REMOVING LEFT RECURSION AND INDIRECT LEFT RECURSION

DEFINITIONS

IMMEDIATE LEFT RECURSION.

A production is immediately left recursive if its left hand side and the head of its right hand side are the same symbol,

e.g. $B \rightarrow Ba$

A grammar is called immediately left recursive if it possesses an immediately left recursive production.

INDIRECT LEFT RECURSION.

A grammar is said to posses indirect left recursion if it is possible, starting from any symbol of the grammar, to derive a string whose head is that symbol.

Example. $A \rightarrow Br$

$B \rightarrow Cs$

$C \rightarrow At$

Here, starting with A, we can derive Atsr

NOTE.

1. Immediate left recursion is a special case of indirect left recursion (in which the derivation involved is just a single step)
2. "Immediate left recursion" is conventionally referred to simply as "left recursion" (leaving out the word "immediate"). But, for our purposes, this is confusing, so we will not follow that practice.

To remove left recursion from G (i.e. produce an **equivalent** grammar with the same language as G, but which is not left recursive), do the following:

For each nonterminal A that occurs as the lhs of a left-recursive production of G,

do the following:

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

$$A \rightarrow \beta_1$$

.....

$$A \rightarrow \beta_r$$

$$A \rightarrow A\alpha_1 | \ldots\ldots | A\alpha_r | \beta_1 | \ldots\ldots | \beta_r$$

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

Replace the above productions by:

$$A \rightarrow \beta_1 | \ldots | \beta_s | \beta_1 K_A | \ldots | \beta_s K_A$$

$$K_A \rightarrow \alpha_1 | \ldots | \alpha_r | \alpha_1 K_A | \ldots | \alpha_r K_A$$

Clearly the grammar G' produced is equivalent to G

EXAMPLE.

$S \rightarrow R\ a \mid A\ a \mid a$

$R \rightarrow a\ b$

$A \rightarrow A\ R \mid A\ T \mid b$

$T \rightarrow T\ b \mid a$

A non-left recursive grammar equiv. to the above is:

$S \rightarrow R\ a \mid A\ a \mid a$

$R \rightarrow a\ b$

$A \rightarrow b \mid b\ K_A$

$K_A \rightarrow R \mid T \mid R\ K_A \mid T\ K_A$

$T \rightarrow a \mid a\ K_T$

$K_T \rightarrow b \mid b\ K_T$

Eliminate direct left recursion

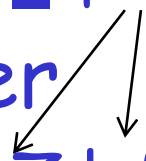
- Before

  $A \rightarrow Aa \mid b$

- After

  $A \rightarrow bZ \mid b$

  $Z \rightarrow aZ \mid a$



- Remove the rule with direct left recursion, and create a new one with recursion on the right

Eliminate direct left recursion

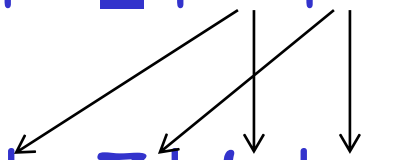
- Before

  $A \rightarrow Aa \mid Ab \mid b \mid c$

- After

  $A \rightarrow bZ \mid cZ \mid b \mid c$

  $Z \rightarrow aZ \mid bZ \mid a \mid b$



- Remove the rules with direct left recursion, and create new ones with recursion on the right

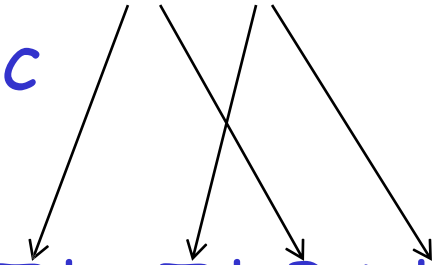
Eliminate direct left recursion

- Before

$A \rightarrow AB \mid BA \mid a$

$B \rightarrow b \mid c$

- After



$A \rightarrow BAZ \mid aZ \mid BA \mid a$

$Z \rightarrow BZ \mid B$

$B \rightarrow b \mid c$

Recall

- GNF (Greibach Normal Form)
- Left Recursion

Greibach Normal Form

Example:

$S \rightarrow XA \mid BB$

$B \rightarrow b \mid SB$

$X \rightarrow b$

$A \rightarrow a$

CNF

$S = A_1$

$X = A_2$

$A = A_3$

$B = A_4$

New Labels

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow b \mid A_1A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Updated CNF

Greibach Normal Form

Example:

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow b \mid A_1A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

First Step

$$A_i \rightarrow A_jX_k \quad j > i$$

X_k is a string of zero or more variables

✗ $A_4 \rightarrow A_1A_4$

Index value of left hand side variable must be less than the first variable from right hand side productions

Greibach Normal Form

Example:

First Step $A_i \rightarrow A_j X_k \quad j > i$

$A_4 \rightarrow A_1 A_4$

$A_4 \rightarrow A_2 A_3 A_4 \mid A_4 A_4 A_4 \mid b$

$A_4 \rightarrow b A_3 A_4 \mid A_4 A_4 A_4 \mid b$

$A_1 \rightarrow A_2 A_3 \mid A_4 A_4$

$A_4 \rightarrow b \mid A_1 A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Greibach Normal Form

Example:

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow bA_3A_4 \mid A_4A_4A_4 \mid b$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Second Step

Eliminate Left Recursions

✗ $A_4 \rightarrow A_4A_4A_4$

If Index value of left hand side variable and index of first variable from right hand side productions both are same.

Greibach Normal Form

Example:

Second Step

Eliminate Left Recursions

$A_4 \rightarrow bA_3A_4 \mid b \mid bA_3A_4Z \mid bZ$

$Z \rightarrow A_4A_4 \mid A_4A_4Z$

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow bA_3A_4 \mid A_4A_4A_4 \mid b$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Greibach Normal Form

Example:

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow bA_3A_4 \mid b \mid bA_3A_4Z \mid bZ$

$Z \rightarrow A_4A_4 \mid A_4A_4\,Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

$A \rightarrow aX$

GNF

Greibach Normal Form

Example:

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow bA_3A_4 \mid b \mid bA_3A_4Z \mid bZ$

$Z \rightarrow A_4A_4 \mid A_4A_4\,Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

$A_1 \rightarrow bA_3 \mid bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4$

$Z \rightarrow bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4 \mid bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4$

Greibach Normal Form

Example:

$A_1 \rightarrow bA_3 \mid bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4$

$A_4 \rightarrow bA_3A_4 \mid b \mid bA_3A_4Z \mid bZ$

$Z \rightarrow bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4 \mid bA_3A_4A_4 \mid bA_4 \mid bA_3A_4ZA_4 \mid bZA_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Grammar in Greibach Normal Form

Greibach Normal Form

Example: Convert the CFG into GNF

CNF	New Labels	Updated CNF
$S \rightarrow AC$	$S = A_1$	$A_1 \rightarrow A_2A_3$
$A \rightarrow CS \mid b,$	$A = A_2$	$A_2 \rightarrow A_3A_1 \mid b$
$C \rightarrow SA \mid a$	$C = A_3$	$A_3 \rightarrow A1A2 \mid a$

Greibach Normal Form

Example:

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_3A_1 \mid b$

$A_3 \rightarrow A_1A_2 \mid a$

First Step

$$A_i \rightarrow A_jX_k \quad j > i$$

X_k is a string of zero or more variables

✗ $A_3 \rightarrow A_1A_2$

Greibach Normal Form

Example:

First Step $A_i \rightarrow A_j X_k \quad j > i$

$A_3 \rightarrow A_1 A_2 \mid a$

$A_3 \rightarrow A_2 A_3 A_2 \mid a$

$A_3 \rightarrow A_3 A_1 A_3 A_2 \mid b A_3 A_2 \mid a$

$A_1 \rightarrow A_2 A_3$

$A_2 \rightarrow A_3 A_1 \mid b$

$A_3 \rightarrow A_1 A_2 \mid a$

Greibach Normal Form

Example:

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_3A_1 \mid b$

$A_3 \rightarrow A_3A_1A_3A_2 \mid bA_3A_2 \mid a$

Second Step

Eliminate Left Recursions

✗ $A_3 \rightarrow A_3A_1A_3A_2$

Greibach Normal Form

Example:

Second Step

Eliminate Left Recursions

$A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2Z \mid aZ$

$Z \rightarrow A_1A_3A_2 \mid A_1A_3A_2Z$

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_3A_1 \mid b$

$A_3 \rightarrow A_3A_1A_3A_2 \mid bA_3A_2 \mid a$

Greibach Normal Form

Example:

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_3A_1 \mid b$

$A3 \rightarrow bA3A2 \mid a \mid bA3A2\ Z \mid aZ$

$Z \rightarrow A_1A_3A_2 \mid A_1A_3A_2\ Z$

$A \rightarrow aX$

GNF

Greibach Normal Form

Example:

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_3A_1 \mid b$

$A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2Z \mid aZ$

$Z \rightarrow A_1A_3A_2 \mid A_1A_3A_2Z$

$A_1 \rightarrow A_3A_1A_3 \mid bA_3$

$A_1 \rightarrow bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2ZA_1A_3 \mid aZA_1A_3 \mid bA_3$

$A_2 \rightarrow bA_3A_2A_1 \mid aA_1 \mid bA_3A_2ZA_1 \mid aZA_1 \mid b$

$Z \rightarrow bA_3A_2A_2A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_2ZA_1A_3A_3A_2 \mid aZA_1A_3A_3A_2 \mid$
$bA_3A_3A_2 \mid bA_3A_2A_2A_3A_3A_2Z \mid aA_1A_3A_3A_2Z \mid bA_3A_2ZA_1A_3A_3A_2Z \mid$
$aZA_1A_3A_3A_2Z \mid bA_3A_3A_2Z$

Greibach Normal Form

Example:

$A_1 \rightarrow bA_3A_2A_2A_3 \mid aA1A_3 \mid bA_3A_2ZA1A_3 \mid aZA1A_3 \mid bA_3$

$A_2 \rightarrow bA_3A_2A_1 \mid aA_1 \mid bA_3A_2ZA_1 \mid aZA_1 \mid b$

$A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2Z \mid aZ$

$Z \rightarrow bA_3A_2A_2A_3\,A_3A_2 \mid aA1A_3\,A_3A_2 \mid bA_3A_2ZA1A_3\,A_3A_2 \mid aZA1A_3A_3A_2 \mid$
$bA_3A_3A_2 \mid bA_3A_2A_2A_3\,A_3A_2Z \mid aA1A_3\,A_3A_2Z \mid bA_3A_2ZA1A_3A_3A_2Z \mid$
$aZA1A_3A_3A_2Z \mid bA_3\,A_3A_2Z$

Grammar in Greibach Normal Form

Problem (Aug-2017 InSem,6 Marks)

- Convert given CFG into GNF

$S \rightarrow BS \mid Aa$

$A \rightarrow Bc$

$B \rightarrow Ac$ where,

$V = \{S, A, B\}$ & $T = \{a, c\}$

Recall

- Normal Forms
- CNF (Chomsky Normal Form)
- GNF (Greibach Normal Form)

Problem (Nov-2016 InSem,6 Marks)

