Mohsin Iban Hossain

AIUB, Theory of Computation Notes

THEORY OF COMPUTATION

Table of Contents

CONTEXT FREE GRAMMAR (CFG)

Context Free Grammar (CFG)

⇨ **A context Free Grammar (CFG) is a 4-tuple such that-**

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

where-

- **V** = Finite non-empty set of variables / non-terminal symbols
- **T/Σ** = Finite set of terminal symbols
- **P/R** = Finite non-empty set of production rules of the form $A \rightarrow \alpha$ where $A \in V$ and $\alpha \in (V \cup T)^*$
- **S** = Start symbol $(S \in V)$

Applications:

- Context Free Grammar (CFG) is of great practical importance. It is used for following purposes-
- For defining programming languages
- For parsing the program by constructing syntax tree
- For translation of programming languages
- For describing arithmetic expressions
- For construction of compilers

Context Free Language (CFL)

⇨ The language generated using Context Free Grammar is called as **Context Free Language**.

Properties:

⇨ If L1 and L2 are two context free languages, then-

- **L1 ∪ L2** is also a context free language.
- **L1.L2** is also a context free language.
- **L1* & L2*** are also context free languages.
- **L1 ∩ L2** is not a context free language.
- **L1′ & L2′** are not context free languages.
- Each Context Free Language is accepted by a **Pushdown automaton.**

Example of CFG

Example 1:

⇨ Construct a CFG for $\{0^n 1^n (n \geq 1)\}$

» Let's break down the conditions:

- if, $n = 1$, → $0^1 1^1 = \{01\}$
- if, $n = 2$, → $0^2 1^2 = \{0011\}$
- if, $n = 3$, → $0^3 1^3 = \{000111\}$ ...

so, CFG:

S → 01 | 0S1 OR S → 01
S → 0S1

∴ **V** = {S}

∴ **T/Σ** = {0,1}

∴ **P/R** = {S → 01 , S → 0S1}

∴ **S** = {S}

» Let's Check 000111 string Accepted or Rejected:

S → 0S1

S → 00S11

S → 000111

∴ Accepted

Example 2:

⇨ Identify the terminal, non-terminal, Start variable for the following grammar.

1. E → E + T | T
   T → T * F | F
   F → (E) | id

so, From CFG

∴ **V** = {E, T, F}

∴ **T/Σ** = {+, *, (,), id}

∴ **S** = {E}

2. S → (L) | a
   L → L, S | S

so, From CFG

∴ **V** = {S, L}

∴ **T/Σ** = {(,), ,}

∴ **S** = {S}

Designing CFG

Problem 1:

⇨ **L(M) = {w| w = $a^n (n \geq 0)$}**

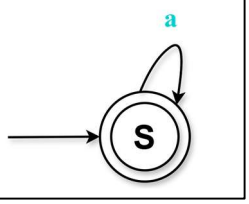
➔ **Is Given,**

∑ = {a},
L(M) = {ε,a, aa, aaa, aaaa,….}

so, CFG:

S → aS | ε

If we create a DFA:



≫ Let's Check aaaa string Accepted or Rejected:

S → aS

S → aaS

S → aaaS

S → aaaaS

S → aaaaε

S → aaaa

∴ Accepted

Problem 2:

⇨ **L(M) = {w| w = $a^n (n \geq 1)$}**

➔ **Is Given,**

∑ = {a},
L(M) = {a, aa, aaa, aaaa,….}

so, CFG:

S → aS | a

℘ Problem 3:

⇨ **L(M) = {w| w Set of all string over a,b}**

➔ **Is Given,**

$\sum = \{a,b\}$,
L(M) = { ε,ab, aab, aabbb, aaaabbbb,….}

so, CFG:

S → aS | bS | ε

» Let's Check aaab string Accepted or Rejected:

S → aS

S → aaS

S → aaaS

S → aaabS

S → aaabε

S → aaab

∴Accepted

℘ Problem 4:

⇨ **L(M) = {w| w Set of all string over a,b which length at least 2}**

➔ **Is Given,**

$\sum = \{a,b\}$,
L(M) = {ab, aab, aabbb, aaaabbbb,….}

$$\therefore RE = \underbrace{(a \cup b)}_{A}\ \underbrace{(a \cup b)}_{A}\ \underbrace{(a \cup b)^*}_{B}$$

so, CFG:

S → AAB

A→ a | b

B→ aB | bB | ε

» Let's Check aab string Accepted or Rejected:

S → AAB
S → aAB
S → aaB
S → aabB
S → aabε
S → aab ∴Accepted

Problem 5:

⇨ **L(M) = {w| w Set of all string over 0,1 which contain at least three 0's}**

➔ **Is Given,**

$\sum = \{0,1\}$,
L(M) = {000, 0001, 000011, 00011,….}

$$\therefore RE = \underbrace{1^*}_{A}\,0\,\underbrace{1^*}_{A}\,0\,\underbrace{1^*}_{A}\,0\,\underbrace{(0 \cup 1)^*}_{B}$$

so, CFG:

S → A0A0A0B

A→ 1A | ε

B→ 0B | 1B | ε

Let's Check 0011 string Accepted or Rejected:

S → A0A0A0B
S → ε0ε0ε01B
S → 00011B
S → 00011ε
S → 00011
∴ Rejected

Problem 6:

⇨ **L(M) = {w| w Set of all string over a,b which length at Most 2}**

➔ **Is Given,**

$\sum = \{a,b\}$,
L(M) = {ε, a, b, ab, aa, ba, bb}

$$\therefore RE = \underbrace{(a \cup b \cup \varepsilon)}_{A}\,\underbrace{(a \cup b \cup \varepsilon)}_{A}$$

so, CFG:

S → AA

A→ a | b | ε

» Let's Check aab string Accepted or Rejected:

S → AA
S → aA
S → aa
∴ Rejected

∅ Problem 7:

⇨ **L(M) = {w | *w* begins with *a* and ends with *b*}**

➔ **Is Given,**

$\sum$ = {a,b},
L(M) = {ab, aab, abb, aabb,…}

$$\therefore \boldsymbol{RE} = \boldsymbol{a}\ \underbrace{(\boldsymbol{a} \cup \boldsymbol{b})^*}_{\boldsymbol{A}}\ \boldsymbol{b}$$

so, CFG:

S → aAb

A→ aA | bA | ε

≫ Let's Check aab string Accepted or Rejected:

S → aAb
S → aAb
S → aaAb
S → aaεb
S → aab

∴Accepted

∅ Problem 8:

⇨ **L(M) = { *w* | *w* begins with *b* or ends with *a* }**

➔ **Is Given,**

$\sum$ = {a,b},
L(M) = {a, b, aa, bb, aba, bab, ba, baa,…}

$$\therefore \boldsymbol{RE} = \boldsymbol{b}\ \underbrace{(\boldsymbol{a} \cup \boldsymbol{b})^*}_{\boldsymbol{T}} \cup \underbrace{(\boldsymbol{a} \cup \boldsymbol{b})^*}_{\boldsymbol{T}}\ \boldsymbol{a}$$

so, CFG:

S → bT | Ta

T→ aT | bT | ε

≫ Let's Check bab string Accepted or Rejected:

S → bT | Ta
S → baT
S → babT
S → babε
S → bab

∴Accepted

Problem 9:

⇨ **L(M) = { *w* | *w* contains at least 3 *a*'s }**

➔ **Is Given,**

∑ = {a,b},
L(M) = {aaa, baaa, bababa,…}

$$\therefore RE = \underbrace{b^*}_{P}\, a\, \underbrace{b^*}_{P}\, a\, \underbrace{b^*}_{P}\, a\, \underbrace{(a \cup b)^*}_{T}$$

so, CFG:

S → PaPaPaT

P → bP | ε

T→ aT | bT | ε

Problem 10:

⇨ **L(M) = { *w* | *w* contains at most 3 *a*'s }**

➔ **Is Given,**

∑ = {a,b},
L(M) = { ε, a,aa, aaa, baaa, bababab,…}

$$\therefore RE = \underbrace{b^*}_{B}\, \underbrace{(a \cup \varepsilon)}_{A}\, \underbrace{b^*}_{B}\, \underbrace{(a \cup \varepsilon)}_{A}\, \underbrace{b^*}_{B}\, \underbrace{(a \cup \varepsilon)}_{A}\, \underbrace{b^*}_{B}$$

so, CFG:

S → BABABAB

B → bB | ε

A→ a | ε

Problem 11:

⇨ **L(M) = { *w* | *w* contains the substring *aba* }**

➔ **Is Given,**

∑ = {a,b},
L(M) = {aba, aaba, bbbaba, abababa,…}

$$\therefore RE = \underbrace{(a \cup b)^*}_{T}\, aba\, \underbrace{(a \cup b)^*}_{T}$$

so, CFG:

S → TabaT

T→ aT | bT | ε

Problem 12:

⇨ **L(M) = { *w* | even length string }**

➔ **Is Given,**

∑ = {a,b},
L(M) = { ε , aa, abab, ababaa,….}

$$\therefore RE = \underbrace{(\underbrace{(a \cup b)}_{T}\ \underbrace{(a \cup b)}_{T})^*}_{P}$$

so, CFG:

S → PS | ε

P → TT

T→ a | b

Problem 13:

⇨ **L(M) = { *w* | odd length string }**

➔ **Is Given,**

∑ = {a,b},
L(M) = { ε , a, aba, ababa,….}

$$\therefore RE = \underbrace{(\underbrace{(a \cup b)}_{T}}_{}\ \underbrace{(\underbrace{(a \cup b)}_{T}\ \underbrace{(a \cup b)}_{T})^*}_{P}$$

so, CFG:

S → TP

P → TTP | ε

T→ a | b

Problem 14:

⇨ **L(M) = { *w* | every odd position of *w* is a }**

➔ **Is Given,**

∑ = {a,b},
L(M) = { ε , aaa, abab, ababaa,….}

so, CFG:

S →aT | ε

T→ aS | bS

Problem 15:

⇨ **L(M) = { w | $w = a^m b^n$ and m = n}**

➔ **Is Given,**

∑ = {a,b},
L(M) = { ε , ab, aabb, aaabbb,….}

so, CFG:

S →aSb | ε

Problem 16:

1. Write the Context-free grammar for Any TWO of the following languages where ∑ = {x,y}
   i. A={ w | each 'x' in w is followed by at least two 'y' }
   ii. A={ w | starts and ends with same symbols }
   iii. A={ w | w contains 'xyxyxy' as substring}

(i)

➔ **Is Given,**

∑ = {x,y},
L(M) = { ε, y, yy, xyy, xyyy, yyyxyy,…}

$$\therefore RE = \underbrace{y^*}_{T}\, xyy\, \underbrace{y^*}_{T}\, \underbrace{(xyy\, \underbrace{y^*}_{T})^*}_{P} \cup \underbrace{y^*}_{T} \cup \varepsilon$$

so, CFG:

S →TxyyTP | T | ε

P→xyyTP | ε

T →yT | ε

(ii)

➔ **Is Given,**

∑ = {x,y},
L(M) = { xyyx, yyy, yyyxyy,…}

$$\therefore RE = x\, \underbrace{(x \cup y)^*}_{T} x \cup y\, \underbrace{(x \cup y)^*}_{T} y \cup x \cup y \cup \varepsilon$$

so, CFG:

S →xTx | yTy | x | y | ε

T →xT | yT | ε

(ii)

➔ **Is Given,**

∑ = {x,y},
L(M) = { xyxyxy, yyy xyxyxy, xyxyxyyyyxyy,…}

$$\therefore RE = \underbrace{(x \cup y)^*}_{T}\, xyxyxy\, \underbrace{(x \cup y)^*}_{T}$$

so, CFG:

S →T*xyxyxy*T

T →xT | yT | ε

AMBIGUOUS GRAMMAR

Ambiguous Grammar

⇨ A grammar is said to ambiguous if for any string generated by it, it produces more than one-

- Parse tree
- Or derivation tree
- Or syntax tree
- Or leftmost derivation
- Or rightmost derivation

℘ Example 1:

⇨ Consider the following grammar-

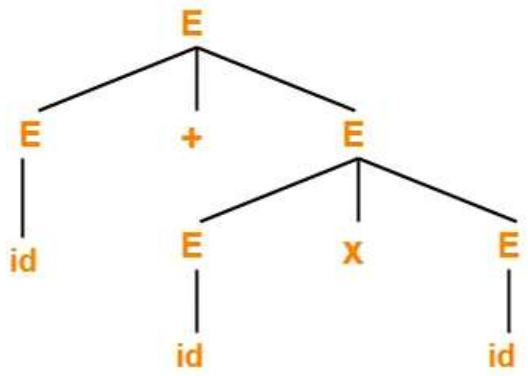
$$E \rightarrow E + E \mid E \times E \mid id$$

Reason-01:

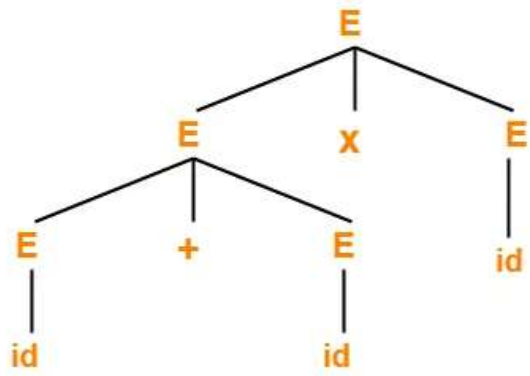
Let us consider a string w generated by the grammar-

w = id + id × id

Now, let us draw the parse trees



Parse Tree-01



Parse Tree-02

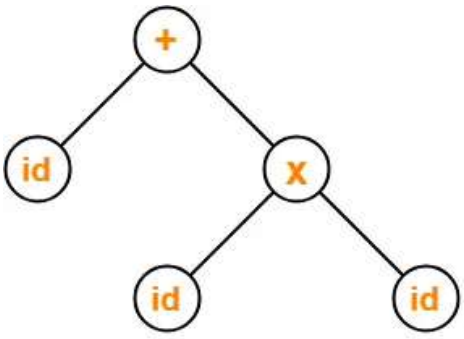
Since two parse trees exist for string w, therefore the grammar is ambiguous.