- Convert given CFG to GNF.
  S->AA | 0
  A-> SS | 1

The Chomsky Hierarchy

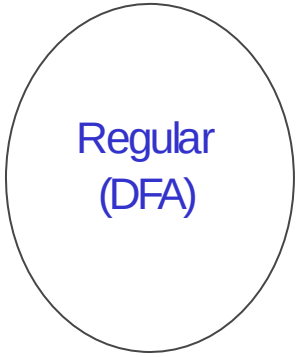
Grammars and languages

The hierarchy

- A containment hierarchy (strictly nested sets) of classes of formal grammars

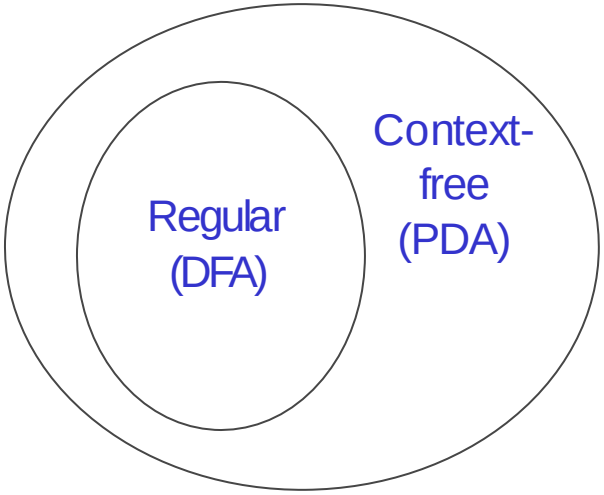
The hierarchy

- A containment hierarchy (strictly nested sets) of classes of formal grammars



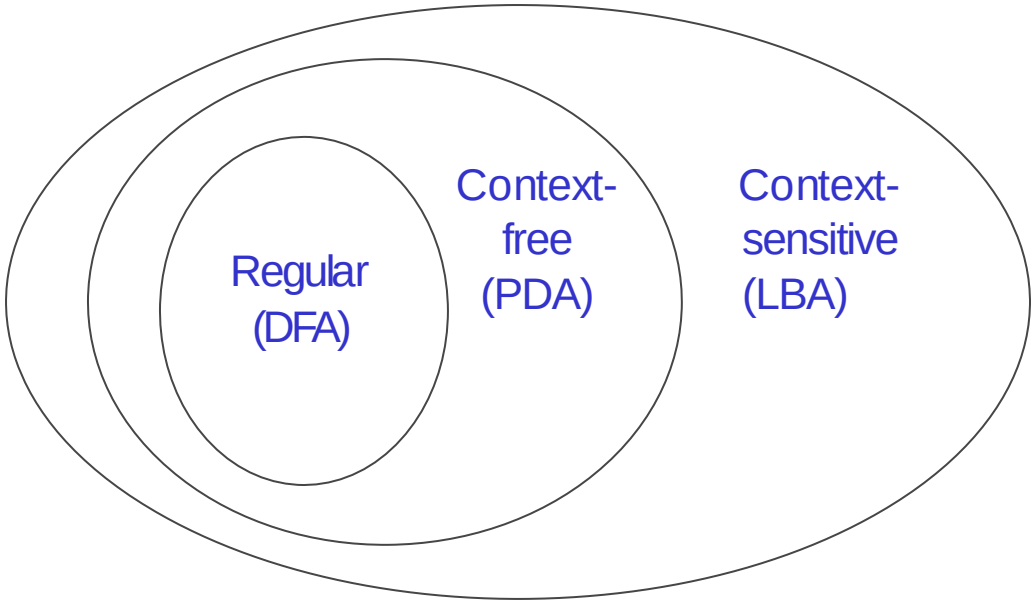
The hierarchy

- A containment hierarchy (strictly nested sets) of classes of formal grammars



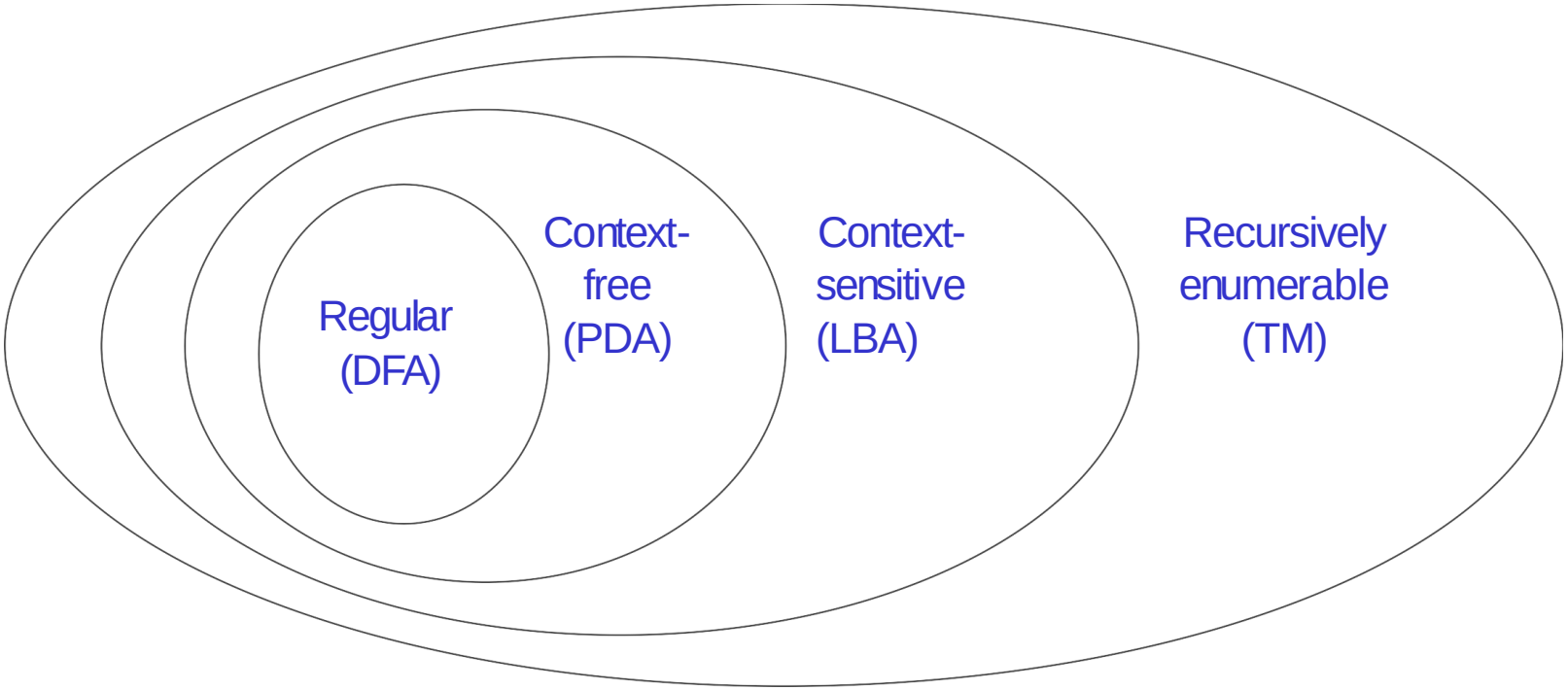
The hierarchy

- A containment hierarchy (strictly nested sets) of classes of formal grammars



The hierarchy

- A containment hierarchy (strictly nested sets) of classes of formal grammars



The hierarchy

The hierarchy

Class	Grammars	Languages	Automaton

The hierarchy

Class	Grammars	Languages	Automaton
Type-0	Unrestricted	Recursively enumerable (Turing-recognizable)	Turing machine

The hierarchy

Class	Grammars	Languages	Automaton
Type-0	Unrestricted	Recursively enumerable (Turing-recognizable)	Turing machine
Type-1	Context-sensitive	Context-sensitive	Linear-bounded

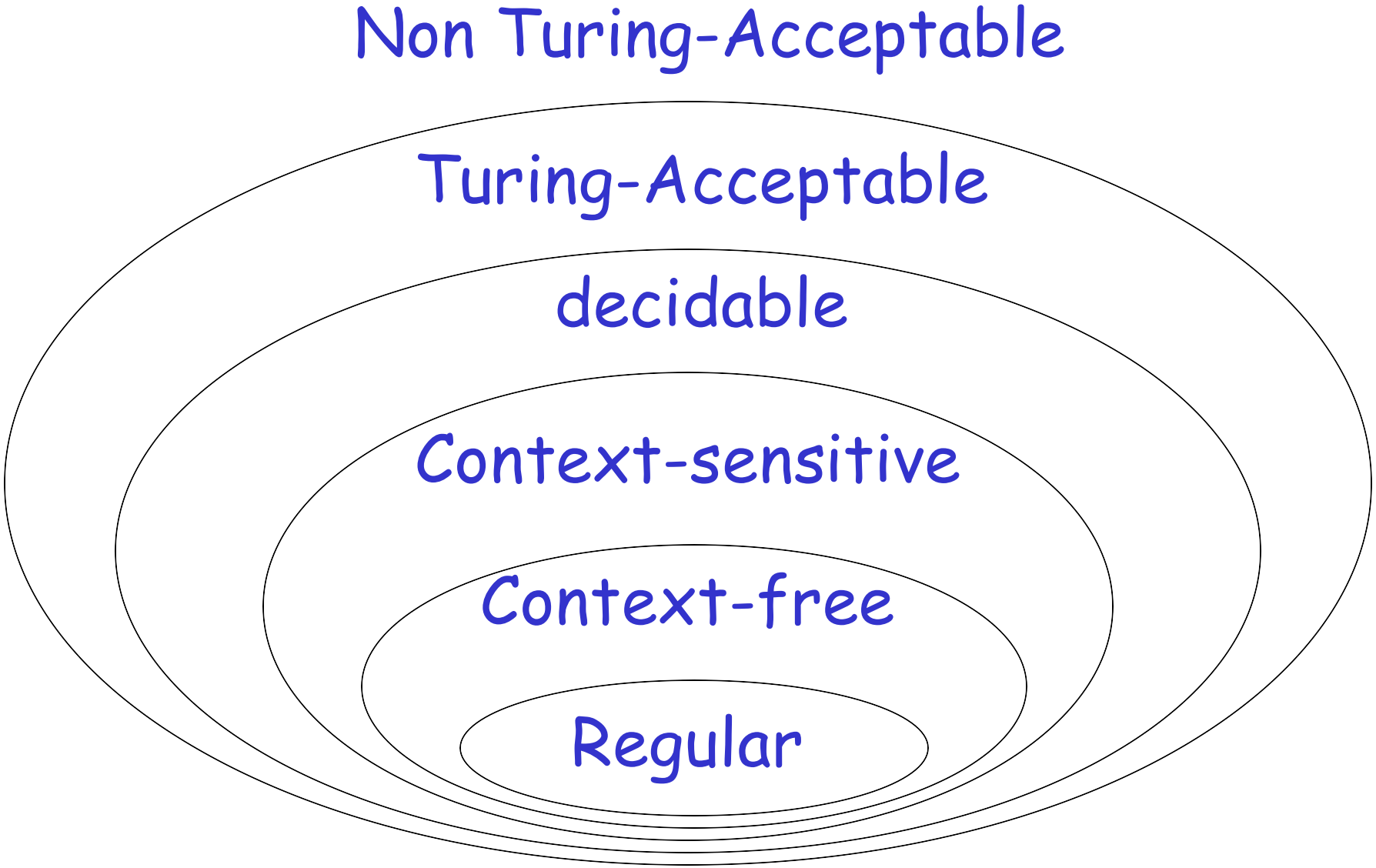
The hierarchy

Class	Grammars	Languages	Automaton
Type-0	Unrestricted	Recursively enumerable (Turing-recognizable)	Turing machine
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
Type-2	Context-free	Context-free	Pushdown

The hierarchy

Class	Grammars	Languages	Automaton
Type-0	Unrestricted	Recursively enumerable (Turing-recognizable)	Turing machine
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
Type-2	Context-free	Context-free	Pushdown
Type-3	Regular	Regular	Finite

The Chomsky Hierarchy



Recall

- Chomsky Hierarchy

Regular Grammer

Linear Grammars

- Grammars with at most one variableat the right side of a production

- Examples:
-

$$S \to aSb \qquad S \to Ab$$

$$S \to \lambda \qquad A \to aAb$$

$$A \to \lambda$$

A Non-Linear Grammar

- Grammar G :

$$S \to SS$$

$$S \to \lambda S$$

$$\to aSb$$

$$S \to bSa$$

$$L(G) = \{w: \quad n_a(w) = n_b(w)\}$$

Number of a in string w

Another Linear Grammar

- Grammar G :

$$S \to A$$

$$A \to aB \,|\, \lambda$$

$$B \to Ab$$

$$L(G) = \{a^n b^n : n \geq 0\}$$

Right-Linear Grammars

- All productions have form: $A \rightarrow xB$

or

$A \rightarrow x$

string of terminals

- Example: $S \rightarrow abS$

$S \rightarrow a$

Left-Linear Grammars

- All productions have form: $A \to Bx$

or

$A \to x$

string of terminals

- Example: $S \to Aab$

$A \to Aab \mid B$

$B \to a$

Regular Grammars

Regular Grammars

- A regular grammar is any
- right-linear or left-linear grammar

- Examples:

G_1

$S \to abS$

$S \to a$

G_2

$S \to Aab$

$A \to Aab \mid B$

$B \to a$

Observation

- Regular grammars generate regular languages

- Examples:

G_1

$S \to abS$

$S \to a$

$L(G_1) = (ab)^* a$

G_2

$S \to Aab$

$A \to Aab \mid B$

$B \to a$

$L(G_2) = aab(ab)^*$

Equivalence of
Regular Grammer and Finite Automata

Theorem

{ Languages Generated by Regular Grammars } { Regular Languages }

Theorem - Part 1

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Grammars}\end{array}\right\} \subseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

Any regular grammar generates a regular language

Theorem - Part 2

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Grammars}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

Any regular language is generated by a regular grammar

Proof - Part 1

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Grammars}\end{array}\right\} \subseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

The language $L(G)$ generated by any regular grammar G is regular

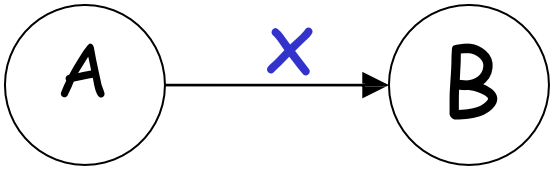
The case of Right-Linear Grammars

- Let G be a right-linear grammar
- We will prove: $L(G)$ is regular

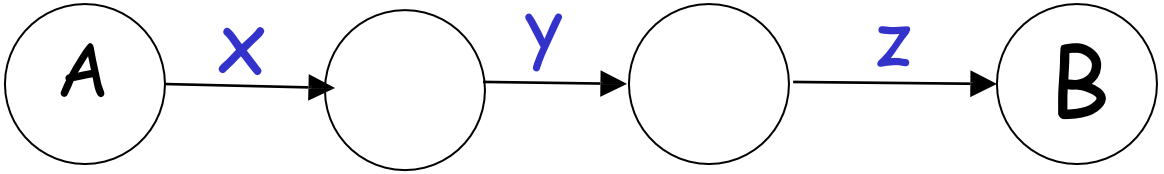
- **Proof idea:** We will construct NFA M
- with $L(M) = L(G)$

Simple connection between right-linear grammars and NFAs,

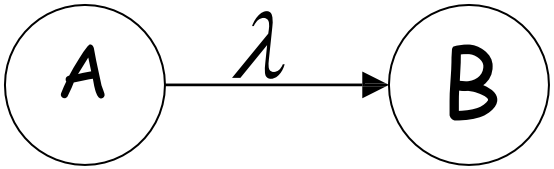
- A → xB



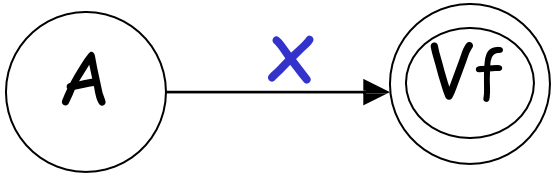
- A → xyzB



- A → B



- A → x



- Grammar G is right-linear

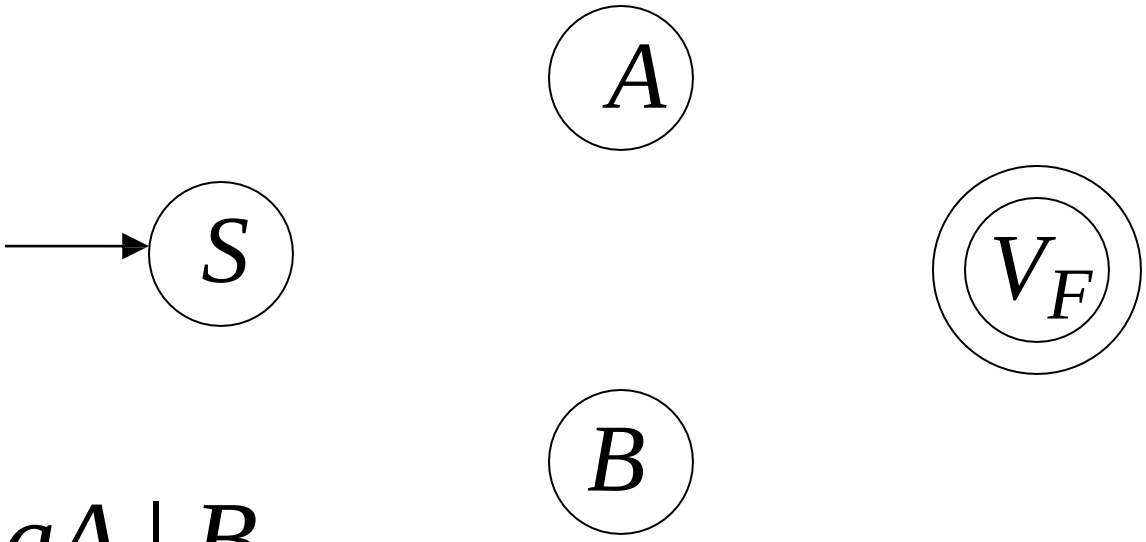
Example:

$$S \to aA \mid B$$

$$A \to aa\ B$$

$$B \to b\ B \mid a$$

- Construct NFA M such that
- every state is a grammar variable:



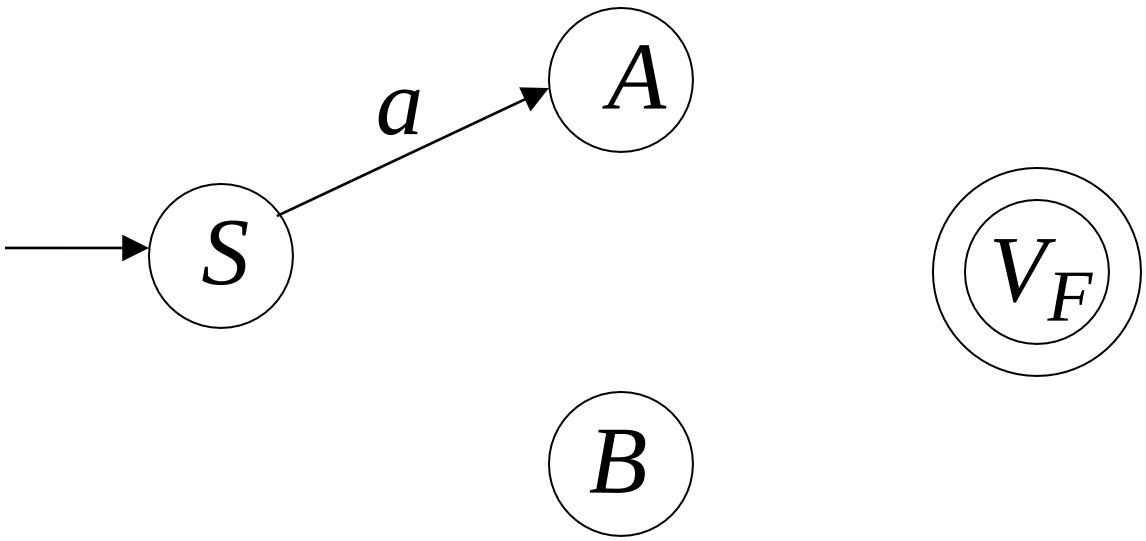
special final state

$$S \rightarrow aA \mid B$$

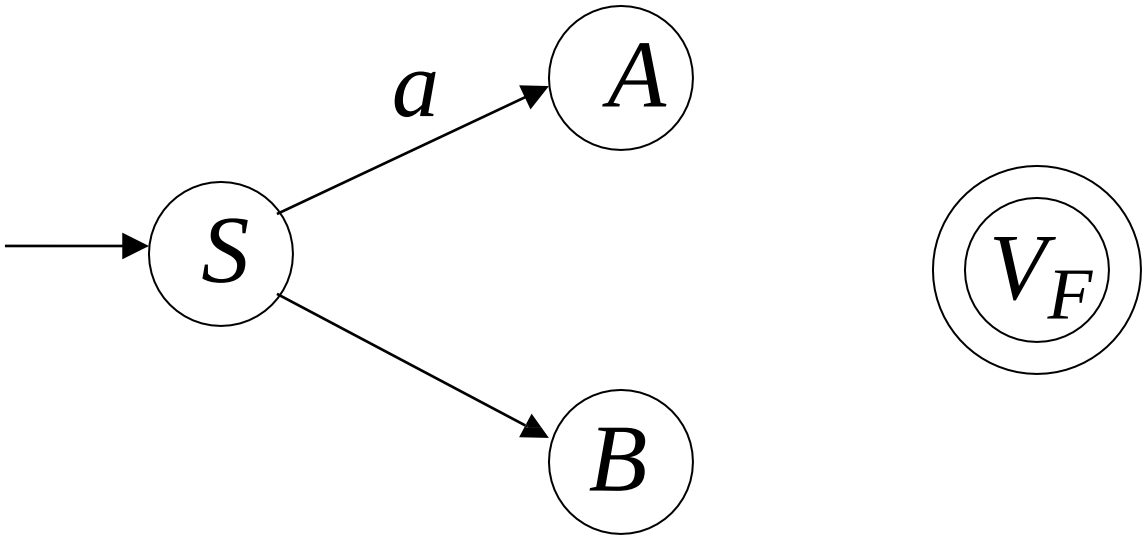
$$A \rightarrow aa\ B$$

$$B \rightarrow b\ B \mid a$$

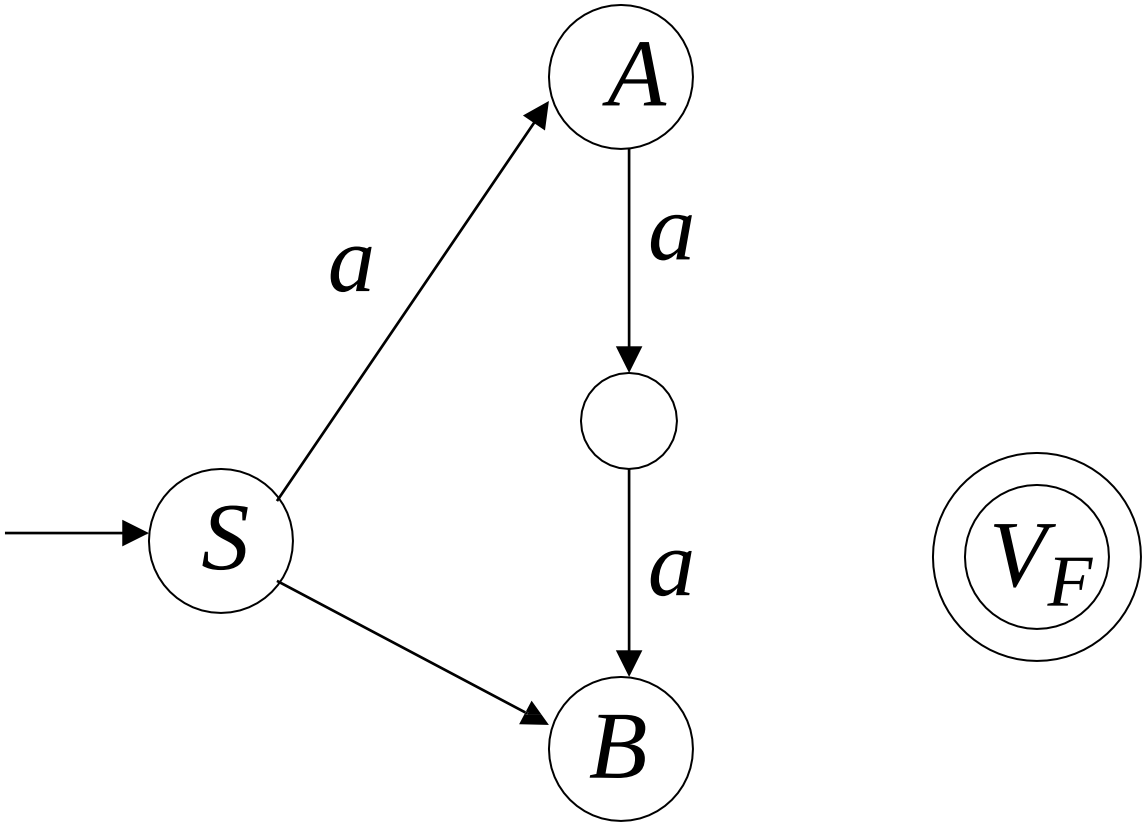
- Add edges for each production:



$$S \rightarrow aA$$

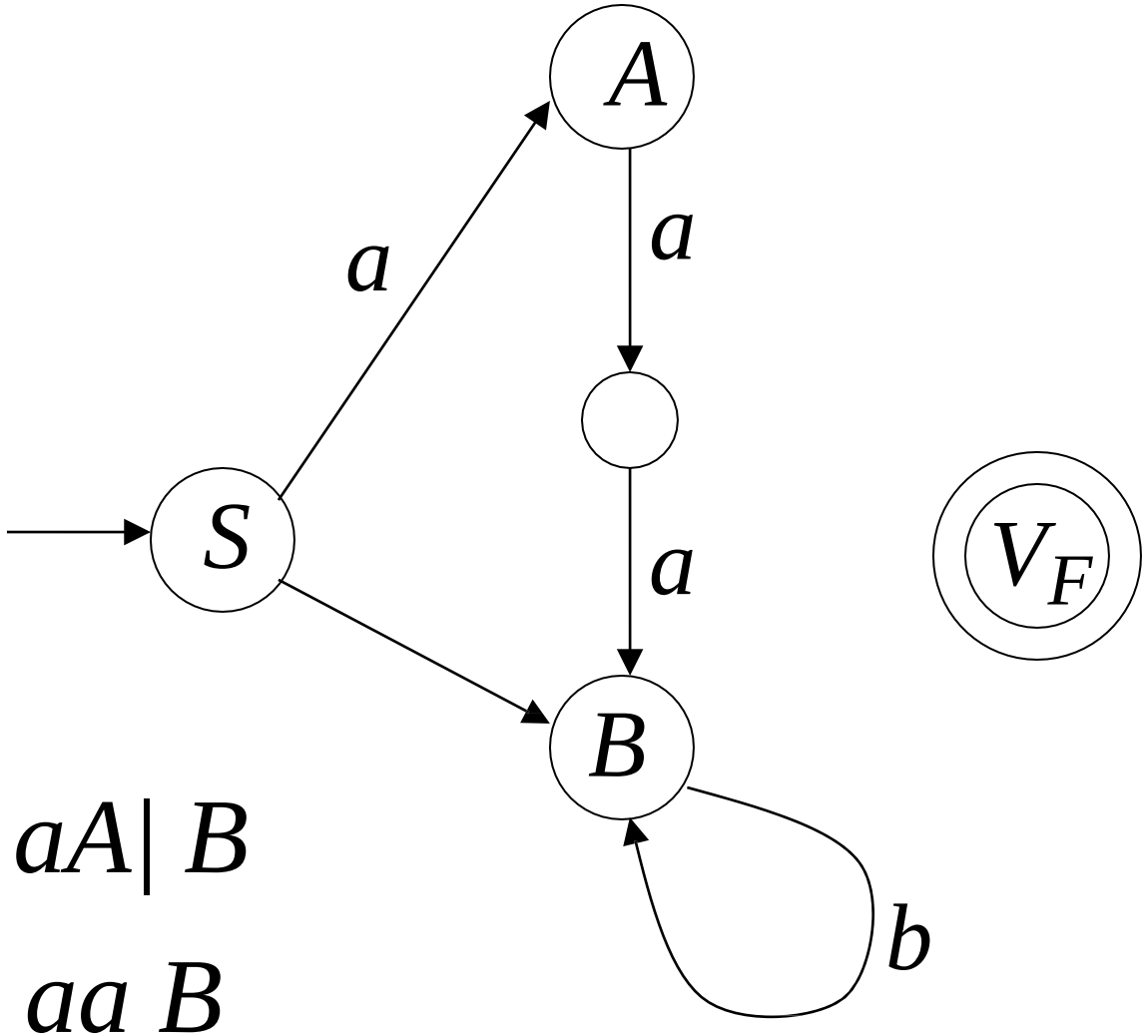


$$S \rightarrow aA \mid B$$



$$S \rightarrow aA \mid B$$

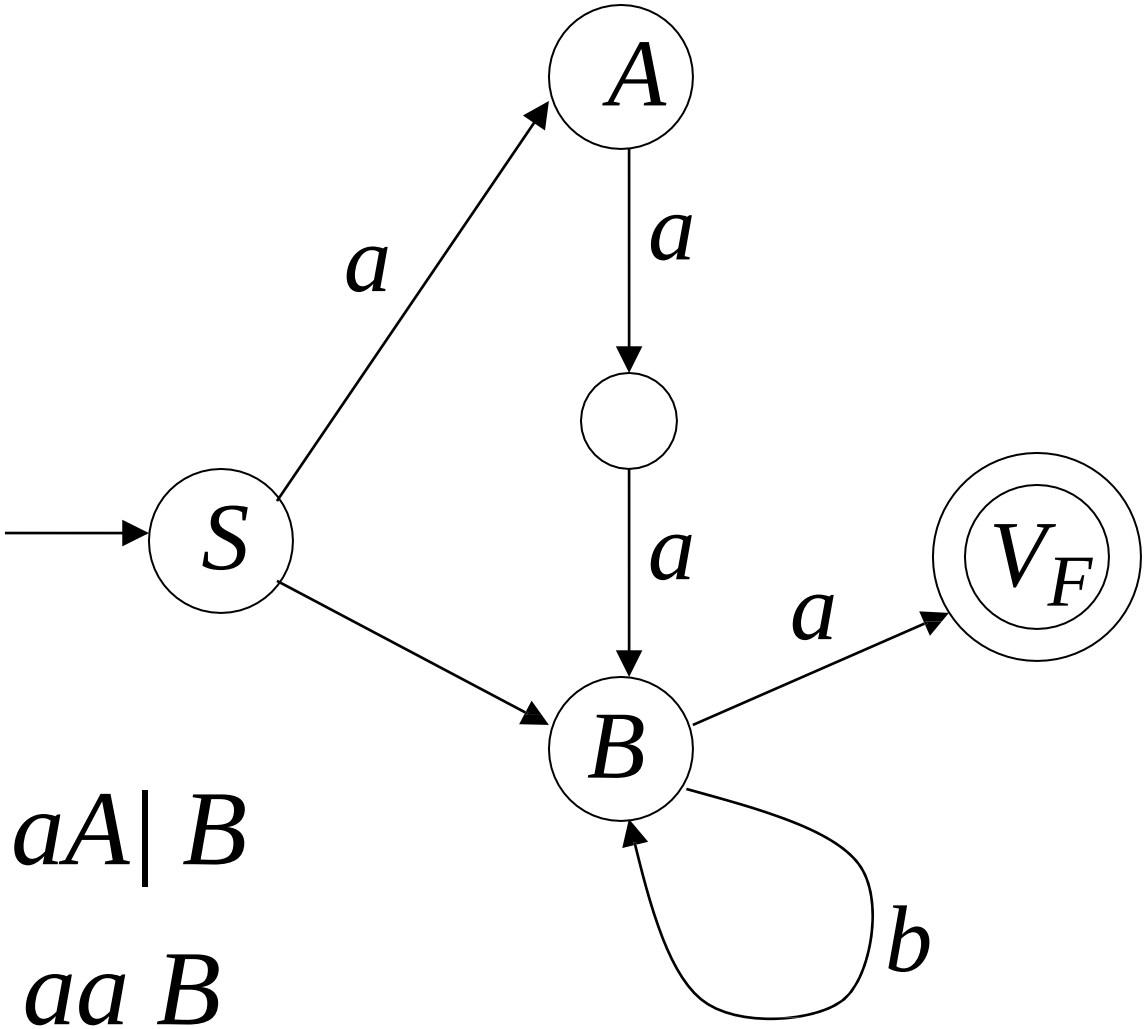
$$A \rightarrow aaB$$



$S \rightarrow aA|\ B$

$A \rightarrow aa\ B$

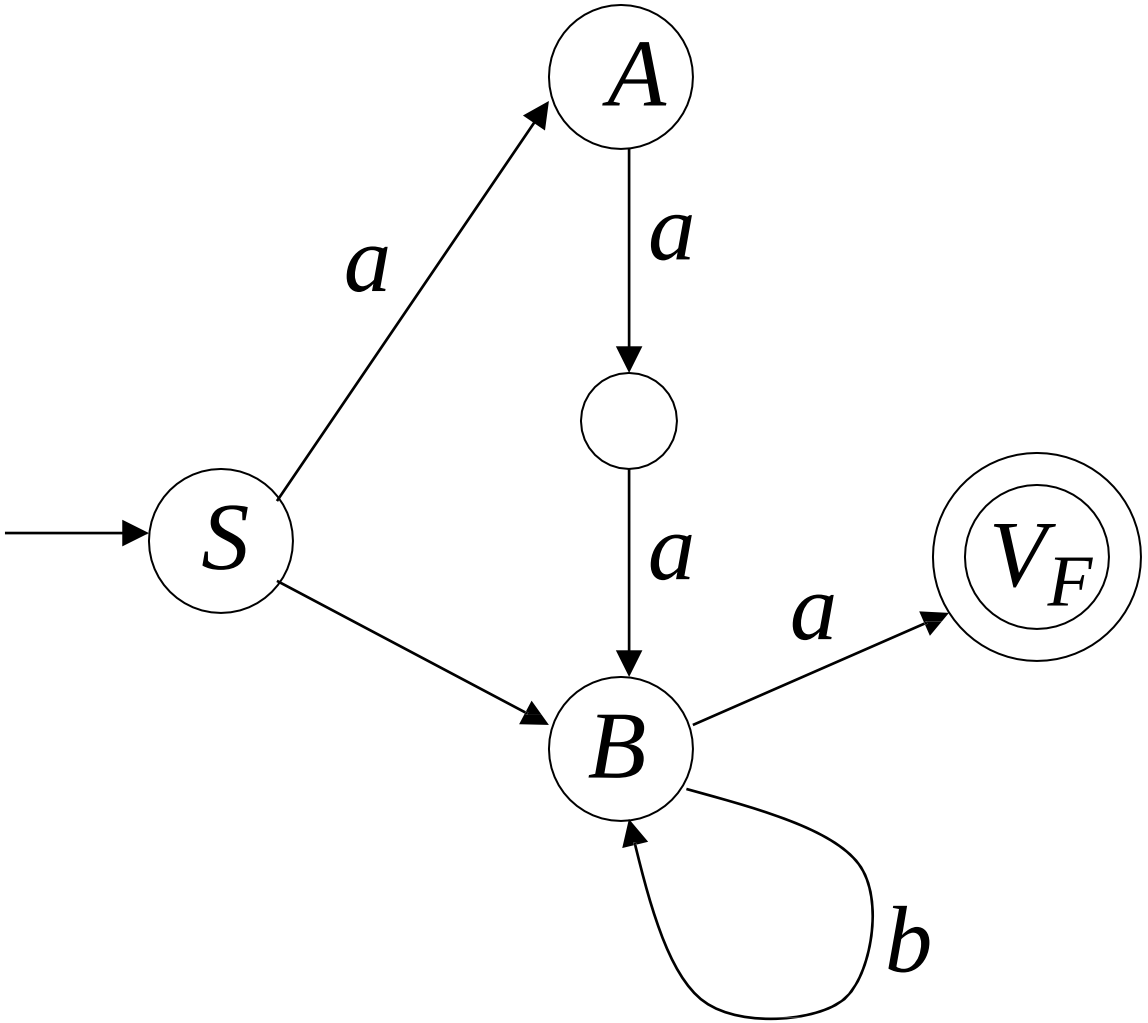
$B \rightarrow bB$



$$S \rightarrow aA \mid B$$

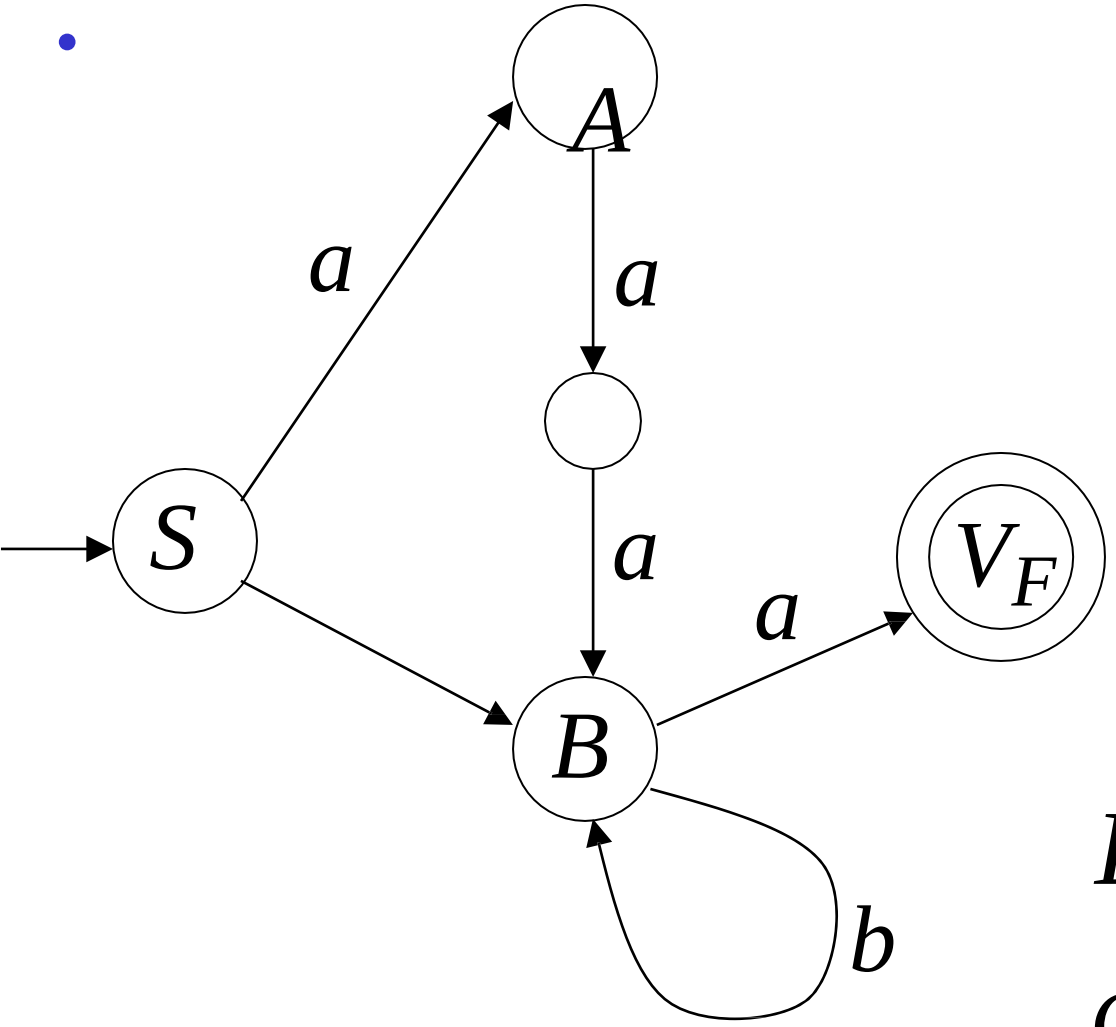
$$A \rightarrow aa\ B$$

$$B \rightarrow bB \mid a$$



$$S \Rightarrow aA \Rightarrow aaaB \Rightarrow aaabB \Rightarrow aaaba$$

NFA M



Grammar

G

$S \rightarrow aA \mid B$

$A \rightarrow aa\ B$

$B \rightarrow bB \mid a$

$L(M) = L(G) =$

$aaab^*a + b^*a$

In General

- A right-linear grammar G

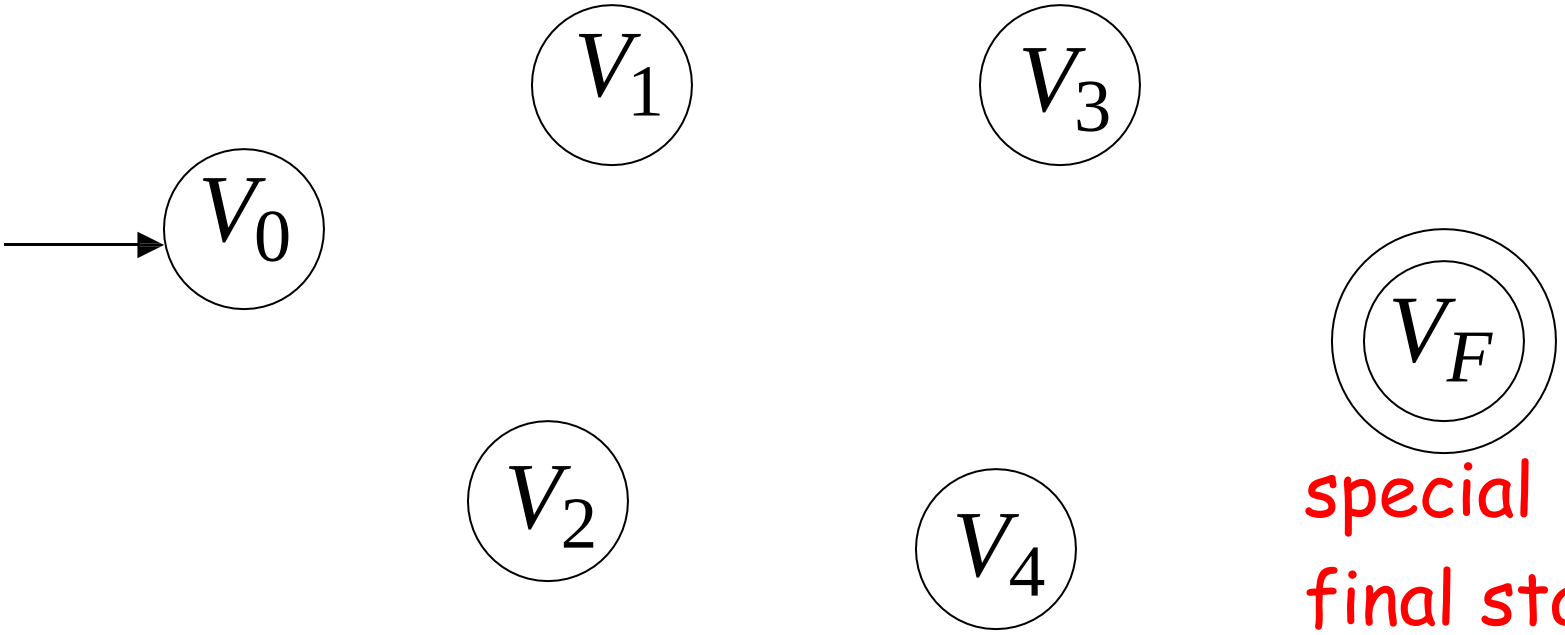
- has variables: $V_0, V_1, V_2, \square$

- and productions: $V_i \to a_1 a_2 \square a_m V_j$

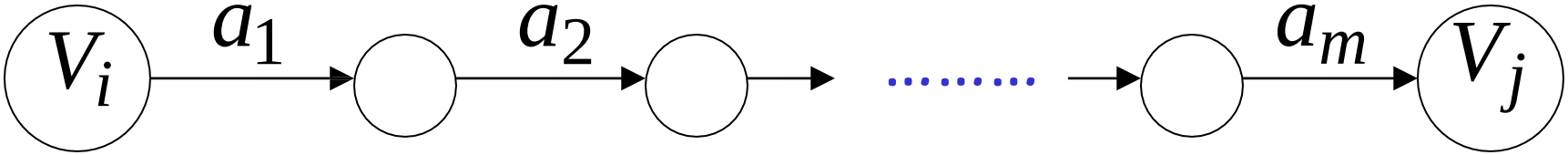
or

$$V_i \to a_1 a_2 \square a_m$$

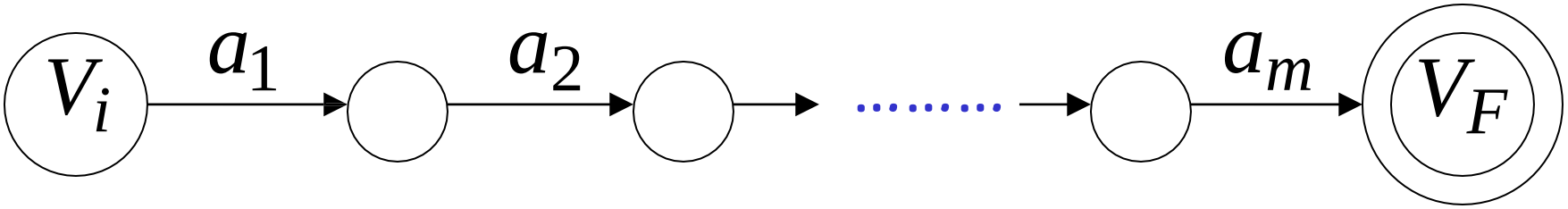
- We construct the NFA M such that:
- each variable V_i corresponds to a node (state):
- Start variable corresponds to a initial node (state):
- If any state have the null transition then that state become the final state otherwise Create Special Final state addition to the existing state



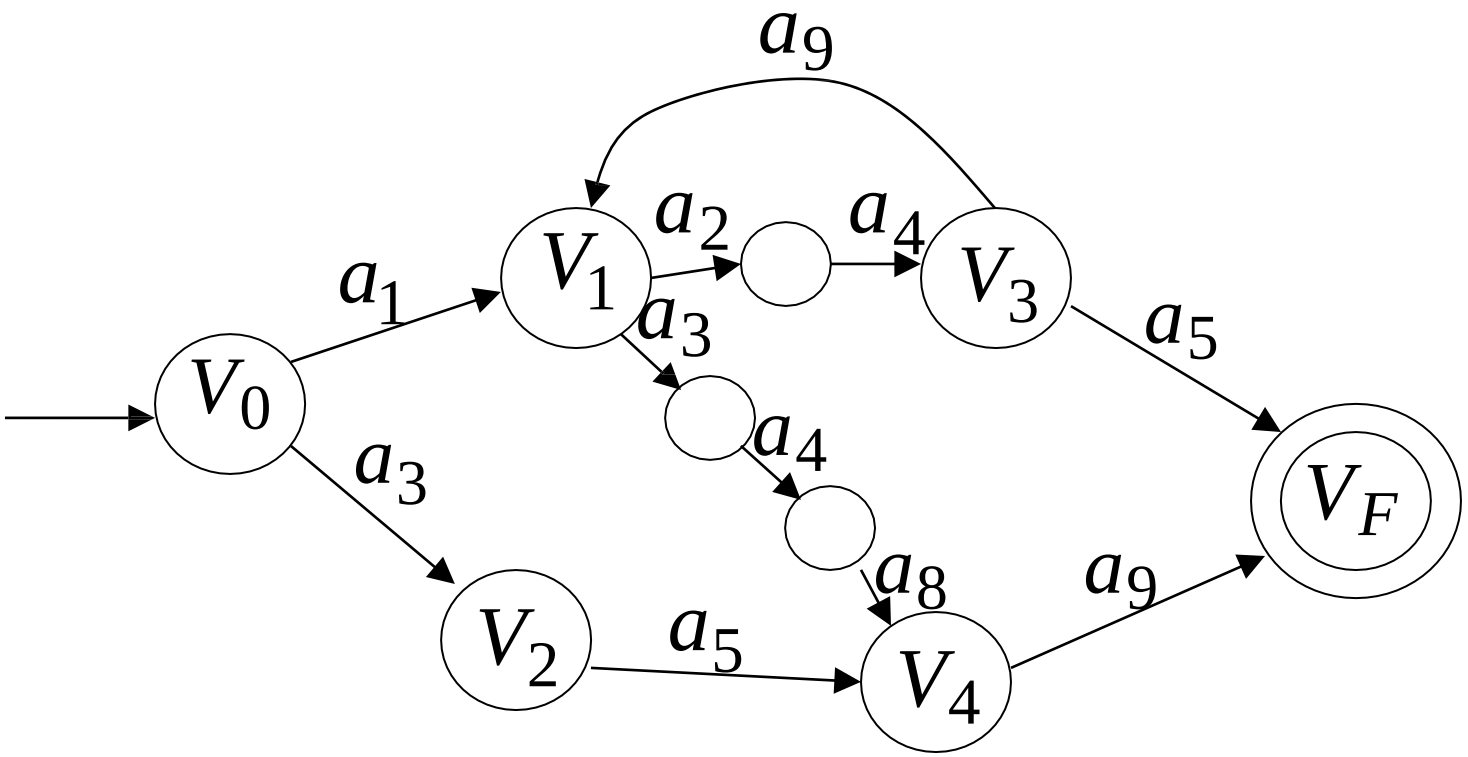
- For each production: $V_i \rightarrow a_1 a_2 \square a_m V_j$
- weadd transitions and intermediate nodes