Reason-02:

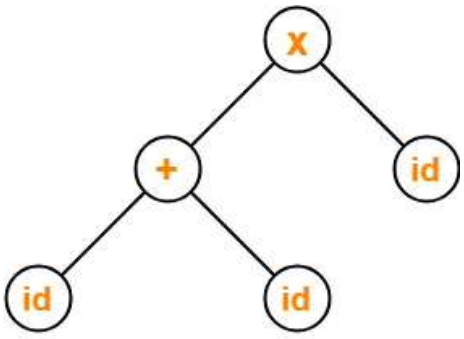
Let us consider a string w generated by the grammar-

w = id + id × id

Now, let us draw the syntax trees



Syntax Tree-01



Syntax Tree-02

Since two syntax trees exist for string w, therefore the grammar is ambiguous.

Reason-03:

Let us consider a string w generated by the grammar-

w = id + id × id

Now, let us write the leftmost derivations

$E \rightarrow \mathbf{E} + E$

$\rightarrow id + \mathbf{E}$

$\rightarrow id + \mathbf{E} \times E$

$\rightarrow id + id \times \mathbf{E}$

$\rightarrow id + id \times id$

Leftmost Derivation-01

$E \rightarrow \mathbf{E} \times E$

$\rightarrow \mathbf{E} + E \times E$

$\rightarrow id + \mathbf{E} \times E$

$\rightarrow id + id \times \mathbf{E}$

$\rightarrow id + id \times id$

Leftmost Derivation-02

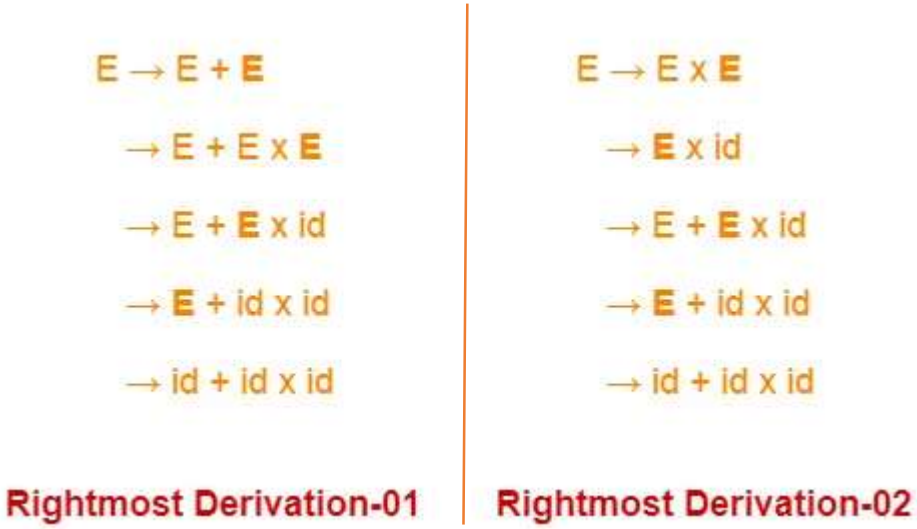
Since two leftmost derivations exist for string w, therefore the grammar is ambiguous.

Reason-04:

Let us consider a string w generated by the grammar-

w = id + id × id

Now, let us write the rightmost derivations



Since two rightmost derivations exist for string w, therefore the grammar is ambiguous.

Problem 1:

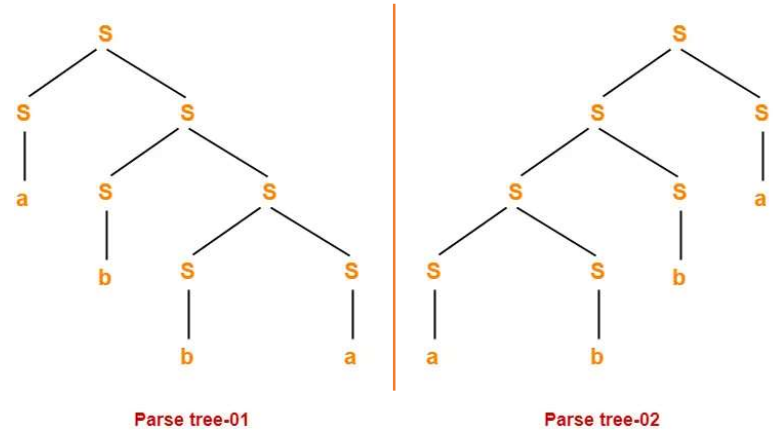
⇨ Check whether the given grammar is ambiguous or not-

S → SS | a | b

➔ **Let us,**

⇨ consider a string w generated by the given grammar- w = abba

Now, draw the parse trees



Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem 2:

⇨ Check whether the given grammar is ambiguous or not-

S → A / B

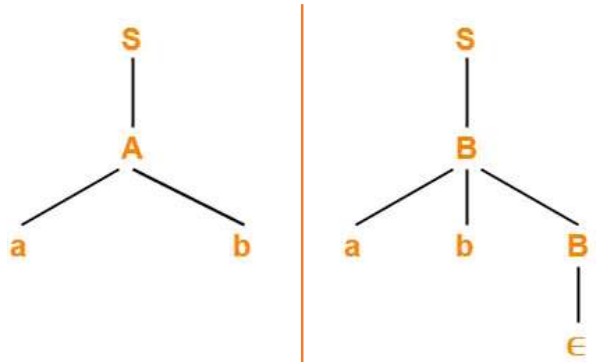
A → aAb / ab

B → abB / ∈

➔ **Let us,**

⇨ consider a string w generated by the given grammar- w = ab

Now, draw the parse trees



Parse tree-01 Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem 3:

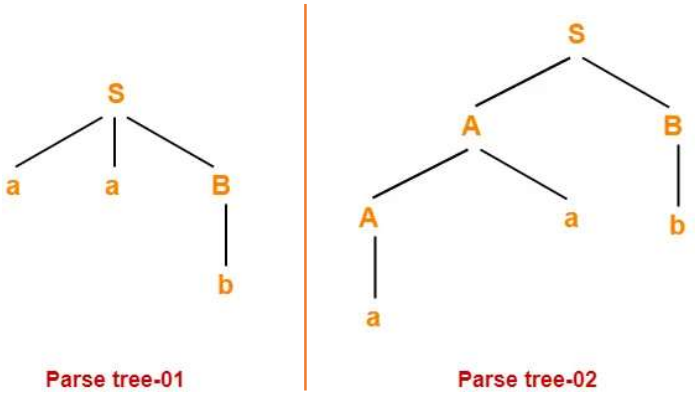
⇨ Check whether the given grammar is ambiguous or not-

S → AB / aaB
A → a / Aa
B → b

➔ **Let us,**

⇨ consider a string w generated by the given grammar- w = aab

Now, draw the parse trees



Parse tree-01 Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem 4:

⇨ Check whether the given grammar is ambiguous or not-

S → AB / C

A → aAb / ab

B → cBd / cd

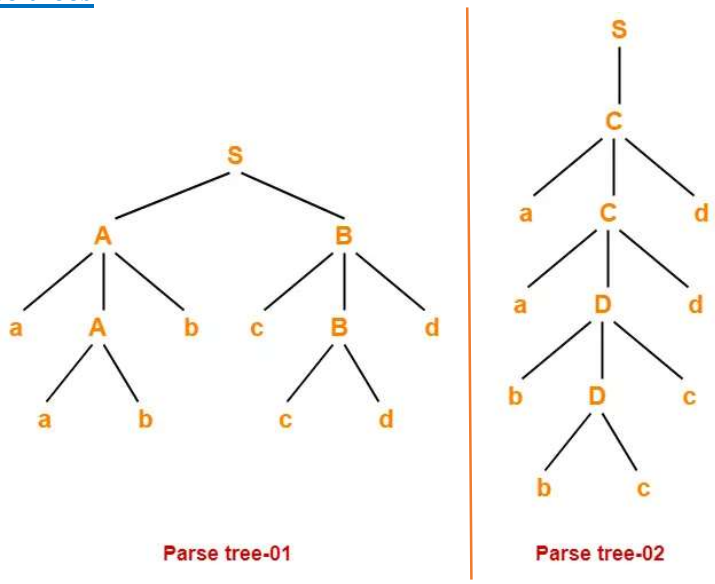
C → aCd / aDd

D → bDc / bc

➔ **Let us,**

⇨ consider a string w generated by the given grammar- w = aabbccdd

Now, draw the parse trees



Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem 5:

⇨ Check whether the given grammar is ambiguous or not-

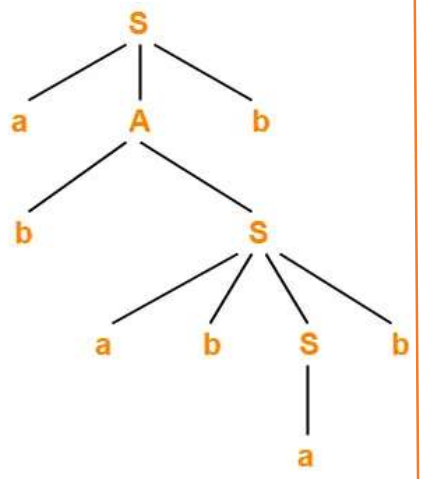
S → a / abSb / aAb

A → bS / aAAb

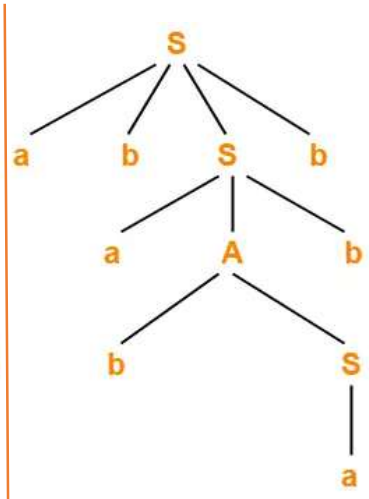
➔ **Let us,**

⇨ consider a string w generated by the given grammar- w = abababb

Now, draw the parse trees



Parse tree-01



Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem 6:

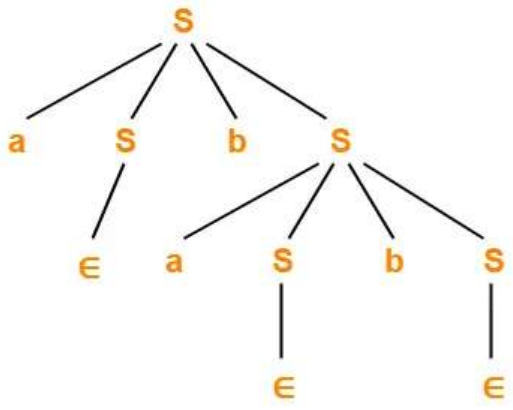
⇨ Check whether the given grammar is ambiguous or not-

$S \rightarrow aSbS / bSaS / \in$

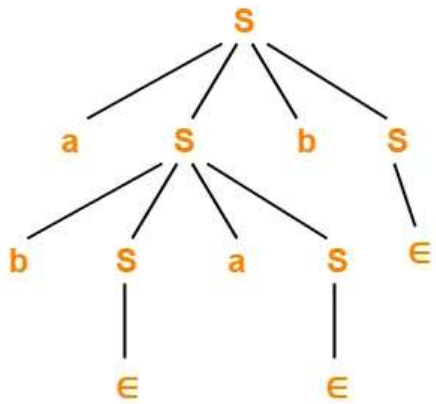
➔ Let us,

⇨ consider a string w generated by the given grammar- w = abab

Now, draw the parse trees



Parse tree-01



Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

Problem 7:

⇨ Check whether the given grammar is ambiguous or not-

$$E \rightarrow E + T / T$$
$$T \rightarrow T \times F / F$$
$$F \rightarrow id$$

➔ There exists no string belonging to the language of grammar which has more than one parse tree.
Since a unique parse tree exists for all the strings, therefore the given grammar is unambiguous.