- For each production: $V_i \rightarrow a_1 a_2 \square a_m$
- weadd transitions and intermediate nodes



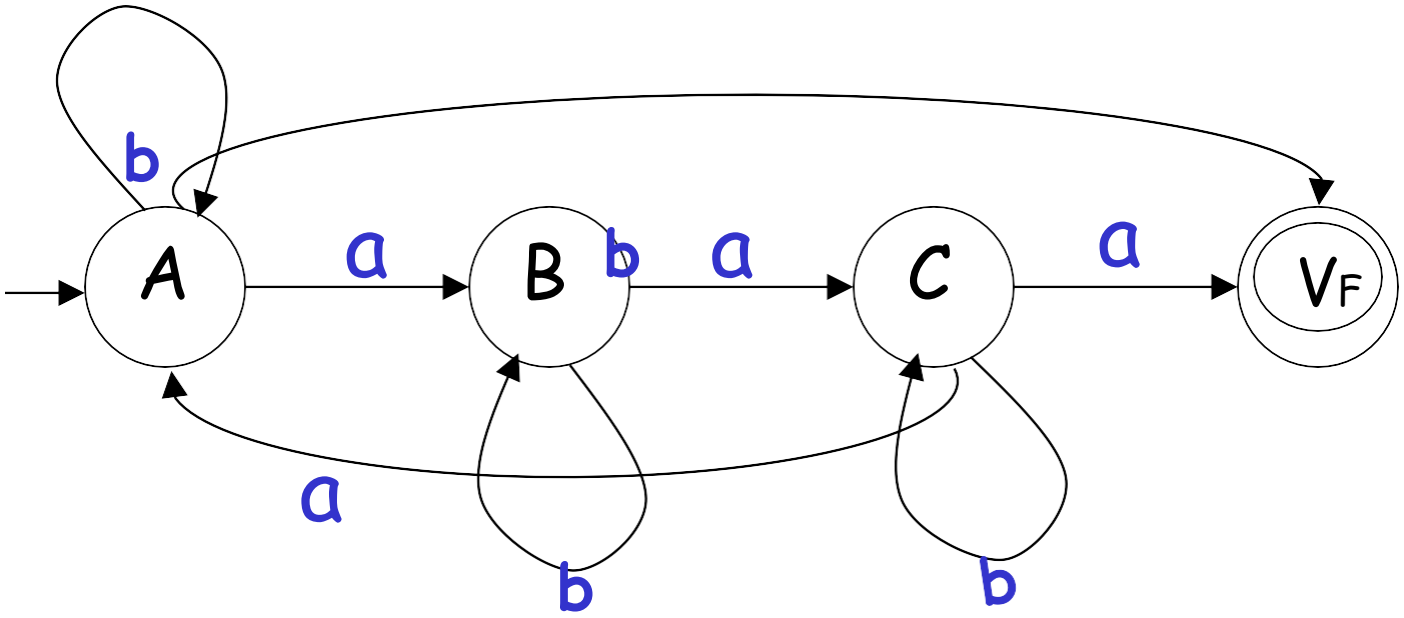
- Resulting NFA M looks like this:



It holds that: $L(G) = L(M)$

Convert the G into Corresponding FA

- A ➆ aB | bA | b
- B ➆ aC | bB
- C ➆ aA | bC | ϕ



Recall

- Regular Grammar
- Right Liner Grammar
- Right Liner Grammar into FA

Problem (Aug-2017 InSem, 8 Marks)

- Convert the given regular grammar to its equivalent FA.
- S ➆ aS / bS / aA
- A ➆ bB
- B ➆ aC
- C ➆ a

Convert the G into Corresponding FA

- Right Linear Grammar:
- S --> 0S | A |
  A --> 1B
  B --> 0A | 1A | 0 | 1

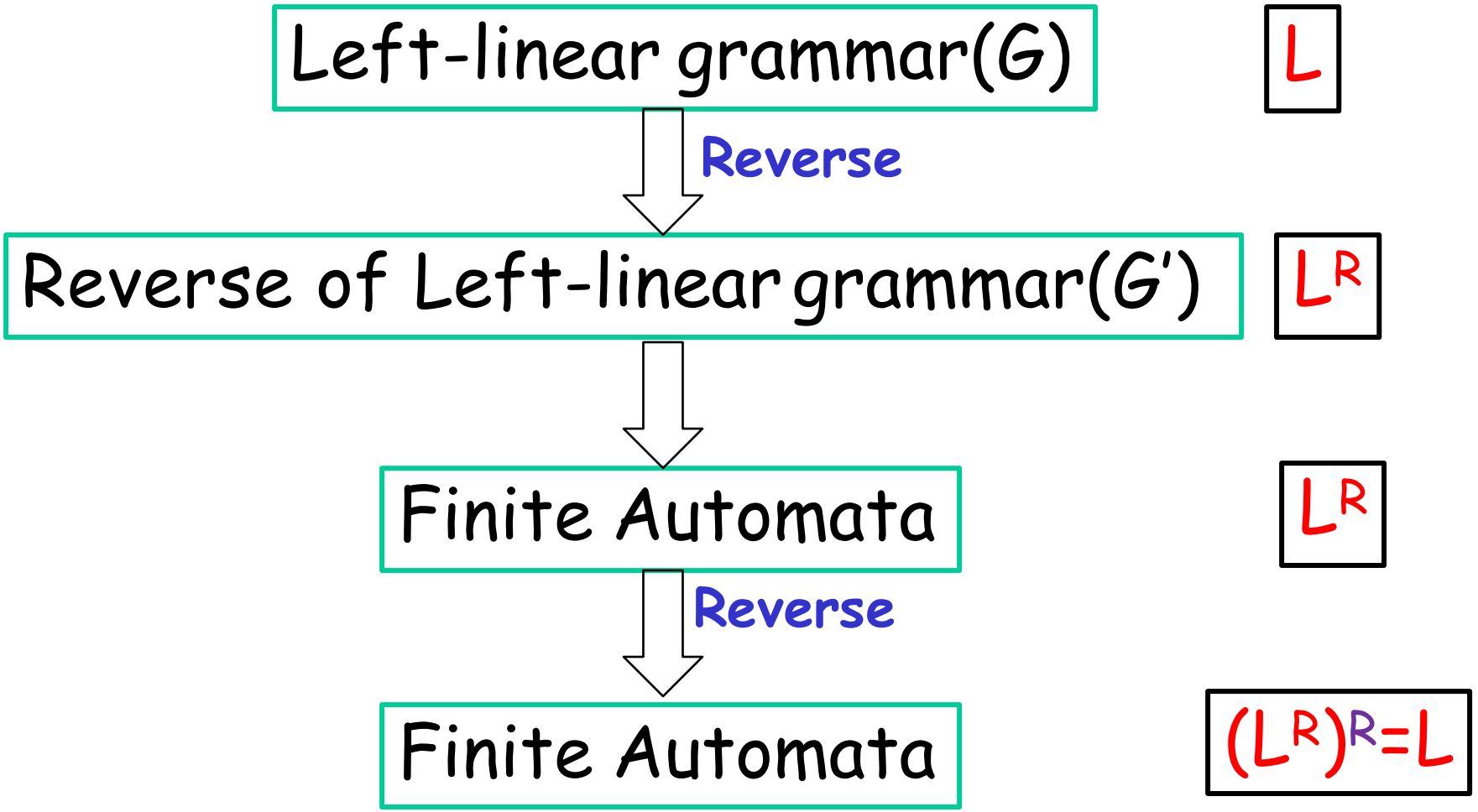
The case of Left-Linear Grammars

- Let G be a left-linear grammar

- We will prove: $L(G)$ is regular

- **Proof idea:** We will construct NFA M
- with $L(M) = L(G)$

Procedure to Prove



- Since G is left-linear grammar the productions look like:

$$A \to Ba_1a_2 \cdots a_k$$

$$A \to a_1a_2 \cdots a_k$$

- Construct right-linear grammar G'

Left linear G

$$A \to Ba_1a_2 \cdots a_k$$

$$A \to Bv$$

Right linear G'

$$A \to a_k \cdots a_2a_1B$$

$$A \to v^RB$$

- Construct right-linear grammar G'

Left linear G

$$A \to a_1 a_2 \square a_k$$

$$A \to v$$

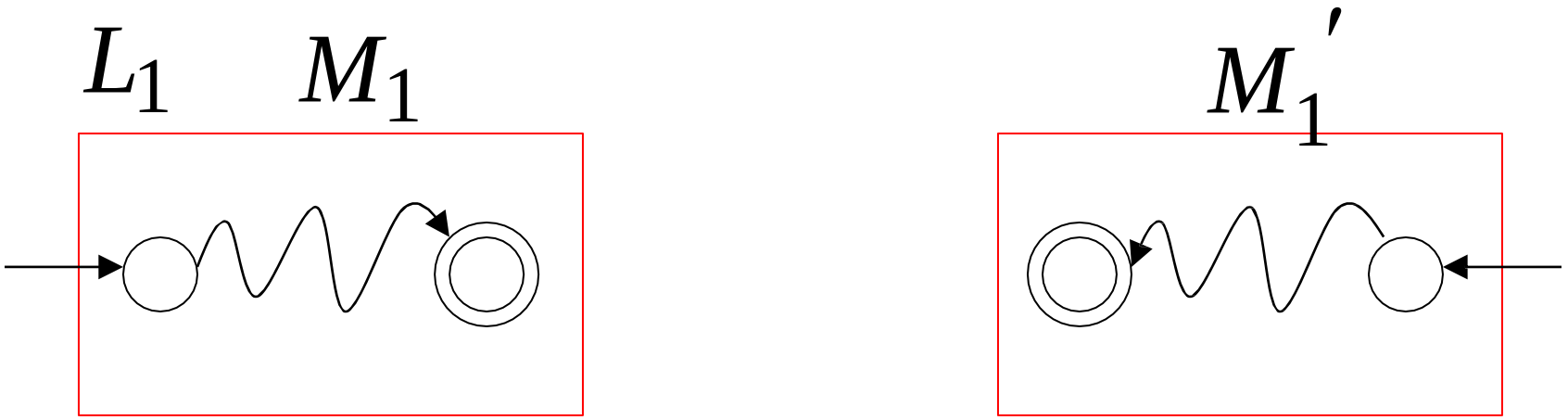
Right linear G'

$$A \to a_k \square a_2 a_1$$

$$A \to v^R$$

Reverse

NFA for L_1^R



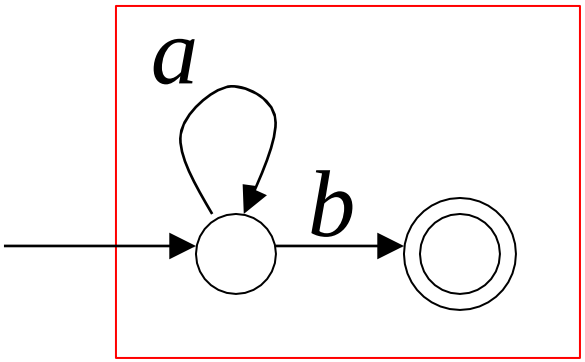
1. Reverse all transitions

2. Make initial state accepting state and vice versa

Example

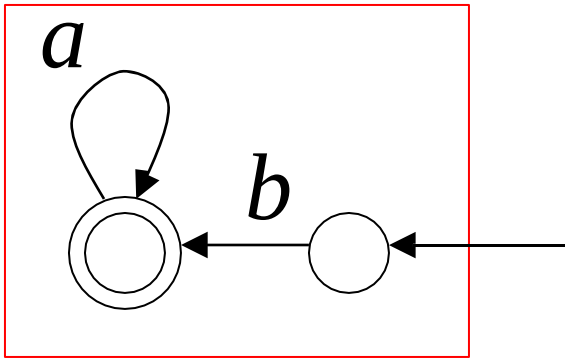
$L_1 = \{a^n b\}$

M_1



$L_1^R = \{b a^n\}$

M_1'



- It is easy to see that: $L(G) = L(G')^R$
- Since G' is right-linear, we have:
-
-

Convert the Grammar into Corresponding FA

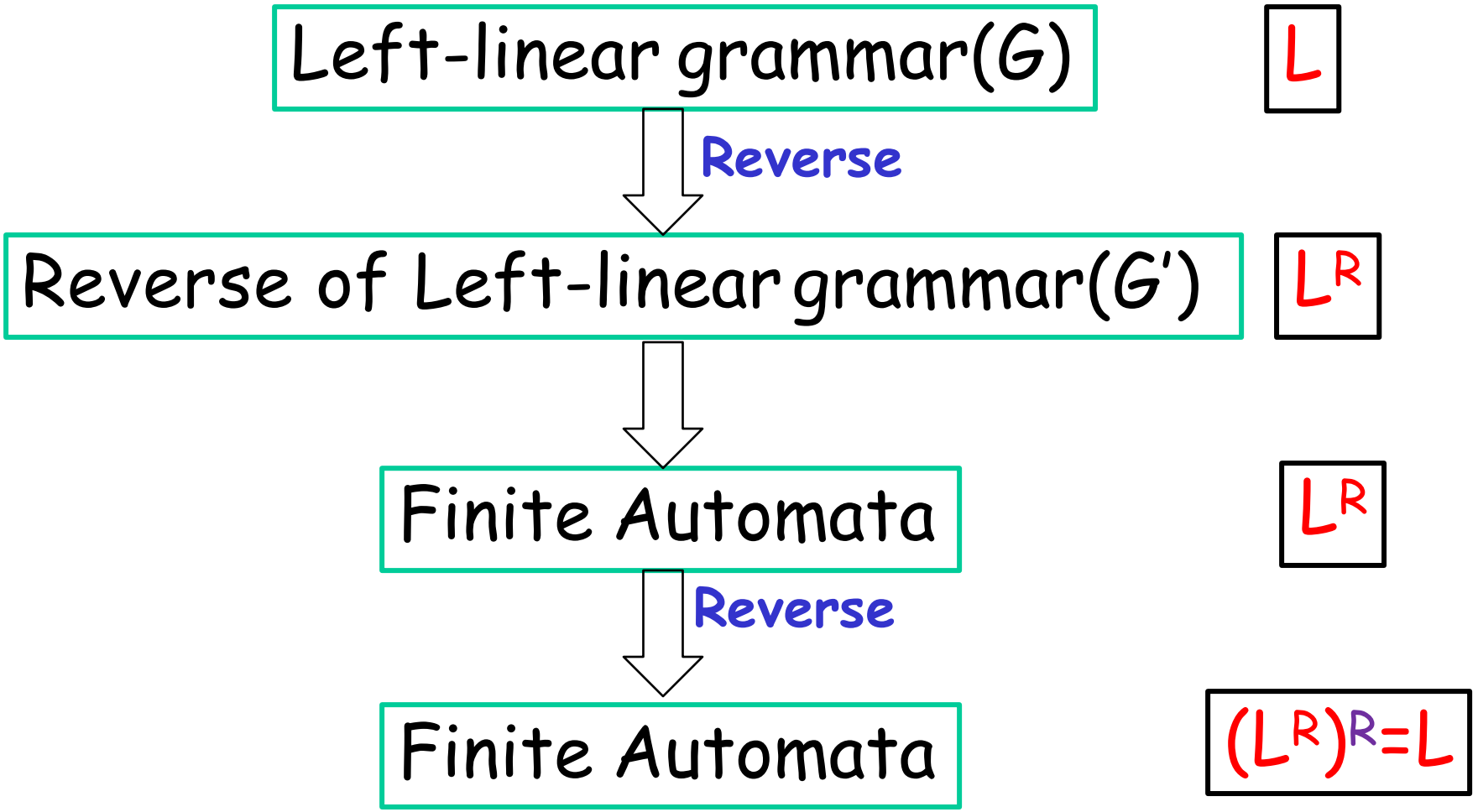
A -> Ba/Ab/b
B -> Ca/Bb
C -> Aa/Cb

A -> aB/bA/b
B -> aC/bB
C -> aA/bC

Convert LLG to its Equivalent FA

- $S \rightarrow Ab$
- $S \rightarrow Sb$
- $A \rightarrow Aa$
- $A \rightarrow a$