Problem 8:

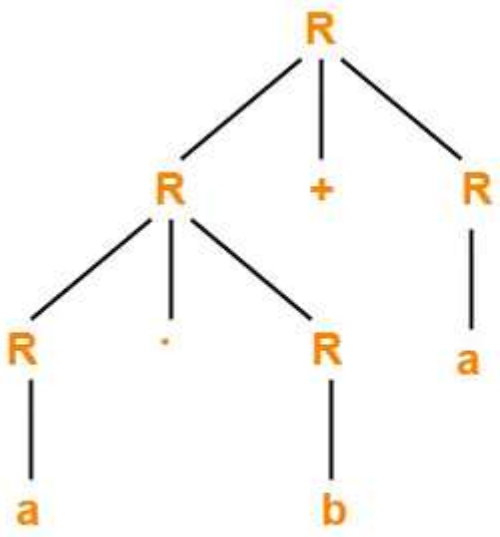
⇨ Check whether the given grammar is ambiguous or not-

$$R \rightarrow R + R / R \,.\, R / R^* / a / b$$

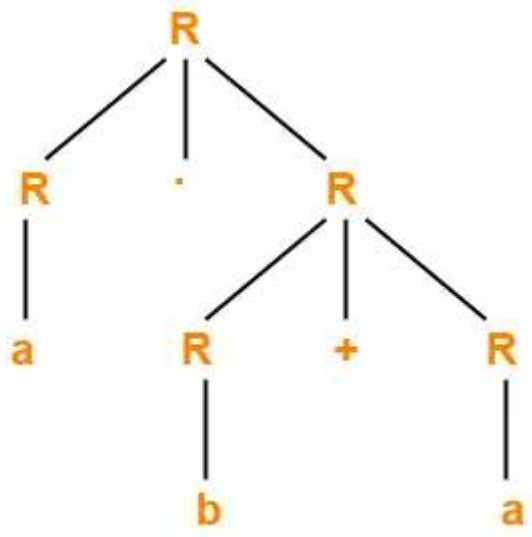
➔ **Let us,**

⇨ consider a string w generated by the given grammar- w = ab + a

Now, draw the parse trees



Parse tree-01



Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

CHOMSKY NORMAL FORM (CNF)

Chomsky Normal Form (CNF)

⇨ A context free grammar is said to be in chomsky normal form (CNF) if all its productions are of the form-

$$A \rightarrow BC \text{ or}$$

$$A \rightarrow a$$

where A, B, C are non-terminals and a is a terminal.

Rules of CNF,

- To be in CNF, all the productions must derive either **two non-terminals** or a **single terminal.**
- CNF restricts the number of symbols on the right side of a production to be two.
- The two symbols must be non-terminals or a single terminal.

Steps- to standardize the grammar using CNF

➢ Steps-01:

- If the Start Symbol S occurs on some right side, create a new Start Symbol S' and a new Production S'→S.

➢ Steps-02:

Reduce the grammar completely by-

- Eliminating ∈ productions
- Eliminating unit productions
- Eliminating useless productions

➢ Steps-03:

⇨ Replace each production of the form $A \rightarrow B_1B_2B_3....B_n$ where $n > 2$ with $A \rightarrow B_1C$ where $C \rightarrow B_2B_3....B_n$.

⇨ Repeat this step for all the productions having more than two variables on RHS.

➢ Steps-04:

⇨ Replace each production of the form $A \rightarrow aB$ with $A \rightarrow XB$ and $X \rightarrow a$.

⇨ Repeat this step for all the productions having the form $A \rightarrow aB$.

Eliminating ∈ productions

Step 1: To remove A→∈, look for all productions whose right side contains A

Step2: Replace each occurrence of 'A' in each of these productions with ∈

Step 3: Add the resultant productions to the Grammar

Example 1:

⇨ Remove the **null** production from the following grammar:

$S \rightarrow ABAC$

$A \rightarrow aA \mid \in$

$B \rightarrow bB \mid \in$

$C \rightarrow c$

➔ **Is Given,** ∈ productions,

A → ∈ , B → ∈

❖ To Eliminate A → ∈

≫ $S \rightarrow ABAC$

≫ $S \rightarrow AB \in C \rightarrow ABC$

≫ $S \rightarrow \in BAC \rightarrow BAC$

≫ $S \rightarrow \in B \in C \rightarrow BC$

∴ $S \rightarrow ABC \mid BAC \mid BC$

≫ $A \rightarrow aA$

≫ $A \rightarrow a \in \rightarrow a$

∴ New Production

≫ S → ABAC | ABC | BAC | BC

≫ A → $aA \mid a$

≫ B → $bB \mid \in$

≫ C → c

❖ To Eliminate B →∈

≫ $S \rightarrow$ ABAC | ABC | BAC | BC

≫ $S \rightarrow$ ABAC → $A \in AC \rightarrow AAC$

≫ $S \rightarrow$ ABC → $A \in C \rightarrow AC$

≫ $S \rightarrow$ BAC →∈ $AC \rightarrow AC$

≫ $S \rightarrow$ BC →∈ $C \rightarrow C$

∴ S → AAC | AC | C

≫ B → bB

≫ $B \rightarrow a \in \rightarrow$ b

∴ New Production

≫ S → $ABAC$ | ABC | BAC | BC |AAC | AC | C

≫ A → $aA \mid a$

≫ B → $bB \mid b$

≫ C → c

Eliminating unit productions

⇨ Any Production Rule of the form A→B where A, B = Non-Terminals is called Unit Production

➢ **Procedure for Removal**

Step 1: To remove **A→B**, add production $A \rightarrow x$ to the grammar rule whenever $B \rightarrow x$ occurs in the grammar. [x ∈ Terminal, x can be Null]

Step 2: Delete **A→B** from the grammar.

Step 3: Repeat from Step 1 until all Unit Productions are removed.

Example 1:

⇨ Remove **Unit** production from the following grammar:

$$S \rightarrow XY$$
$$X \rightarrow a$$
$$Y \rightarrow Z|b$$
$$Z \rightarrow M$$
$$M \rightarrow N$$
$$N \rightarrow a$$

➔ **Is Given,** unit productions:

$$Y \rightarrow Z, \quad Z \rightarrow M, \quad M \rightarrow N$$

❖ To Eliminate $M \rightarrow N$:

⇨ Since $N \rightarrow a$, We add $M \rightarrow a$:

$$\therefore S \rightarrow XY, X \rightarrow a, Y \rightarrow Z|b,\ Z \rightarrow M,\ M \rightarrow a, N \rightarrow a$$

❖ To Eliminate $Z \rightarrow M$:

⇨ Since $M \rightarrow a$, We add $Z \rightarrow a$:

$$\therefore S \rightarrow XY, X \rightarrow a, Y \rightarrow Z|b,\ Z \rightarrow a,\ M \rightarrow a, N \rightarrow a$$

❖ To Eliminate $Y \rightarrow Z$:

⇨ Since $Z \rightarrow a$, We add $Y \rightarrow a$:

$$\therefore S \rightarrow XY, X \rightarrow a, Y \rightarrow a|b,\ Z \rightarrow a,\ M \rightarrow a, N \rightarrow a$$

Eliminating useless productions

Example 1:

⇨ Remove **useless** production from the following grammar:

$S \rightarrow XY,$

$X \rightarrow a,$

$Y \rightarrow a|b,$

$Z \rightarrow a,$

$M \rightarrow a,$

$N \rightarrow a$

➔ **Is Given,** useless productions:

$Z \rightarrow a, \quad M \rightarrow a, \quad N \rightarrow a$

❖ To Eliminate $Z \rightarrow a, M \rightarrow a, N \rightarrow a$:

$\therefore S \rightarrow XY, X \rightarrow a, Y \rightarrow a|b$

Example 1:

S → AB

A → a

B → b

This context free grammar is in chomsky normal form.

Problem 1:

⇨ Convert the given grammar to CNF-

S → aAD

A → aB | bAB

B → b

D → d

Steps-01:

⇨ The given grammar is already completely reduced.

Steps-02: Find in CNF & not in CNF

⇨ The productions **already in chomsky** normal form are-

- $B \rightarrow b$ ……..(1)
- $D \rightarrow d$ ……..(2)

These productions will remain as they are.

⇨ The productions **not in chomsky** normal form are-

- $S \rightarrow aAD$ ……..(3)
- $A \rightarrow aB \mid bAB$ ……..(4)

We will convert these productions in chomsky normal form.

Steps-03: Replace the terminal symbols **a** & **b** by new variables C_a and C_b.

⇨ This is done by introducing the following two new productions in the grammar-

- $C_a \rightarrow a$ ……..(5)
- $C_b \rightarrow b$ ……..(6)

⇨ Now, the productions (3) and (4) modifies to-

- $S \rightarrow C_aAD$ ……..(7)
- $A \rightarrow C_aB \mid C_bAB$ ……..(8)

Steps-04: Replace **AD** and **AB** by new variables C_{AD} and C_{AB} respectively.

⇨ This is done by introducing the following two new productions in the grammar-

- $C_{AD} \rightarrow AD$ ……..(9)
- $C_{AB} \rightarrow AB$ ……..(10)

⇨ Now, the productions (7) and (8) modifies to-

- $S \rightarrow C_aC_{AD}$ ……..(11)
- $A \rightarrow C_aB \mid C_bC_{AB}$ ……..(12)

Steps-05: From (1), (2), (5), (6), (9), (10), (11) and (12), the resultant grammar is-

$S \rightarrow C_aC_{AD}$
$A \rightarrow C_aB \mid C_bC_{AB}$
$B \rightarrow b$
$D \rightarrow d$
$C_a \rightarrow a$
$C_b \rightarrow b$
$C_{AD} \rightarrow AD$
$C_{AB} \rightarrow AB$

This grammar is in chomsky normal form.

℘ Problem 2:

⇨ Convert the given grammar to CNF-

$S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \in$

Step 1: Eliminating ∈ productions

➔ **Is Given,** ∈ productions,

$B \rightarrow \in$

Let,

$S_0 \rightarrow S$

❖ To Eliminate B →∈

≫ $S \rightarrow ASA \mid aB$

≫ $S \rightarrow aB$

≫ $S \rightarrow aB \rightarrow a \in\rightarrow a$

∴ $S \rightarrow ASA \mid aB \mid a$

≫ $A \rightarrow B \mid S$

≫ $A \rightarrow B$

≫ $A \rightarrow \in$

∴ $A \rightarrow B \mid S \mid \in$

∴ $B \rightarrow b$

∴ New Production

≫ $S_0 \rightarrow S$

≫ $S \rightarrow ASA \mid aB \mid a$

≫ $A \rightarrow B \mid S \mid \in$

≫ $B \rightarrow b$

➔ **Now,** ∈ productions,

$A \rightarrow \in$

❖ To Eliminate A →∈

≫ $S \rightarrow ASA \mid aB \mid a$

≫ $S \rightarrow ASA$

≫ $S \rightarrow AS \in\rightarrow AS$

≫ $S \rightarrow \in SA \rightarrow SA$

≫ $S \rightarrow \in S \in\rightarrow S$

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

∴ $A \rightarrow B \mid S$

∴ $B \rightarrow b$

∴ New Production

≫ $S_0 \rightarrow S$

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

≫ $A \rightarrow B \mid S$

≫ $B \rightarrow b$

Step 2: Eliminating unit productions

➔ **Is Given,** New Production

≫ $S_0 \rightarrow S$
≫ $S \rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid S$
≫ $A \rightarrow B \mid S$
≫ $B \rightarrow b$

➔ **Here,** unit productions:

$S \rightarrow S$, $S_0 \rightarrow S$, $A \rightarrow B$, $A \rightarrow S$

❖ To Eliminate **$S \rightarrow S$**:

⇨ Since **S** → ASA | aB | a | AS | SA | S, We add **S** → ASA | aB | a | AS | SA | S:

∴**$S \rightarrow$ ASA | aB | a | AS | SA**

❖ To Eliminate **$S_0 \rightarrow S$**:

⇨ Since **S** → ASA | aB | a | AS | SA , We add **S_0** → ASA | aB | a | AS | SA:

∴**$S_0 \rightarrow$ ASA | aB | a | AS | SA**

❖ To Eliminate **$A \rightarrow B$**:

⇨ Since **B** → b, We add **A** → b:

∴ **$A \rightarrow b \mid S$**

❖ To Eliminate **$A \rightarrow S$**:

⇨ Since **S** → ASA | aB | a | AS | SA, We add **A** → ASA | aB | a | AS | SA:

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

∴New Production

≫ **$S_0 \rightarrow$ ASA | aB | a | AS | SA**
≫ **$S \rightarrow$ ASA | aB | a | AS | SA**
≫ **$A \rightarrow b \mid$ ASA | aB | a | AS | SA**
≫ **$B \rightarrow b$**

Step 3: Find not in CNF & Convert it into CNF

➔ **Is Given,** New Production

≫ $S_0 \rightarrow$ ASA | aB | a | AS | SA
≫ S → ASA | aB | a | AS | SA
≫ A → b | ASA | aB | a | AS | SA
≫ B → b

❖ Add C → SA:

∴ New Production

≫ $S_0 \rightarrow$ AC | aB | a | AS | SA
≫ S → AC | aB | a | AS | SA
≫ A → b | AC | aB | a | AS | SA
≫ B → b
≫ C → SA

❖ Add D → a:

∴ New Production

≫ $S_0 \rightarrow$ AC |DB | a | AS | SA
≫ S → AC | DB | a | AS | SA
≫ A → b | AC | DB | a | AS | SA
≫ B → b
≫ C → SA
≫ D → a

∴ The resultant grammar is-

≫ $S_0 \rightarrow$ AC |DB | a | AS | SA
≫ S → AC | DB | a | AS | SA
≫ A → b | AC | DB | a | AS | SA
≫ B → b
≫ C → SA
≫ D → a

Problem 3:

⇨ Convert the given grammar to CNF-

$S \rightarrow bS \mid aT \mid \in$
$T \rightarrow aT \mid bR \mid \in$
$R \rightarrow bS \mid \in$

Is Given, Original CFG	**Step1: Add new Start Variable P**	**Step2: Remove $R \rightarrow \in$**
$S \rightarrow bS \mid aT \mid \in$ $T \rightarrow aT \mid bR \mid \in$ $R \rightarrow bS \mid \in$	$P \rightarrow S$ $S \rightarrow bS \mid aT \mid \in$ $T \rightarrow aT \mid bR \mid \in$ $R \rightarrow bS \mid \in$	$P \rightarrow S$ $S \rightarrow bS \mid aT \mid \in$ $T \rightarrow aT \mid bR \mid \in \mid b$ $R \rightarrow bS$
Step3: Remove $T \rightarrow \in$	**Step4: Remove $S \rightarrow \in$**	**Step5: Remove $P \rightarrow S$ (Unit Remove)**
$P \rightarrow S$ $S \rightarrow bS \mid aT \mid \in \mid a$ $T \rightarrow aT \mid bR \mid b \mid a$ $R \rightarrow bS$	$P \rightarrow S \mid \in$ [ε is allowed only for the start symbol] $S \rightarrow bS \mid aT \mid a \mid b$ $T \rightarrow aT \mid bR \mid b \mid a$ $R \rightarrow bS \mid b$	$P \rightarrow bS \mid aT \mid a \mid b \mid \in$ $S \rightarrow bS \mid aT \mid a \mid b$ $T \rightarrow aT \mid bR \mid b \mid a$ $R \rightarrow bS \mid b$
Step6: Find rules not in CNF format	**Step7: Add $B \rightarrow b$, $A \rightarrow a$**	**Step8: Finally, in CNF format**
$P \rightarrow bS \mid aT \mid a \mid b \mid \in$ $S \rightarrow bS \mid aT \mid a \mid b$ $T \rightarrow aT \mid bR \mid b \mid a$ $R \rightarrow bS \mid b$	$P \rightarrow BS \mid AT \mid a \mid b \mid \in$ $S \rightarrow BS \mid AT \mid a \mid b$ $T \rightarrow AT \mid BR \mid b \mid a$ $R \rightarrow BS \mid b$ $A \rightarrow a$ $B \rightarrow b$	$P \rightarrow BS \mid AT \mid a \mid b \mid \in$ $S \rightarrow BS \mid AT \mid a \mid b$ $T \rightarrow AT \mid BR \mid b \mid a$ $R \rightarrow BS \mid b$ $A \rightarrow a$ $B \rightarrow b$

Problem 4:

⇨ Convert following Context free grammar to Chomsky normal form. Show all the steps.

D → xDx | yE | ∈
E → yEy | xF | ∈
F → xD | ∈

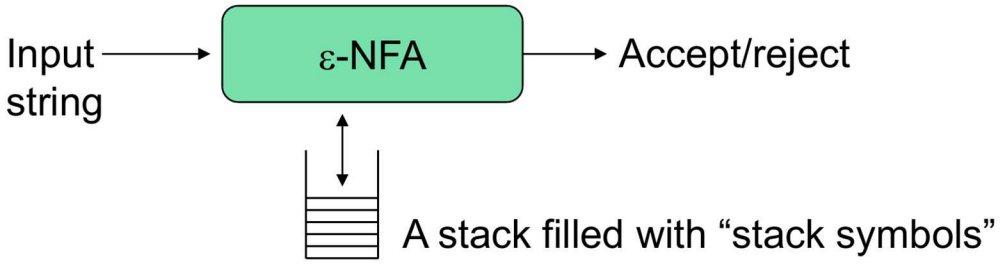
Is Given, Original CFG	**Step1: Add new Start Variable P**	**Step2: Remove F → ∈**
D → xDx | yE | ∈ E → yEy | xF | ∈ F → xD | ∈	P → D D → xDx | yE | ∈ E → yEy | xF | ∈ F → xD | ∈	P → D D → xDx | yE | ∈ E → yEy | xF | ∈ | x F → xD
Step3: Remove E → ∈	**Step4: Remove D → ∈**	**Step5: Find Unit Production**
P → D D → xDx | yE | ∈ | y E → yEy | xF | x | yy F → xD	P → D | ∈ D → xDx | yE | y | xx E → yEy | xF | x | yy F → xD | x	P → D | ∈ D → xDx | yE | y | xx E → yEy | xF | x | yy F → xD | x
Step6: Remove P → D (Unit Remove)	**Step7: Find rules not in CNF format**	**Step8: Add A→ x, B→ y**
P → xDx | yE | y | xx | ∈ D → xDx | yE | y | xx E → yEy | xF | x | yy F → xD | x	P → xDx | yE | y | xx | ∈ D → xDx | yE | y | xx E → yEy | xF | x | yy F → xD | x	P → ADA | BE | y | AA | ∈ D → ADA | BE | y | AA E → BEB | AF | x | BB F → AD | x A→ x B→ y
Step9: Add C→ DA, G→ EB	**Step10: Finally, in CNF format**	
P → AC | BE | y | AA | ∈ D → AC | BE | y | AA E → BF | AF | x | BB F → AD | x A→ x B→ y C→ DA G→ EB	P → AC | BE | y | AA | ∈ D → AC | BE | y | AA E → BF | AF | x | BB F → AD | x A→ x B→ y C→ DA G→ EB	

PUSHDOWN AUTOMATA (PDA)

Pushdown Automata (PDA)

⇨ A Pushdown Automata (PDA) is a way to implement a Context Free Grammar in a similar way we design Finite Automata for Regular Grammar

≫It is more powerful than FSM
≫FSM has a very limited memory, but PDA has more memory
≫**PDA** = Finite State Machine + A Stack



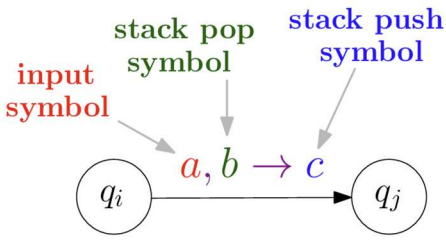
Formal Definition

⇨ A PDA P is a 7-tuple **P = (Q, Σ, Γ, δ, q_0, $, F)**

Where,

1. **Q** → finite set of states
2. **Σ** → finite input alphabet
3. **Γ** → finite stack alphabet
4. **δ** : $Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow P(Q \times \Gamma_\epsilon)$ → transition function
5. $q_0 \in Q$ → initial state
6. **$** $\in \Gamma$ → initial stack symbol
7. **F** $\subseteq Q$ → set of final states

Design Using PDA



∴ Transition function:

$(q_i, a, b) \rightarrow (q_j, c)$

➢ Label Format:

x, y → z

❖ What each part means:

- **x = input symbol from the alphabet ∑**
  - use **ε** if it doesn't read any input
  - use **Ø** if it's the end of the input
- **y = symbol currently on top of the stack to be popped**
  - use **ε** if nothing is popped
- **z = symbol to push onto the stack**
  - use **ε** if nothing is pushed

❖ Common operations made simple:

Action	Label Format	Meaning
Do nothing	`x, ε → ε`	No pop, no push
Pop only	`x, y → ε`	Pop `y` if it's on top, push nothing
Pop and push	`x, y → z`	Pop `y` and push `z`
Push only	`x, ε → z`	Push `z` without popping

Problem 1:

⇨ Design Push Down Automata for the following language.

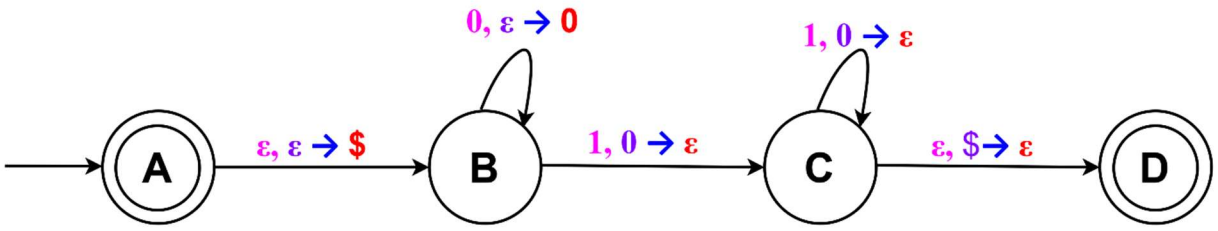
i. $L = \{0^n 1^n | n \geq 0\}$

» Let's break down the conditions:

- if, $n = 0$, → $0^0 1^0 = \{\varepsilon\}$
- if, $n = 1$, → $0^1 1^1 = \{01\}$
- if, $n = 2$, → $0^2 1^2 = \{0011\}$
- if, $n = 3$, → $0^3 1^3 = \{000111\}$ …

➔ **Is Given,**

- ∑ = {a,b},
- L(M) = {ε,ab, aabb, aaabbb, aaaabbbb,….}



∴Transition function:

(A, ε, ε) → (B, $)
(B, 0, ε) → (B, 0)
(B, 1, 0) → (C, ε)
(C, 1, 0) → (C, ε)
(C, ε, $) → (D, ε)

∴Transition Table:

Input	0			1			ε		
Stack	0	$	ε	0	$	ε	0	$	ε
A									(B,$)
B			(B,0)	(C,ε)					
C				(C,ε)				(D,ε)	
D									

Problem 2:

⇨ Design Push Down Automata for the following language.