Procedure to Prove



Proof - Part 2

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Grammars}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

Any regular language L is generated by some regular grammar G

Any regular language L is generated by some regular grammar G

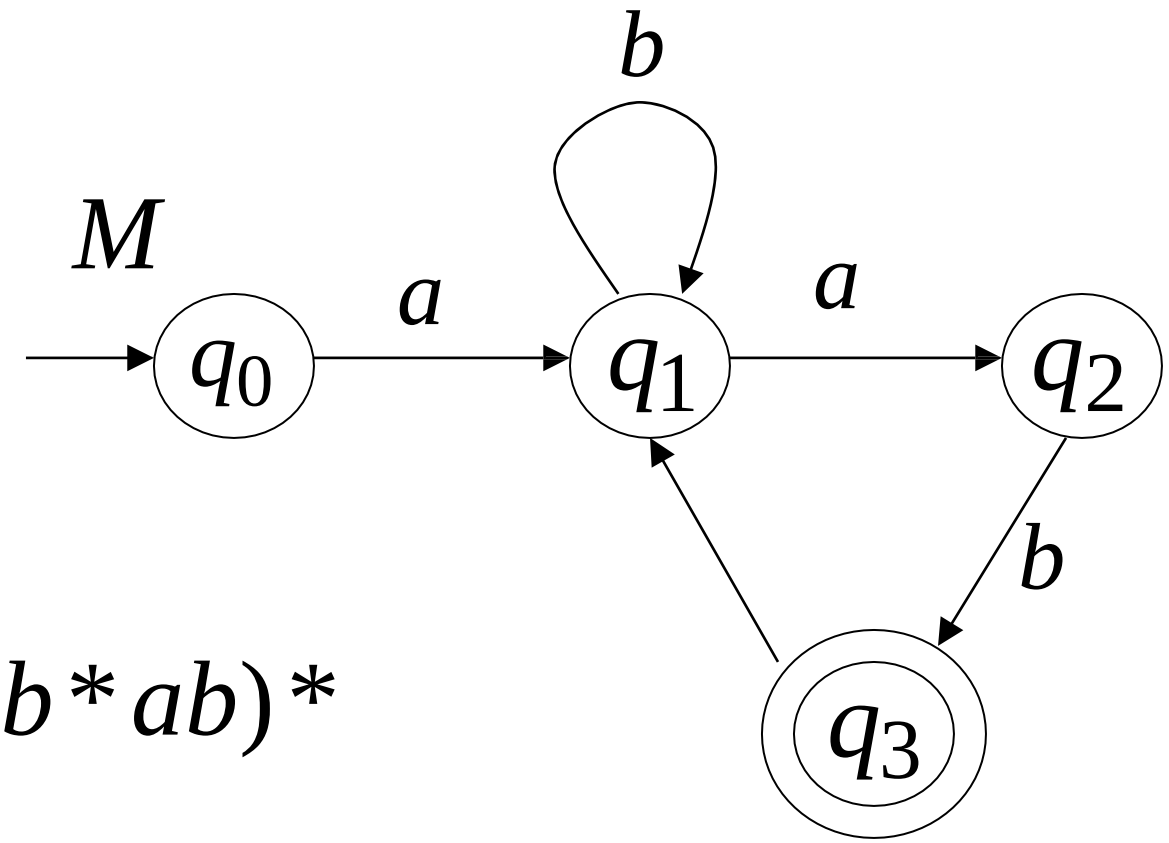
Proof idea:

Let M be the NFA with $L = L(M)$.

Construct from M a regular grammar G such that $L(M) = L(G)$

- Since L is regular
- there is an NFA M such that $L = L(M)$

Example:

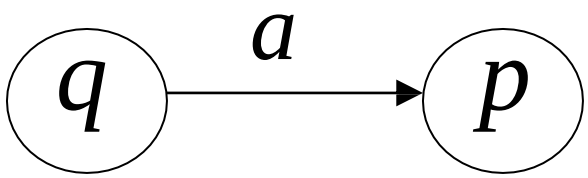


$$L = ab*ab(b*ab)*$$

$$L = L(M)$$

In General

For any transition:

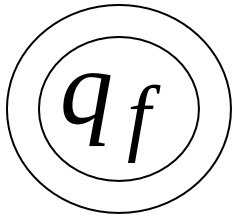


Add production:

$$q \rightarrow ap$$

variable terminal variable

For any final state:

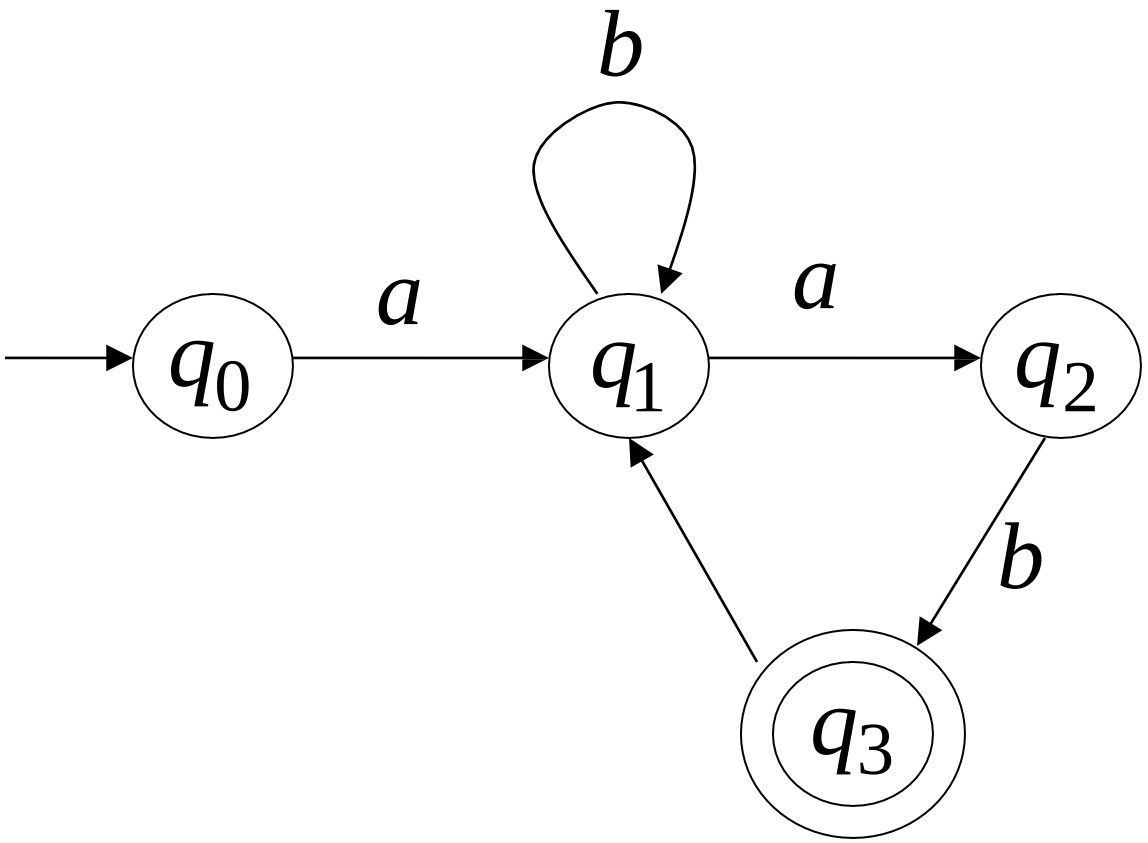


Add production:

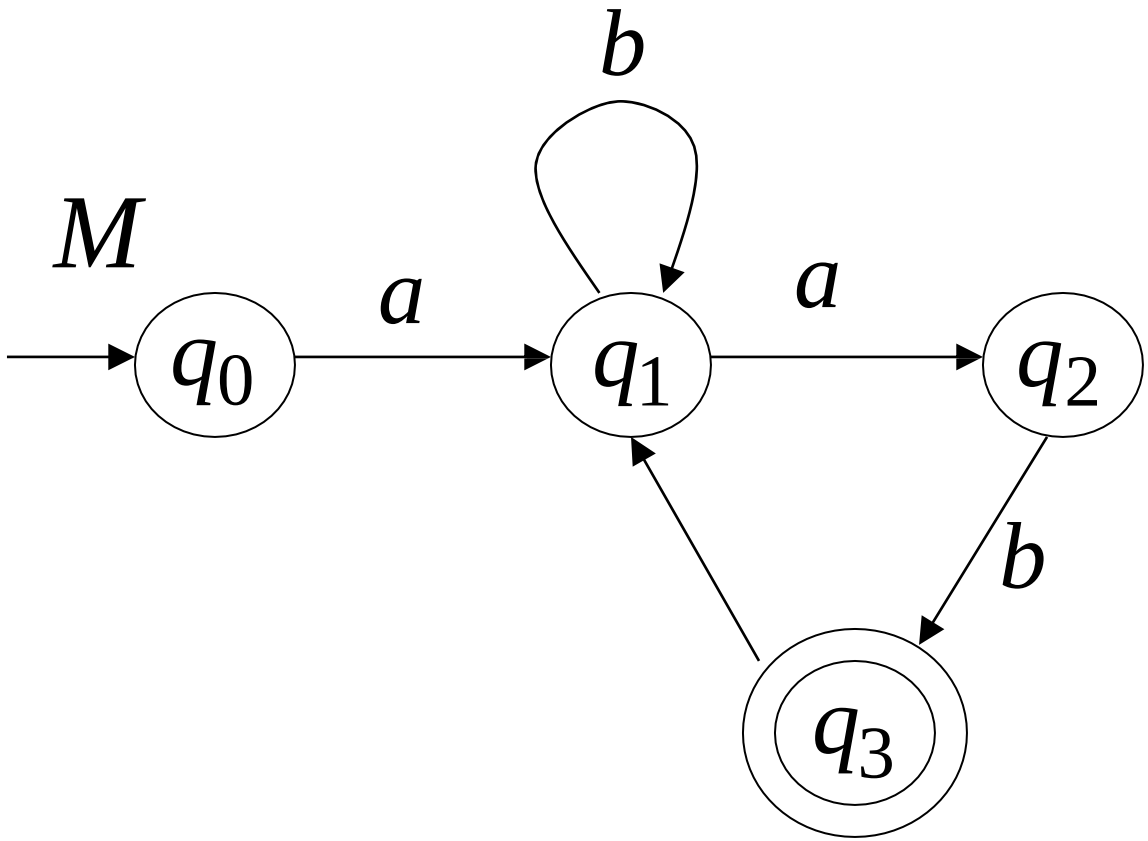
$$q_f \rightarrow \lambda$$

- Convert M to a right-linear grammar

M

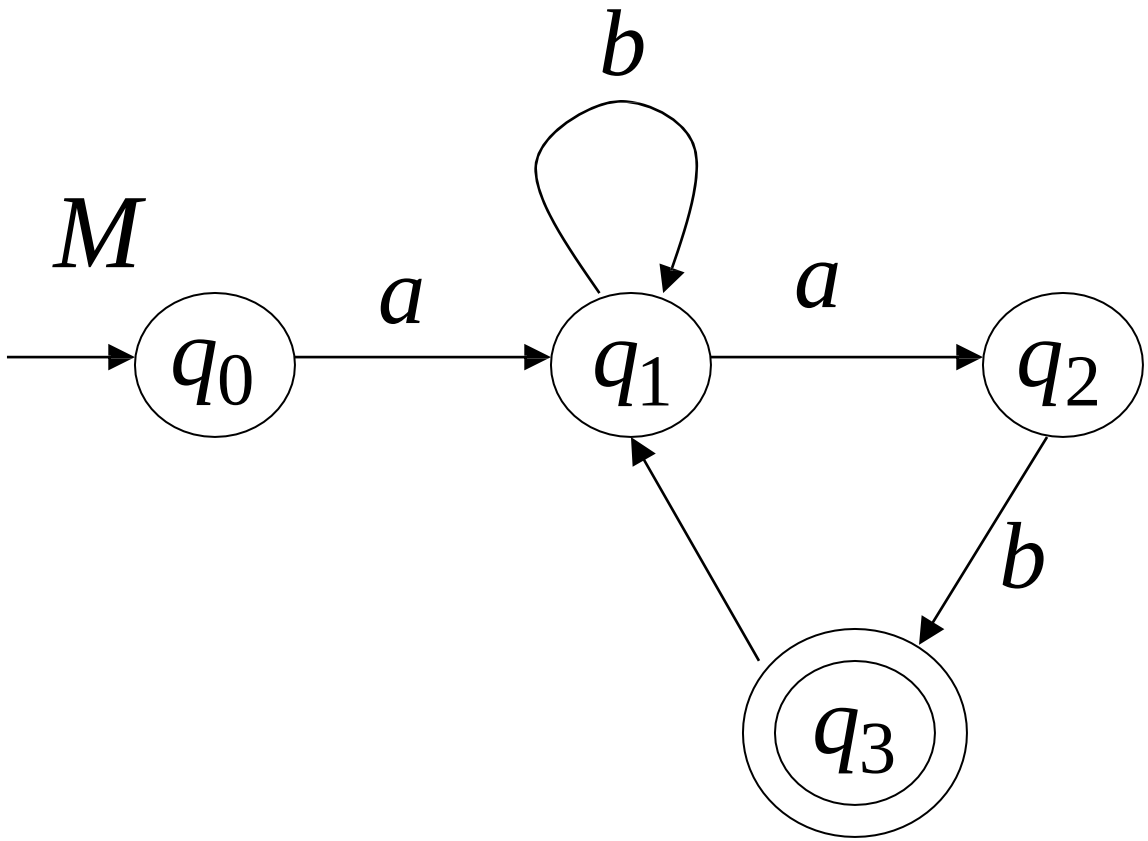


$q_0 \to aq_1$



$$q_0 \rightarrow aq_1$$

$$q_1 \rightarrow bq_1$$



$q_0 \rightarrow aq_1$

$q_1 \rightarrow bq_1$

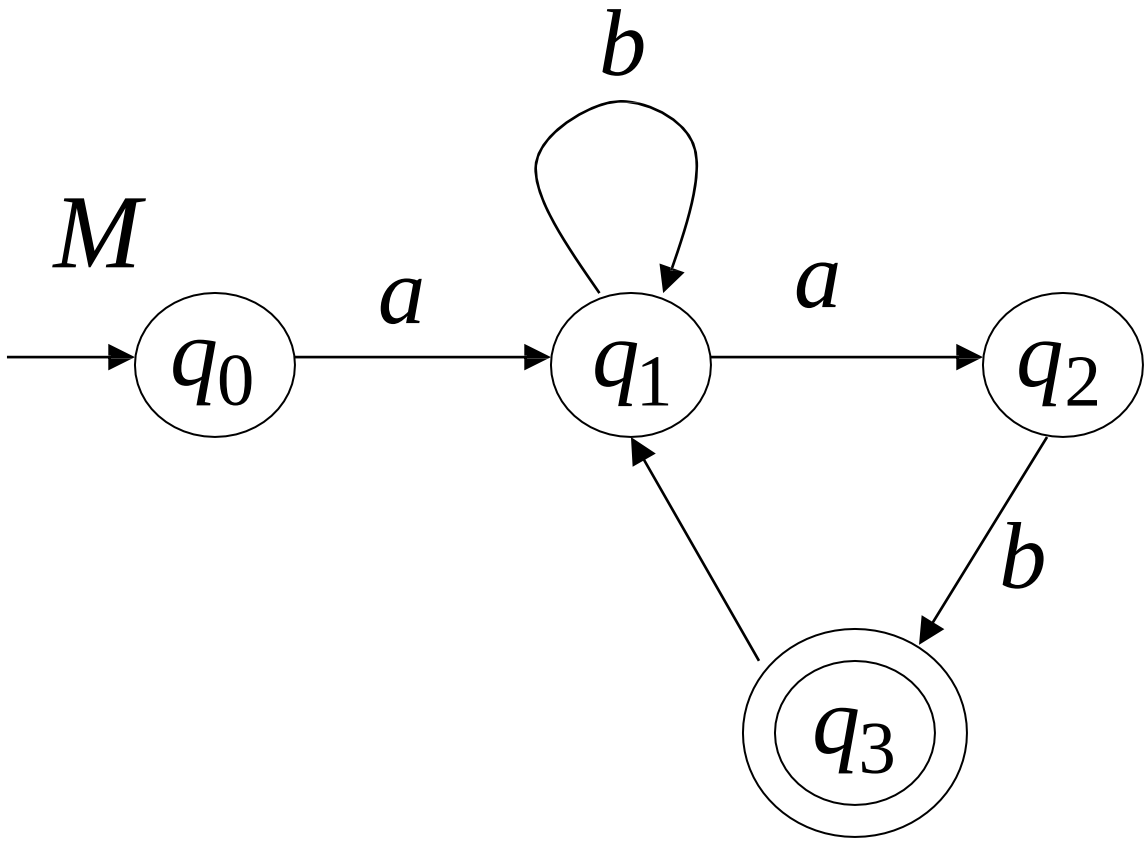
$q_1 \rightarrow aq_2$

$q_0 \to aq_1$

$q_1 \to bq_1$

$q_1 \to aq_2$

$q_2 \to bq_3$



$$L(G) = L(M) = L$$

G

$$q_0 \to aq_1$$

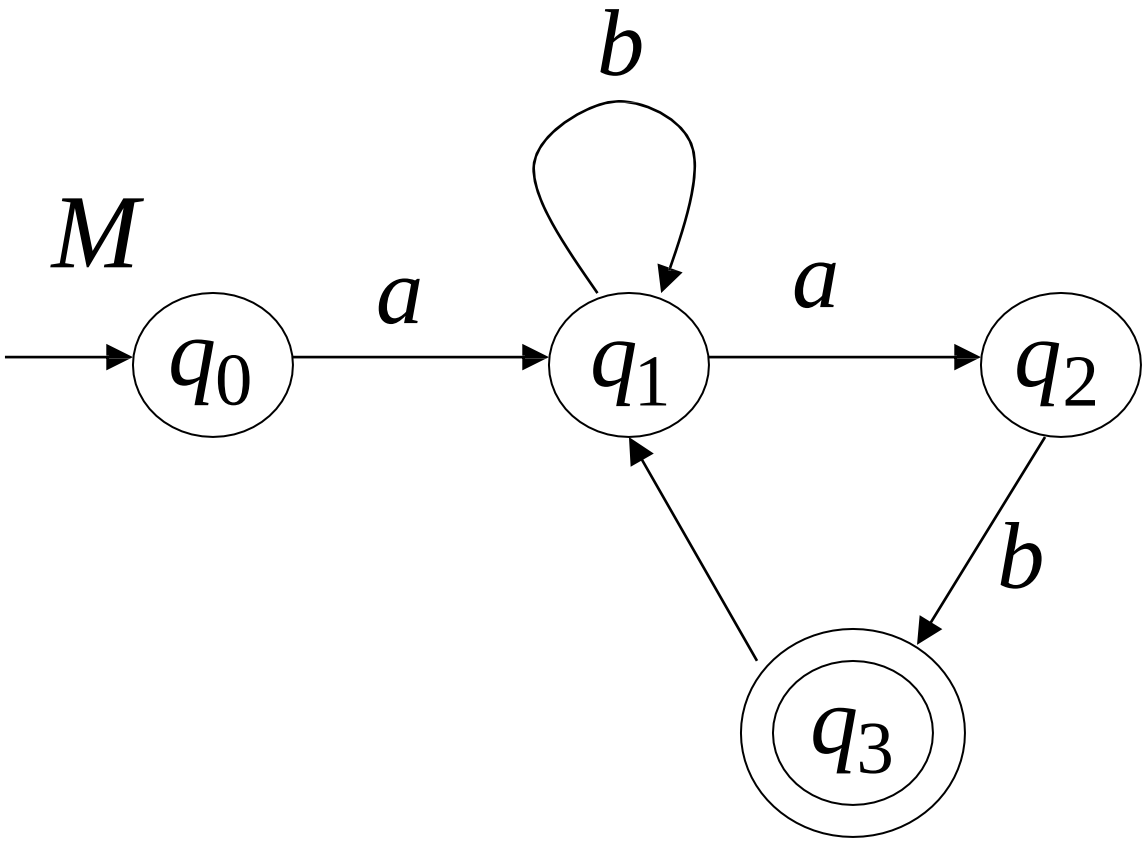
$$q_1 \to bq_1$$

$$q_1 \to aq_2$$

$$q_2 \to bq_3$$

$$q_3 \to q_1$$

M



$$L(G) = L(M) = L$$

G

$q_0 \to aq_1$

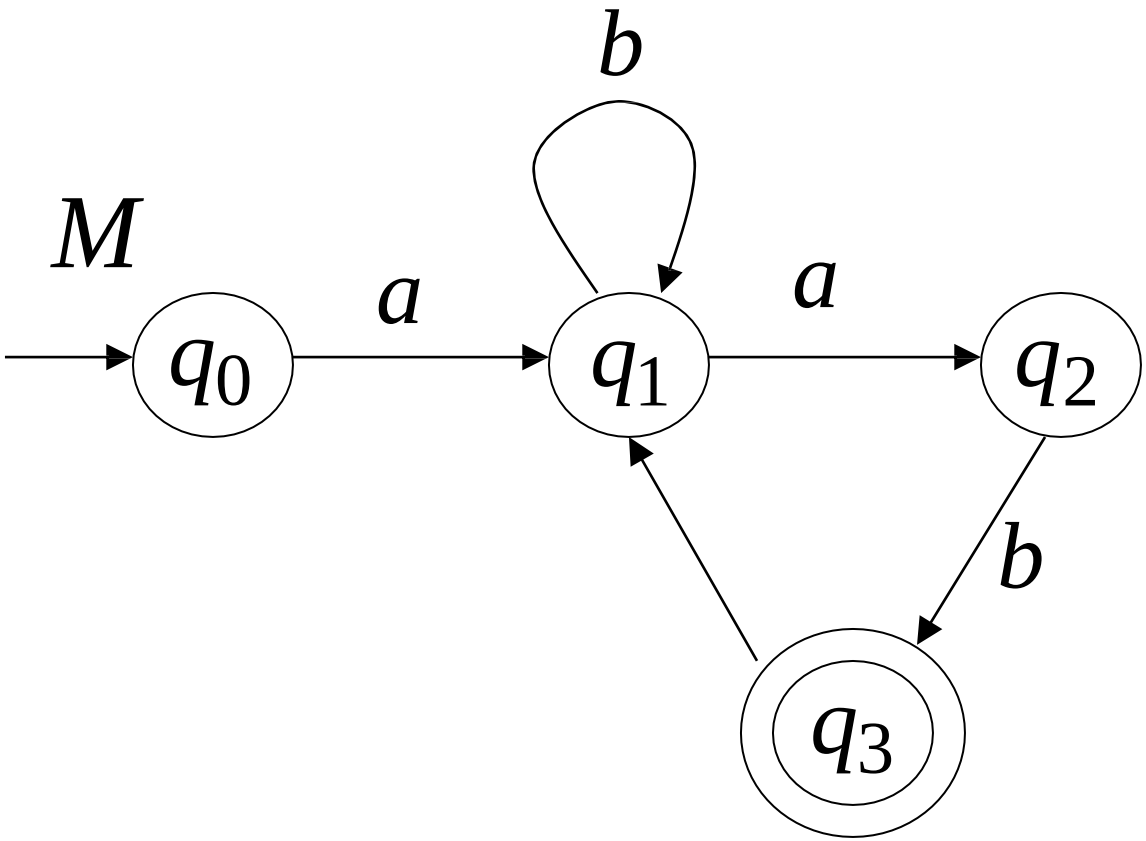
$q_1 \to bq_1$

$q_1 \to aq_2$

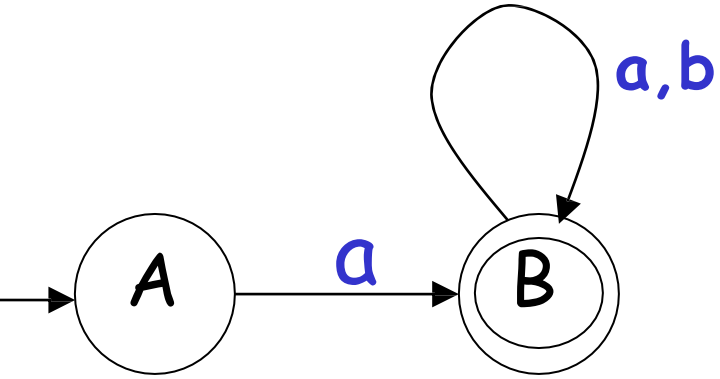
$q_2 \to bq_3$

$q_3 \to q_1$

$q_3 \to \lambda$



- Since G is right-linear grammar
-
- G is also a regular grammar
- with $L(G) = L(M) = L$



- $A \rightarrow aB$
- $B \rightarrow aB|bB|\lambda$

The case of Left-Linear Grammars

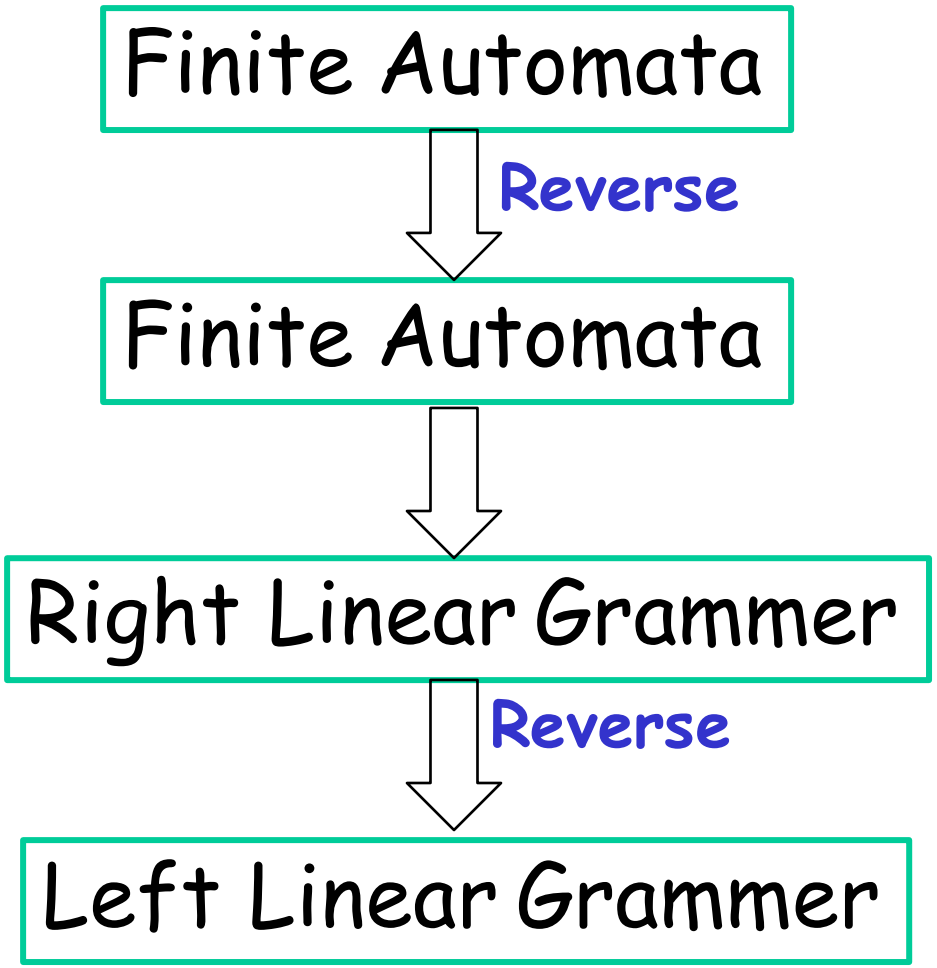
Any regular language L is generated by some regular grammar G

Proof idea:

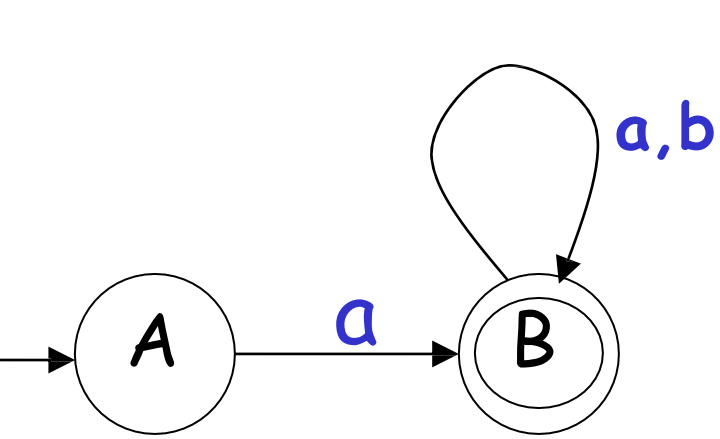
Let M be the NFA with $L = L(M)$.