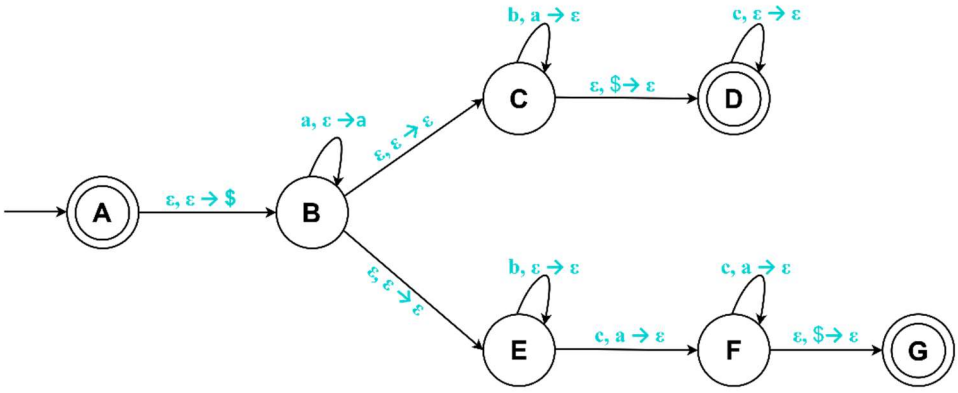
i. $L = \{a^i b^j c^k | i, j, k \geq 0 \ \& \ i = j \ or \ i = k\}$

» Let's break down the conditions:

- if, $i = j = k = 0$, → $a^0 b^0 c^0 = \{\varepsilon\}$
- if, $i = j = k = 1$, → $a^1 b^1 c^1 = \{abc\}$
- if, $i = j = 2$ & $k = 1$, → $a^2 b^2 c^1 = \{aabbc\}$
- if, $i = k = 2$ & $j = 1$, → $a^2 b^1 c^2 = \{aabcc\}$…

➔ **Is Given,**

- ∑ = {a,b,c},
- L(M) = {ε,abc, aabbc, aabcc, aaaabbbbc,….}



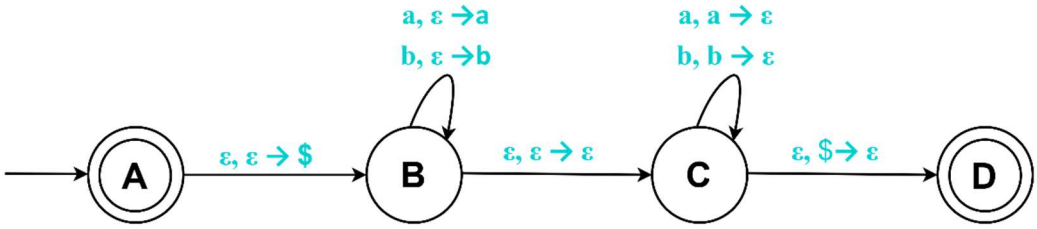
Problem 3:

⇨ Design Push Down Automata for the following language.

i. $L = \{WW^R \mid W \in \{a,b\}^*\}$

➔ **Is Given,**

- ∑ = {a,b},
- L(M) = {ε,abba, baab, ababbaba, abaaba,….}



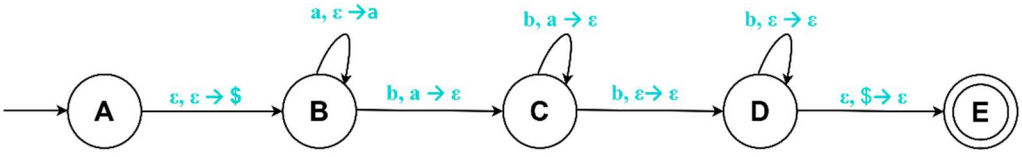
Problem 4:

⇨ Design Push Down Automata for the following language.

i. $L = \{a^m b^n \mid m < n \ \& \ m,n > 0\}$

➔ **Is Given,**

- ∑ = {a,b},
- L(M) = {abb, aabbb, aaabbbb, aabbbbbbbbbbbb,….}



Problem 5:

⇨ Design Push Down Automata for the following language.

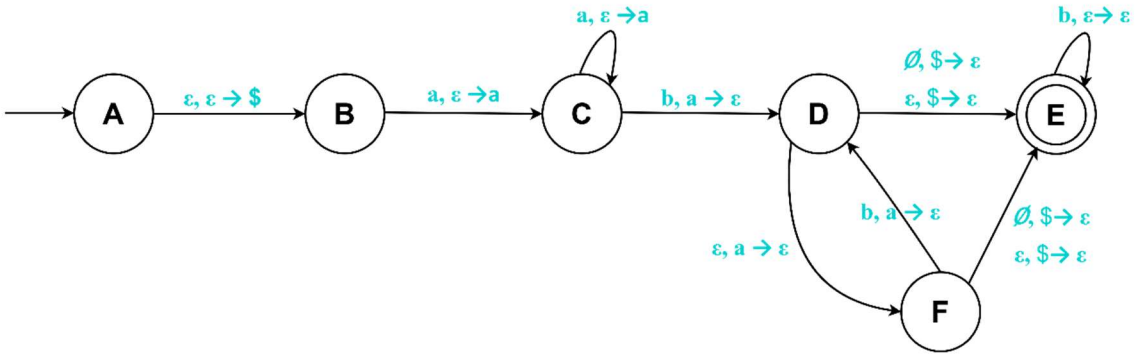
i. $L = \{a^m b^n \mid m \leq 2n \;\&\; n, m > 0\}$

» Let's break down the conditions:

- if, $n = 1, then\ m = 1,2 \rightarrow a^{1/2}b^1 = \{ab, aab\}$
- if, $n = 2, then\ m = 1,2,3,4 \rightarrow a^{1/2/3/4}b^2 = \{abb, aabb, \ldots, aaaabb\}\ldots$

➔ **Is Given,**

- ∑ = {a,b},
- L(M) = {ab, aab, abb, aabb, aaaabb, aaaaaabbb,….}



Problem 6:

⇨ Design Push Down Automata for the following language.

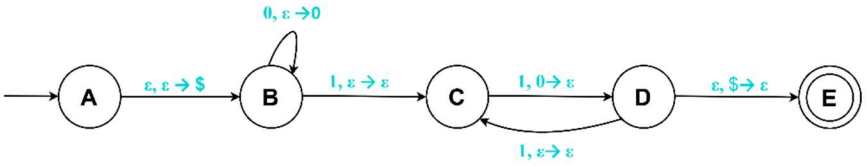
i. $L = \{0^n 1^{2n} \mid n > 0\}$

» Let's break down the conditions:

- if, $n = 1, \rightarrow 0^1 1^2 = \{011\}$
- if, $n = 2, \rightarrow 0^2 1^4 = \{001111\}$
- if, $n = 3, \rightarrow 0^3 1^6 = \{000111111\}\ldots$

➔ **Is Given,**

- ∑ = {0,1},
- L(M) = {011, 001111, 000111111, ….}



Problem 7:

⇨ Design Push Down Automata for the following language.

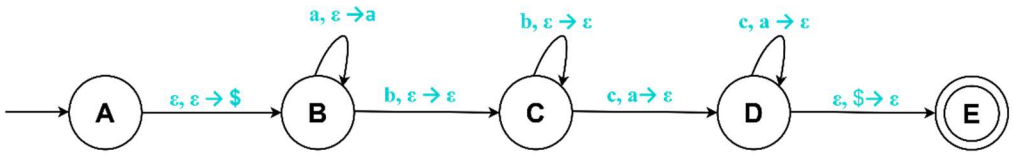
i. $L = \{a^n b^m c^n | \, m, n \geq 1\}$

» Let's break down the conditions:

- if, $n = 1, m = 1,$ → $a^1 b^1 c^1 = \{abc\}$
- if, $n = 2, m = 1,$ → $a^2 b^1 c^2 = \{aabcc\}$
- if, $n = 3, m = 3,$ → $a^3 b^3 c^3 = \{aaabbbccc\}$
- if, $n = 3, m = 2,$ → $a^3 b^2 c^3 = \{aaabbccc\}\ldots$

➔ **Is Given,**

- ∑ = {a,b,c},
- L(M) = { abc, $aabcc$, $aaabbbccc$, $aaabbccc$,….}



Problem 8:

⇨ Design Push Down Automata for the following language.

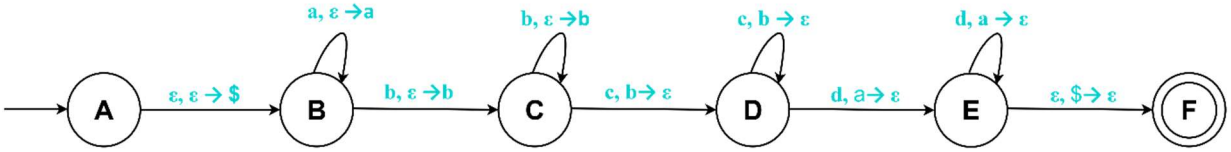
i. $L = \{a^n b^m c^m d^n | \, m, n \geq 1\}$

» Let's break down the conditions:

- if, $n = 1, m = 1,$ → $a^1 b^1 c^1 d^1 = \{abcd\}$
- if, $n = 2, m = 1,$ → $a^2 b^1 c^1 d^2 = \{aabcdd\}$
- if, $n = 3, m = 3,$ → $a^3 b^3 c^3 d^3 = \{aaabbbcccddd\}$
- if, $n = 3, m = 2,$ → $a^3 b^2 c^2 d^3 = \{aaabbccddd\}\ldots$

➔ **Is Given,**

- ∑ = {a,b,c},
- L(M) = { $abcd$, $aabcdd$, $aaabbbcccddd$, $aaabbccddd$,….}



Problem 9:

⇨ Design Push Down Automata for the following language.

i. $A = \{a^i b^j c^k \mid where\ i + j = k\ \&\ i, j, k \geq 1\}$

ii. $A = \{a^m b^n \mid where\ m = 2n\ \&\ m, n > 1\}$

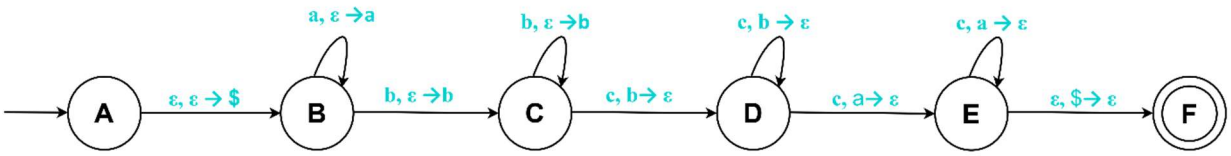
(i)

≫ Let's break down the conditions:

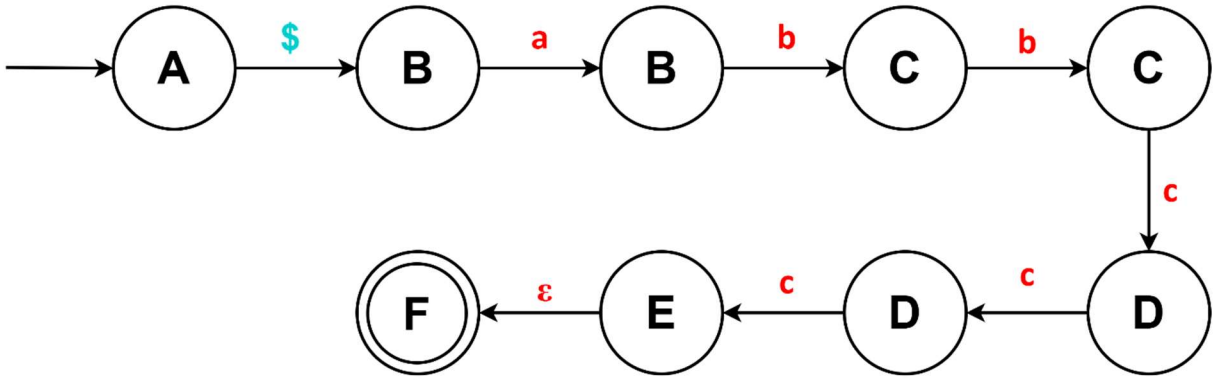
- if, $i = j = 1, k = 2,$ $then\ i + j = k$ → $a^1 b^1 c^2 = \{abcc\}$
- if, $i = 2, j = 1, k = 3,$ $then\ i + j = k$ → $a^2 b^1 c^3 = \{aabccc\}$
- if, $i = 1, j = 2, k = 3,$ $then\ i + j = k$ → $a^1 b^2 c^3 = \{abbccc\}\ldots$

➔ **Is Given,**

- ∑ = {a,b,c},
- L(M) = { $abcc, aabccc, abbccc, aabbcccc,\ldots.$}



≫ Let's Check string "*abbccc*" is Accepted or Rejected:



Accepted

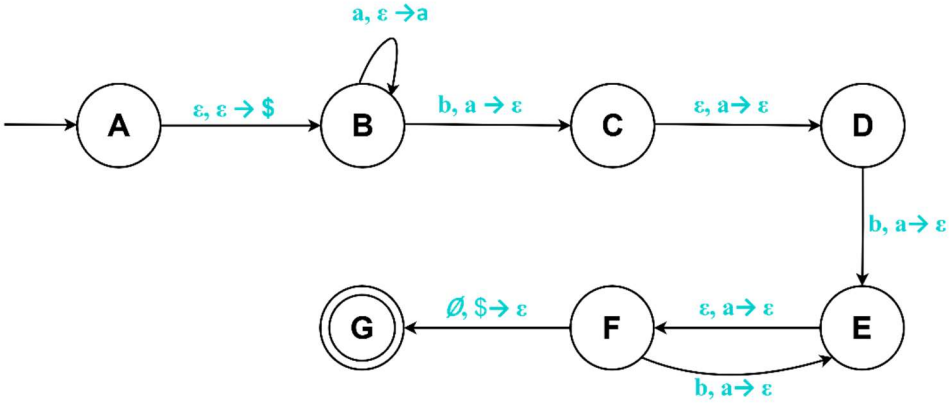
(ii)

≫ Let's break down the conditions:

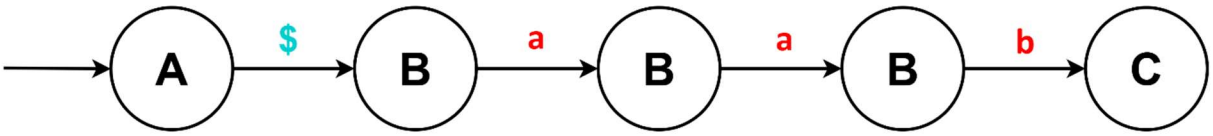
- if, $n = 2, m = 4 \quad then, m = 2n \quad \rightarrow \quad \boldsymbol{a^4b^2} = \{aaaabb\}$
- if, $n = 3, m = 6 \quad then, m = 2n \quad \rightarrow \quad \boldsymbol{a^6b^3} = \{aaaaaabbbb\}\ldots$

➔ **Is Given,**

- ∑ = {a,b,c},
- L(M) = { *aaaabb*, *aaaaaabbbb*,….}

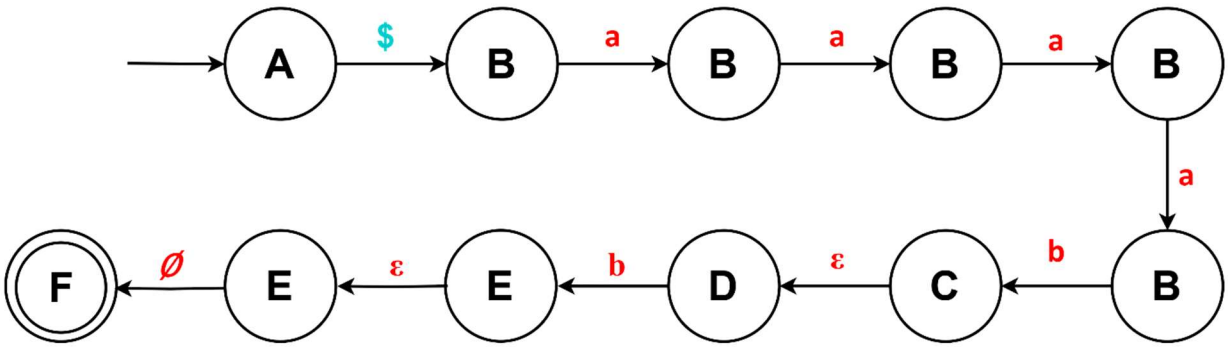


≫ Let's Check string "*aab*" is Accepted or Rejected:



Rejected

≫ Let's Check string "*aaaabb*" is Accepted or Rejected:



Accepted

TURING MACHINE

Turing Machine

A Turing machine (TM) is a computer model that reads and writes to an infinite tape to perform computations. The tape is divided into cells into which input is given. It contains a reading head that reads the input tape.

A state register is used to store the state of the Turing machine. The cell in the tape is replaced with another symbol, its internal state is changed, and it travels from one cell to the right or left after reading an input symbol.

The input string is accepted if the TM reaches the end state; else, it is denied.

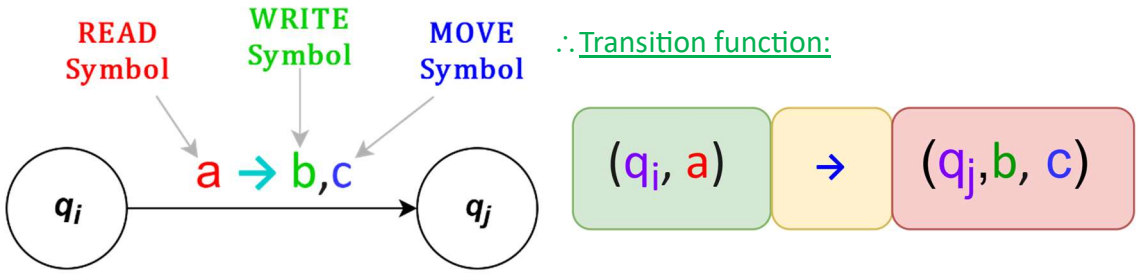
Formal definition of Turing machine

A TM is defined as a **7-tuple ($Q, \Gamma, \sum, \delta, q0, q_{accept}, q_{reject}$)**

where –

1. **Q** = Finite set of states
2. **Γ** = Tape alphabet
3. **∑** = Input alphabet
4. **δ** = Transition function; $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.
5. q_0 =Initial state
6. q_{accept} = Accept state
7. q_{reject} = Reject state

Design Using Turing Machine



Label Conventions:

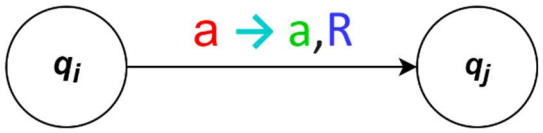
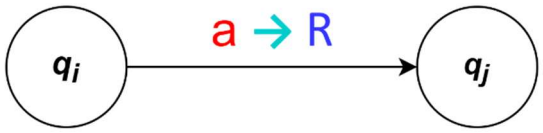
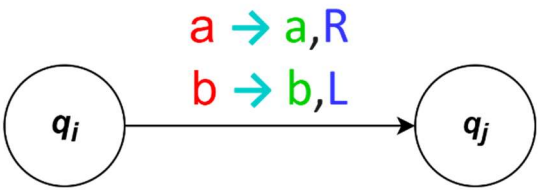
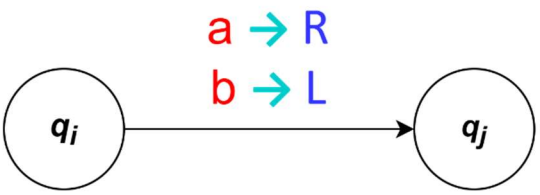
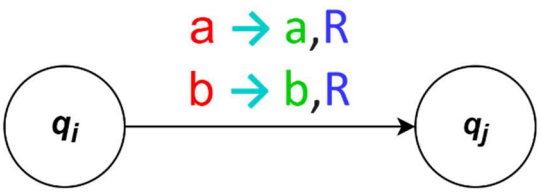
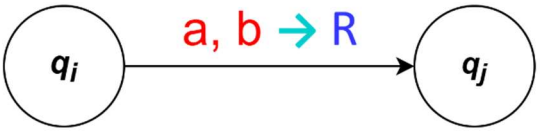
If Alphabet:

- **Σ = {α, β, γ}** (these are the symbols on the tape)
- **D = {L, R}** (directions: Left or Right)

Label Shortcuts:

Original Notation	Simplified Notation
$\alpha \rightarrow \alpha, D$	$\alpha \rightarrow D$
$\alpha \rightarrow D,$ $\gamma \rightarrow D$	$\alpha, \gamma \rightarrow D$
$\alpha \rightarrow D,$ $\beta \rightarrow D$	$\alpha, \beta \rightarrow D$

Examples for clarity:

Original Notation	Simplified Notation
	
	
	

Problem 1:

⇨ Design **Turing Machine** for the following language.

i. **L = {01*0}**

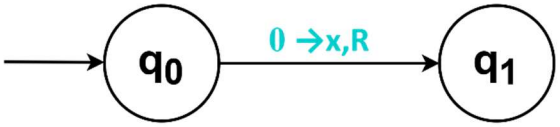
➔ **Is Given,**

- ∑ = {0,1},
- L(M) = {00,010, 0110, 01110, 011110, ...},
- Tape alphabet Γ = {0,1, ␣}

Transition Logic:	
Step1:	**Step2:**
• **q0**: If the first symbol is `0`, go to `q1`. Else → reject.	• **q1**: Keep moving right through all `1`s. If you find `0` (at the end) → go to `q2`.
Step3:	
• **q2**: If current symbol is blank → accept. Else → reject.	

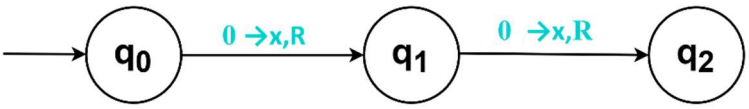
For Accepting minimum String: 00

Step1:



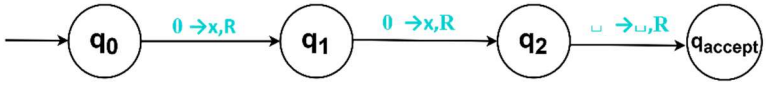
␣	0	0	␣
␣	X		

Step2:



␣	0	0	␣
␣	X	X	

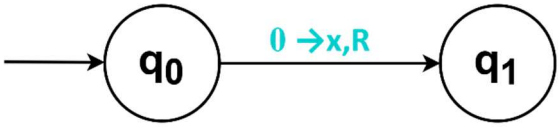
Step3:



␣	0	0	␣
␣	X	X	␣

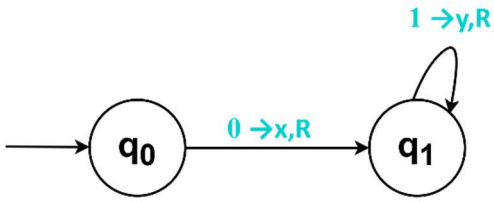
For Accepting minimum String: 010

Step1:



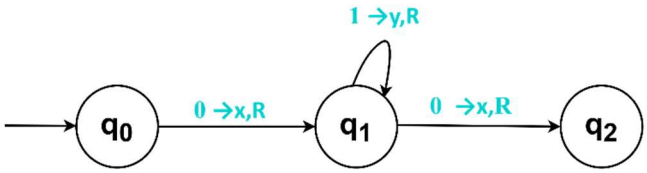
␣	0	1	0	␣
␣	X			

Step2:



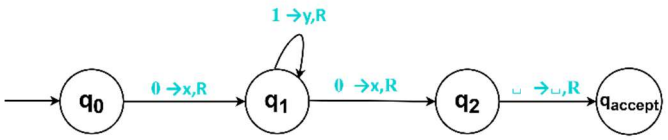
␣	0	1	0	␣
␣	X	Y		

Step3:



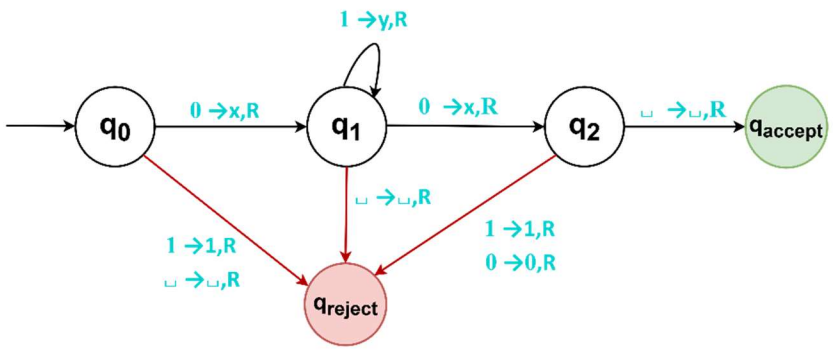
␣	0	1	0	␣
␣	X	Y	X	

Step4:



␣	0	1	0	␣
␣	X	Y	X	␣

Step5: We must show the rejected strings



Transition Function:

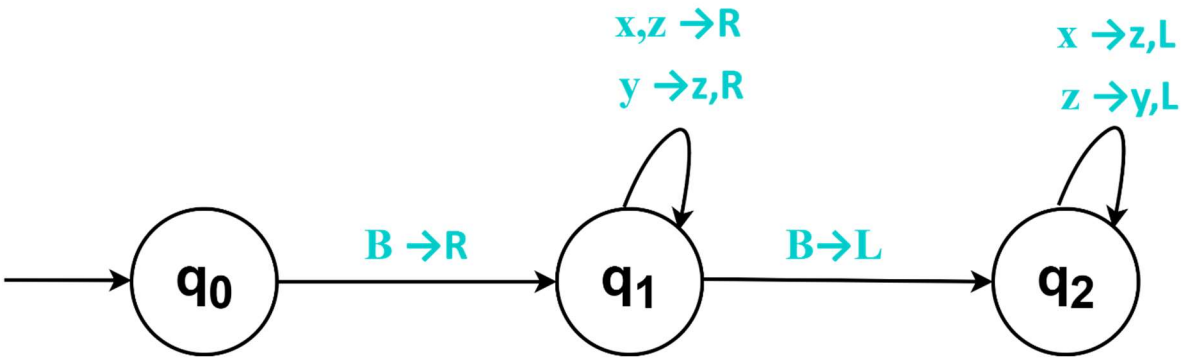
δ	0	1	␣
q_0	(q1, X, R)	(q_{reject}, 1, R)	(q_{reject}, ␣, R)
q_1	(q2, X, R)	(q1, Y, R)	(q_{reject}, ␣, R)
q_2	(q_{reject}, 0, R)	(q_{reject}, 1, R)	(q_{accept}, ␣, R)

Problem 2:

➔ Design a TM that Q = {q0,q1,q2}, Σ = {x,y,z}, q_0 = q0,

δ	x	y	z	B(blank)
q_0				(q1, B, R)
q_1	(q1, x, R)	(q1, z, R)	(q1, z, R)	(q2, B, L)
q_2	(q2, z, L)		(q2, y, L)	

➔ Solution:



Problem 3:

⇨ Design **Turing Machine** for the following language.

i. $L = \{0^n 1^n; n > 0\}$

➔ **Is Given,**

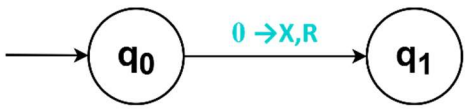
- ∑ = {0,1},
- L(M) = {01,0011, 000111, 00001111, ...},
- Tape alphabet Γ = {0,1, ␣}

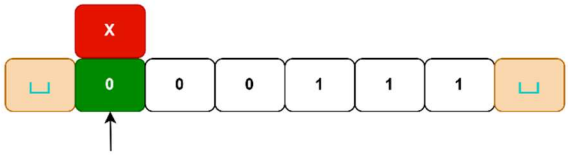
Design Idea: We'll match each **0** with a **1**:

- **Step1:** Replace the **leftmost 0** with **X**.
- **Step2:** Move right to find the **rightmost unmatched 1**, replace with **Y** & Move left.
- **Step3:** Move left **Until we find the X.**
- **Step4:** Repeat **Step 1,2,3.**
- **Step5:** If all 0s and 1s are replaced with X and Y in equal number → ACCEPT.
- **Step6:** If mismatch happens (more 0s or more 1s) → REJECT.

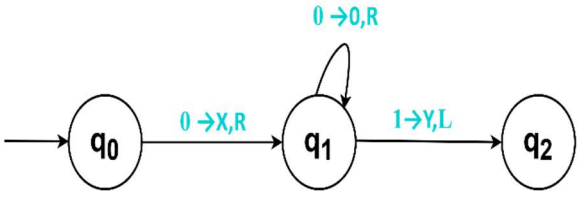
Let's Take a Valid Input Tape:

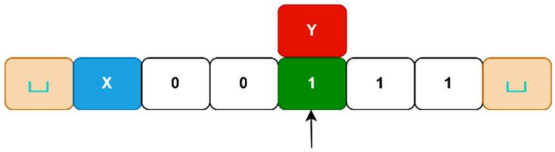
Step1: Replace the **leftmost 0** with **X**.



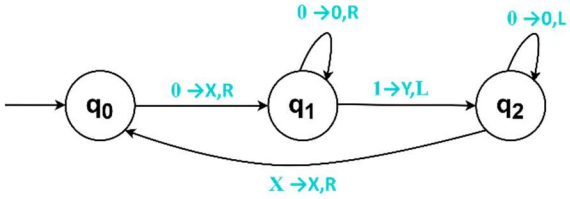


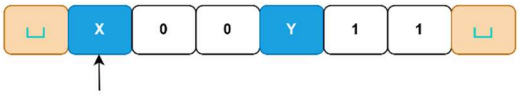
Step2: Move right to find the **rightmost unmatched 1**, replace with **Y** & Move left.





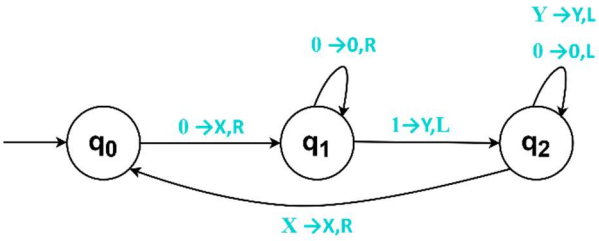
Step3: Move left **Until we find the X.**

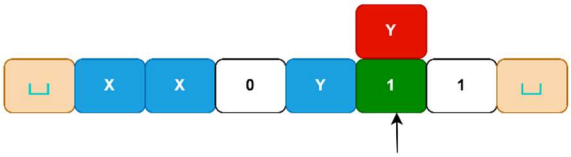




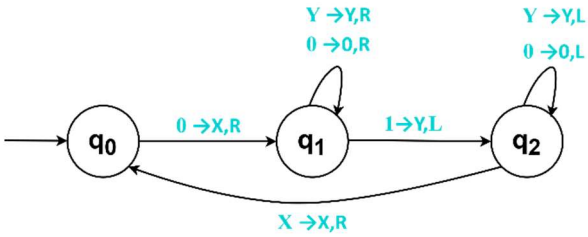
Step4: Repeat **Step 1,2,3.**

Step4.2:

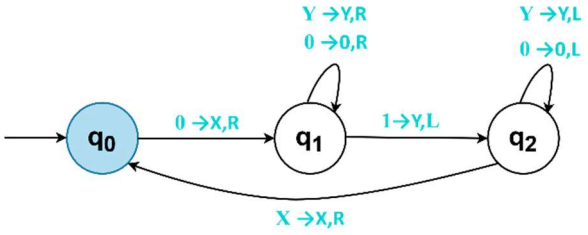


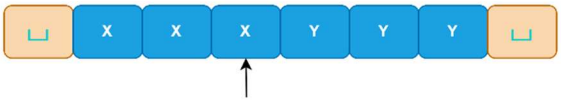


Step4.4:



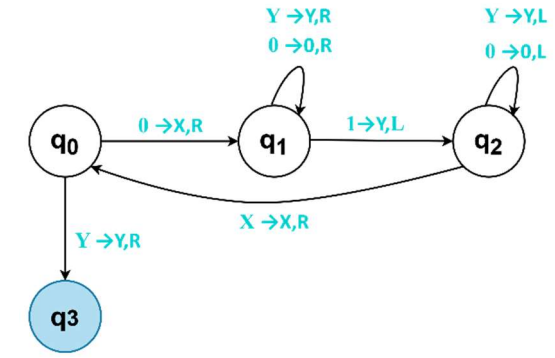
After Completions of Step4:

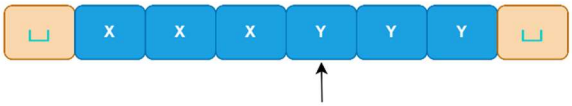




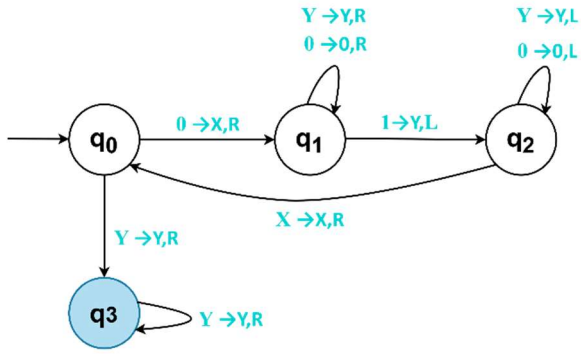
Step5: If all 0s and 1s are replaced with X and Y in equal number → ACCEPT.

Step5.1:

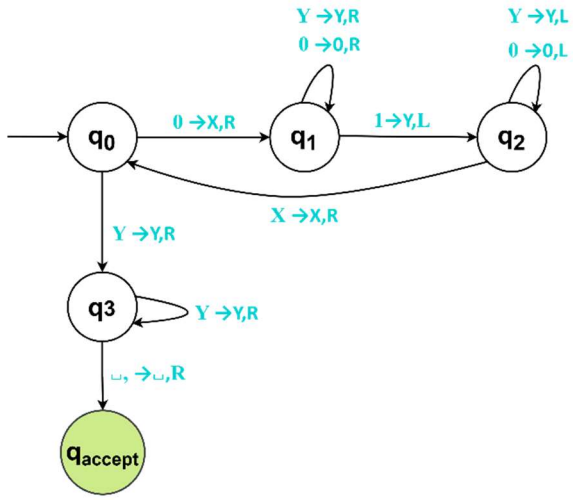




Step5.2 & 5.3:



Step5.4:



Problem 4:

⇨ Design **Turing Machine** for the following language.

i. **L = {w#w | w ∈ {0,1}* }**

➔ **Is Given,**

- ∑ = {0,1},
- L(M) = {01#01,1100#1100, 0101#0101, ...},
- Tape alphabet Γ = {0,1, ␣}

🧠 **Design Idea**: We will use **marking** (with X or Y) and **comparison** from both sides of the #.

🎯 Key Concepts:

- **Mark and store** one symbol from the left of `#`.
- **Find the corresponding symbol** on the right of `#` and compare it.
- If they match, **mark both**, go back and repeat for the next.
- If any mismatch, or leftover symbols, → **REJECT**.
- If all matched and no extra symbols → **ACCEPT**.

Step1: Initialization:

- Start at the left of the input.
- Read the first unmarked symbol (either `0` or `1`) **before the #**.
- **Store** it in the state.
- Replace it with a marker (e.g., `X` or `Y`) to mark it as processed.

Step2: Move to the Right Part:

- Move right until you reach the `#`.
- Continue moving to the **right side** of the `#`.
- Skip already marked symbols (X/Y).
- When you find the **first unmarked** symbol, **compare** it with the stored one:
  - If match → **mark it** (e.g., with `x` or `y`)
  - If not → **REJECT**

Step3: Return Back to the Left:

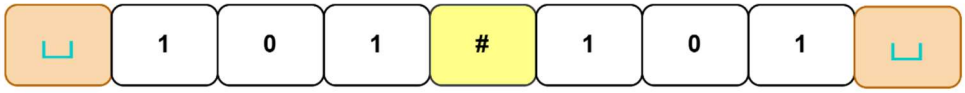
- After matching, go back left to the beginning.
- Repeat step 1 with the next unmarked symbol before the `#`.