Construct from M a regular grammar G such that $L(M) = L(G)$

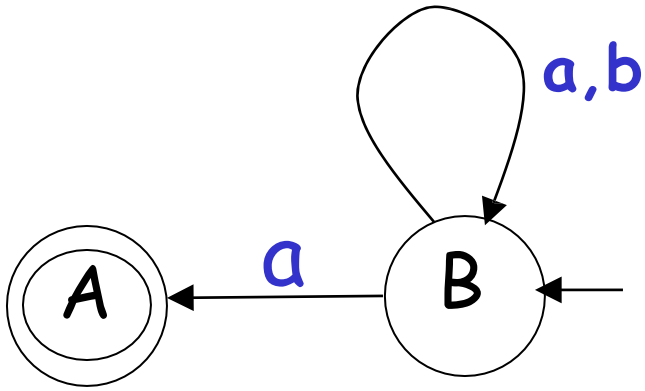
Procedure to Prove



Convert FA to its Equivalent LLG



FA (reverse)



RLG

- $B \rightarrow aA|bB|aB$
- $A \rightarrow \lambda$

LLG(reverse)

- $B \rightarrow Aa|Bb|Ba$
- $A \rightarrow \lambda$

Recall

- Linear Grammer
- Non-Linear Grammer
- Right Linear Grammer
- Left Linear Grammer
- Regular Grammer
- Equivalence of RG and FA
- Conversion of RLG to NFA
- Conversion of LLG to NFA

Equivalence of FA and RG

- RG ❼ FA
  - R L G ❼F A
  - L L G ❼F A (L L G ❼ Reverse of L L G ❼F A ❼ Reveses of FA)

- F A ❼R G
  - F A ❼R L G
  - F A ❼L L G (F A ❼ Reverse F A ❼R G ❼ [illegible]f RG)

Conversion of RLG to LLG

- 1. Create FA for RLG
- 2. Reverse the FA Generated from step 1
- 3.Find out the LLG from FA Generated from step 2. OR find the RLG from step 2 FA and Reverse the Resultant Grammer.

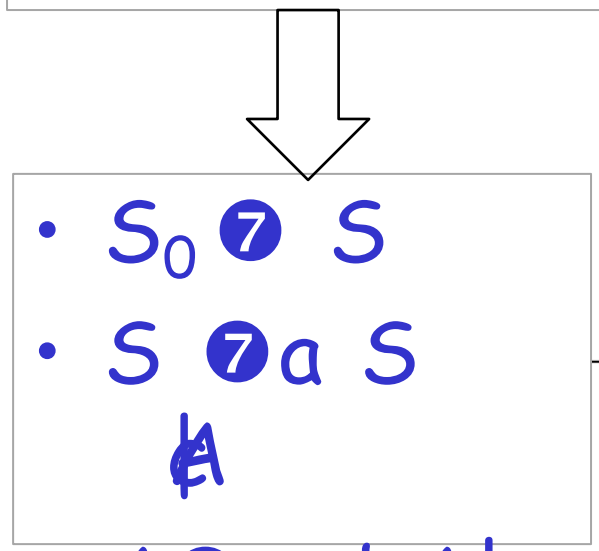
May need to make a new start symbol

The algorithm on the following slides assume that the left/Right linear grammar doesn't have any rules with the start symbol on the right hand side.

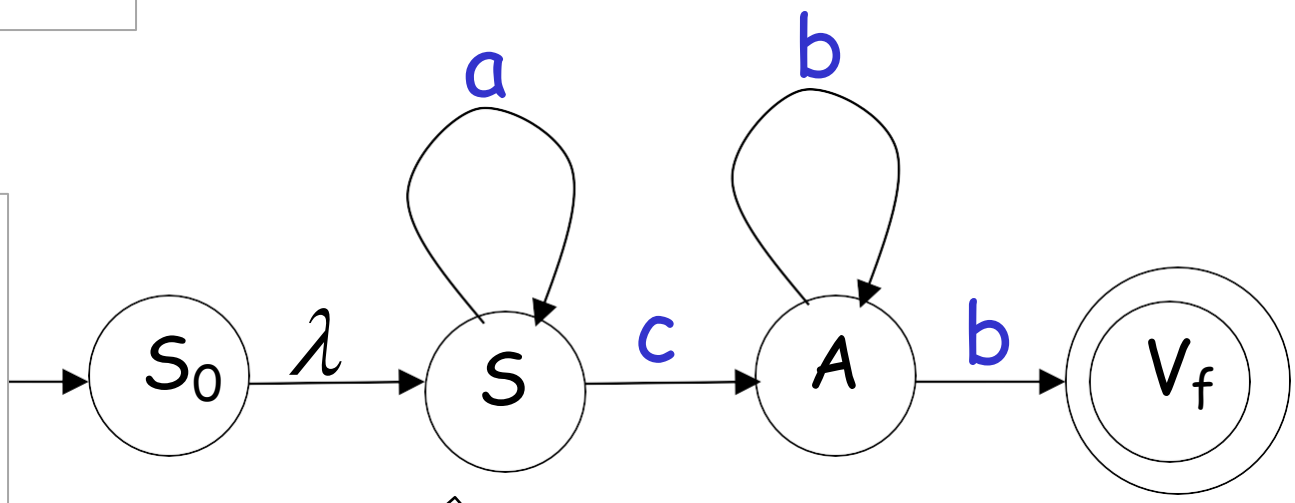
- If the left linear grammar has a rule with the start symbol S on the right hand side, simply add this rule:

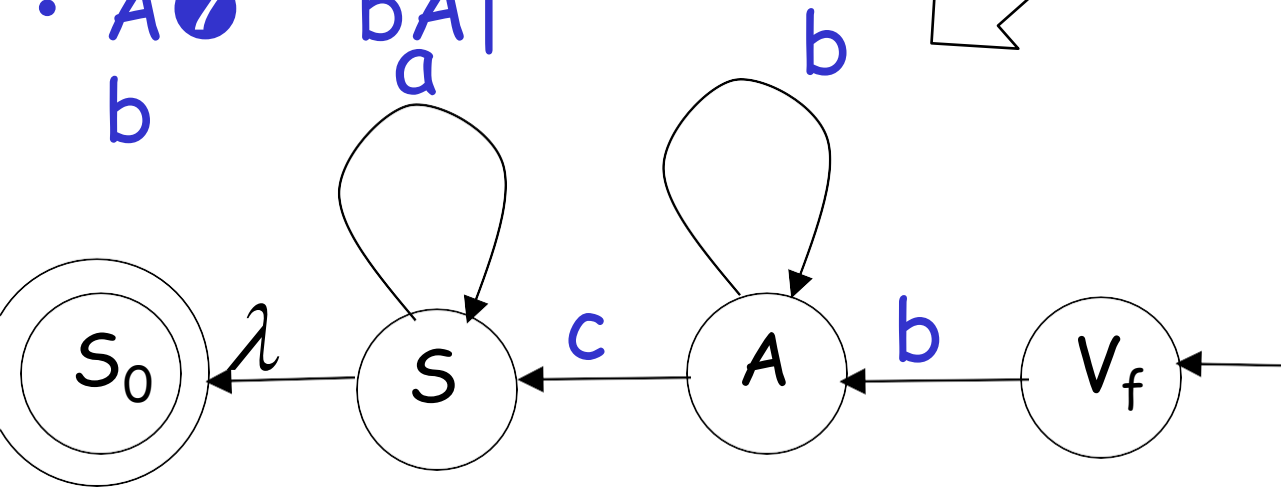
  $S_0 \rightarrow S$

- S → a S | cA
- A → b A | b



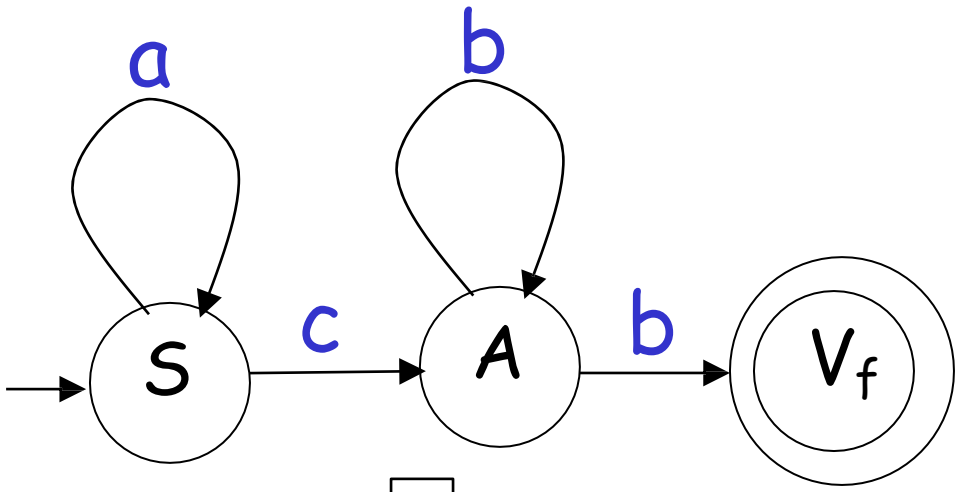
- S_0 → S
- S → a S
  | cA
- A → b A |
  b



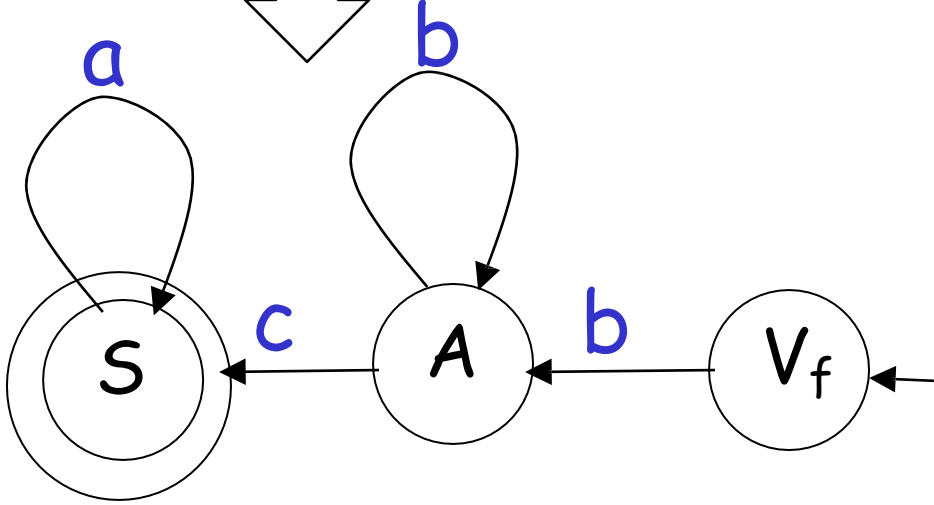


- V_f → A b
- A → A b | c
- S → S a | a
- S_0 → λ

- S → a S | A
- A → b A | b

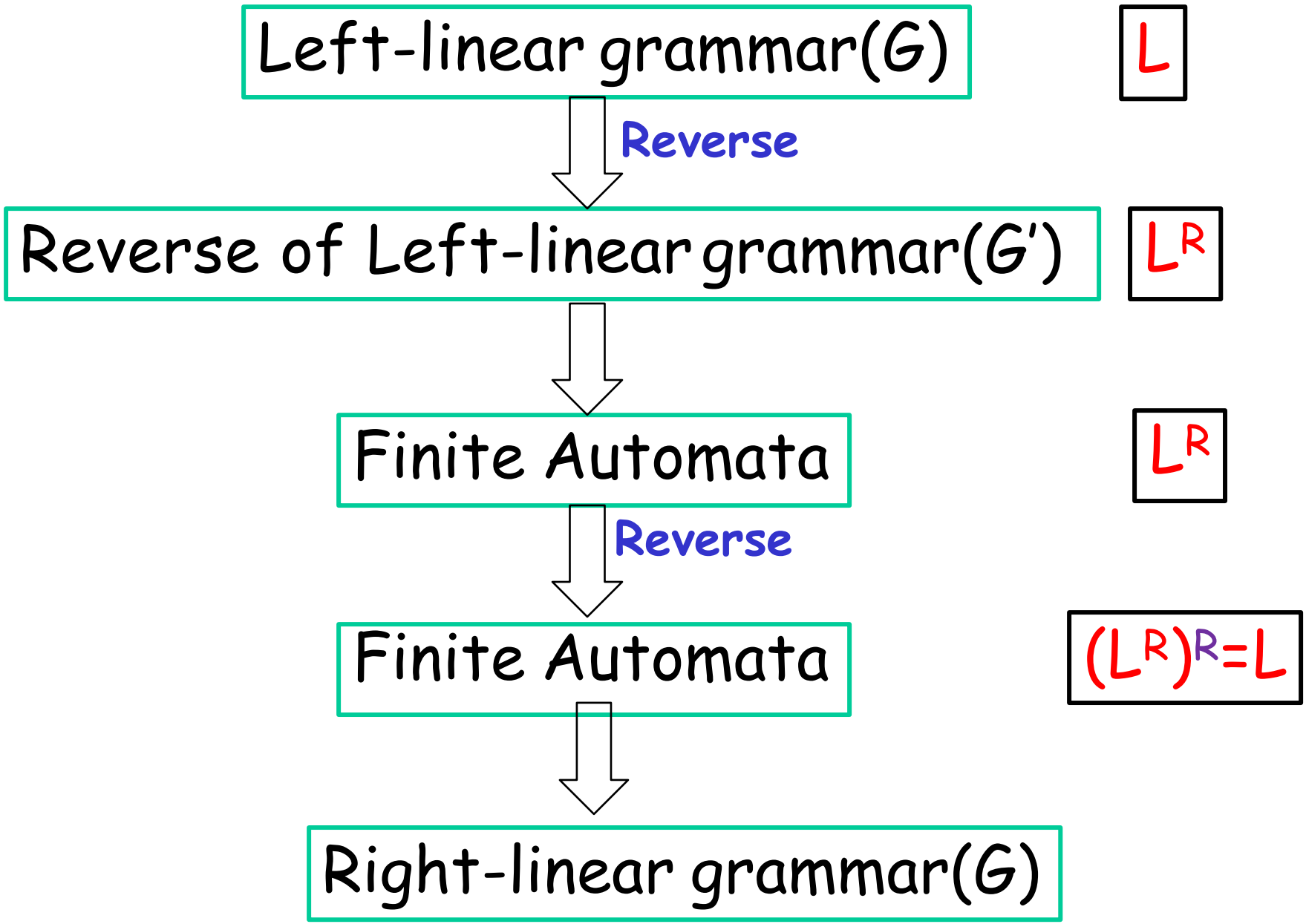


- V_f → bA
- A → b A | b
- S → a S | λ



- V_f → A b
- A → A b | Sc
- S → S a | λ

Conversion of LLG to RLG



Conversion of LLG to RLG

- Convert the given left linear grammar GL to its equivalent right linear grammar GR.
- S ➔ Bl / A0 / C0
- A ➔ C0 / Al / Bl / 0
- B ➔ Bl / l
- C ➔ A0

Aug-2017, Insem , 4 marks

Problem (May-2018 EndSem, 5 Marks)

- Write an equivalent left-linear grammar for the right-linear grammar which is defined as
- $S \rightarrow 0A \mid 1B$
- $A \rightarrow 0C \mid 1A \mid 0$
- $B \rightarrow 1B \mid 1A \mid 1$
- $C \rightarrow 0 \mid 0A$

Problem (Aug-2015 InSem, 4 Marks)

- Convert Right Linear Grammar to equivalent Left Linear grammar
- S->bB
- B->bC
- B->aB
- C->a
- B->b