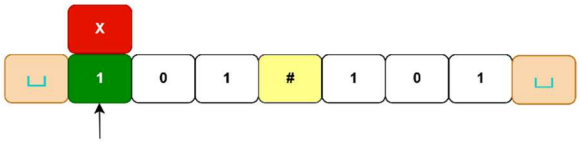
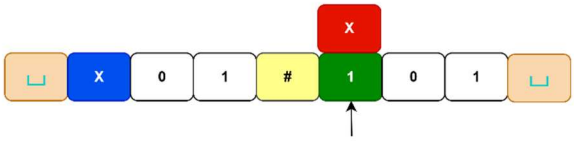
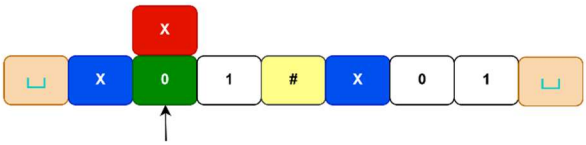
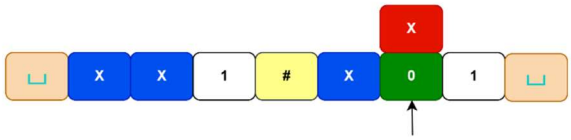
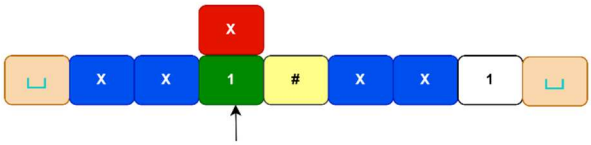
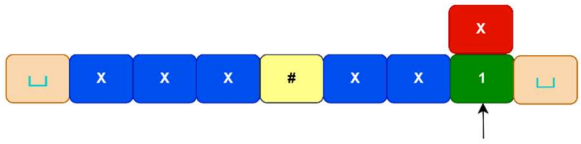
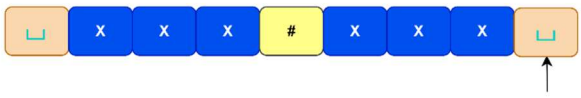
Step4: Final Check:

- When no unmarked symbols remain before the `#`:
  - Move to the right side of `#`
  - If all symbols after `#` are marked and nothing extra remains → **ACCEPT**
  - If any unmarked symbols are still there → **REJECT**

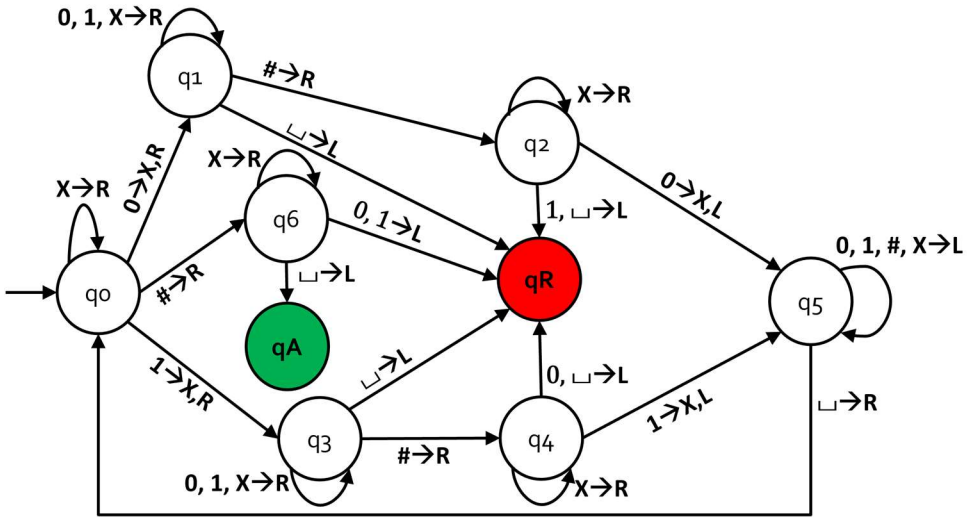
Example Trace: `101#101`

Initial Tape:



Mark '1':	
	
Mark '0':	
	
Mark '1':	
	
Final Check:	
	`No unmarked left → check right → all marked →` ✅ **`ACCEPT`**

SO, the **Turing Machine:**



Problem 5:

⇨ Design **Turing Machine** for the following language.

ii. $L = \{ 0^j \mid j=2^n ; n > 0\}$

≫ Let's break down the conditions:

- if, $n = 1$, $j = 2^1 = 2$, → $\mathbf{0^2}$ $= \{\mathbf{00}\}$
- if, $n = 2$, $j = 2^2 = 4$, → $\mathbf{0^4}$ $= \{\mathbf{0000}\}$
- if, $n = 3$, $j = 2^3 = 8$, → $\mathbf{0^8}$ $= \{\mathbf{00000000}\}$ …

➔ **Is Given,**

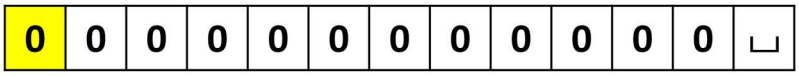
- $\sum = \{0,1\}$,
- L(M) = {00,0000,00000000},
- Tape alphabet Γ = {0, ␣}

🧠 **Design Idea**:

1. **Replace** the **leftmost** 0 with ␣.
2. From left to right, **cross (×) one 0, skip the next**, and repeat. (Ignore ␣ and ×.)
   - If **no 0s remain**, **ACCEPT**
   - If **no 0 remains after skip the next 0**, **REJECT**
   - Else, **go back to the leftmost** ␣ and repeat from step 2.

Example Trace:

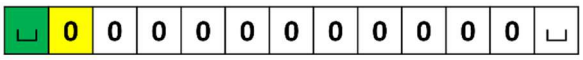
Initial Tape:



Previous Head Position

Current Head Position

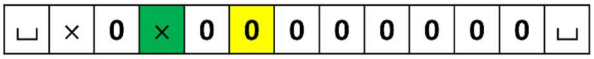
Step1: Replace the leftmost 0 with ␣



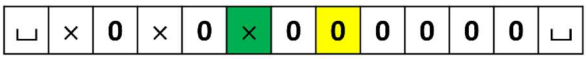
Step2: Cross (×) one 0, skip the next & repeat

Repeat Step 2:

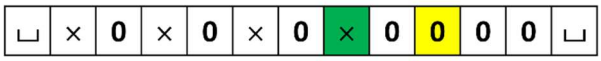
Step 2.1:



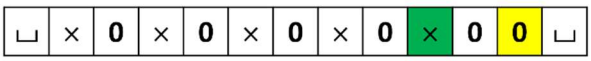
Step 2.2:



Step 2.3:

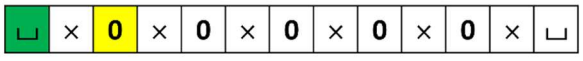


Step 2.4:

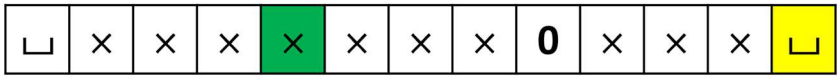


Step 2.5:

Step 2.6:



Final Check:

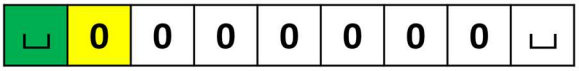


REJECTED

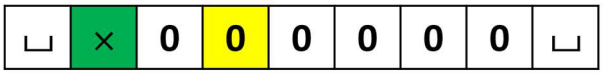
Example Trace:

Initial Tape:

Step1: Replace the leftmost 0 with ␣

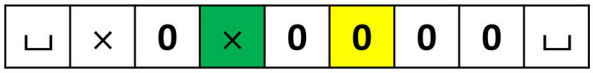


Step2: Cross (×) one 0, skip the next & repeat

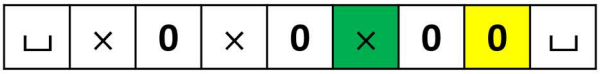


Repeat Step 2:

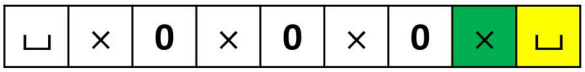
Step 2.1:



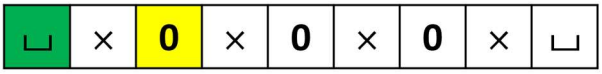
Step 2.2:



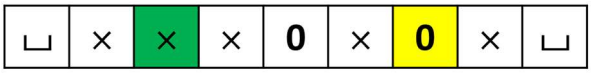
Step 2.3:



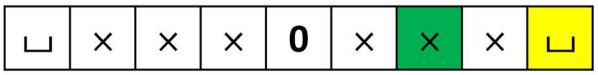
Step 2.4:



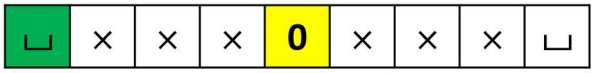
Step 2.5:



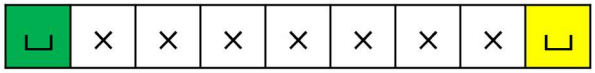
Step 2.6:



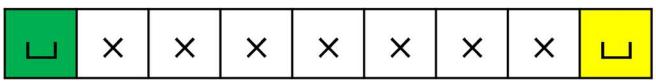
Step 2.7:



Step 2.8:

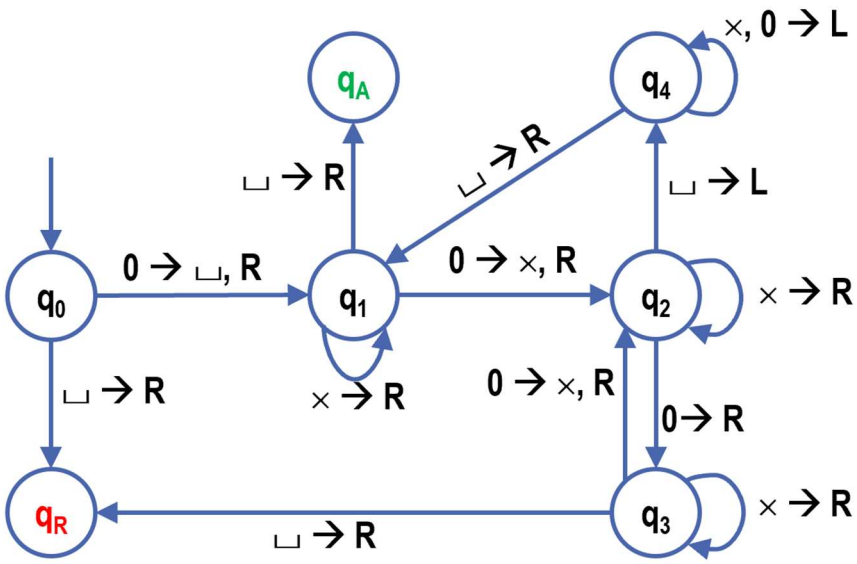


Final Check:



ACCEPTED

SO, the **Turing Machine:**



Problem 6:

⇨ Design **Turing Machine** for the following language.

iii. $L = \{ a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1 \}$

≫ Let's break down the conditions:

- if, $k = 1$, $i \times j = 1 \times 1 = 1$, → $\mathbf{a^1b^1c^1} = \{abc\}$
- if, $k = 2$, $i \times j = 1 \times 2 = 2$, → $\mathbf{a^1b^2c^2} = \{abbcc\}$
- if, $k = 3$, $i \times j = 3 \times 1 = 3$, → $\mathbf{a^3b^1c^3} = \{aaabccc\}$…

➔ **Is Given,**

- $\sum = \{0,1\}$,
- L(M) = {abc,abbcc,aabcc,aaabccc,aabbbccccc},

```
aabbbccccc
Xabbbccccc
XaYYYZZZccc
XabbbZZZccc
XXYYYZZZZZZ
```

```
For every a
    replace a by X
    for every b
        replace b by Y
        replace next c by Z
    end for
    for every Y
        replace Y by b
    end for
end for
```

Problem 7:

⇨ Design **Turing Machine** for the following language.

iv. **L = {w | w is binary string and number of 0s and 1s are equal}**

➔ **Is Given,**

- ∑ = {0,1},
- L(M) = {01, 0101, 110100},

```
0 1 1 0 1 0 0 1
X 1 1 0 1 0 0 1
X Y 1 0 1 0 0 1
X Y 1 X 1 0 0 1
X Y Y X 1 0 0 1
X Y Y X 1 X 0 1
X Y Y X Y X 0 1
X Y Y X Y X X 1
X Y Y X Y X X Y
```

```
For every 0
    replace next 0 from left by X
    replace next 1 from left by Y
End for
```

Problem 8:

⇨ Design **Turing Machine** for the following language. & Show **Computation Steps for Input** 01001100010, 11011011

v. **L = { ww^R | w is a binary string}**

➔ **Is Given,**

- ∑ = {0,1},
- L(M) = {0110, 1001, 11011011},

```
0 1 1 0 0 1 1 0
X 1 1 0 0 1 1 X
X X 1 0 0 1 X X
X X X 0 0 X X X
X X X X X X X X
```

```
For each leftmost alphabet (means not X)
    if alphabet is 0
        replace 0 by X
        if the rightmost alphabet is 0 replace by X
    else {alphabet is 1}
        replace 1 by X
        if the rightmost alphabet is 1 replace by X
End for
```

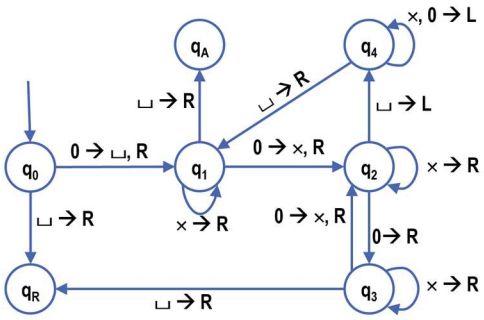
⇨ **Computation Steps for Input** 01001100010

Step	Current Tape	Action Taken
0	**01001100010**	Initial input
1	**X100110001X**	Match Leftmost 0 with rightmost 0 → mark both as X
2	**XX0011000XX**	Match Leftmost 1 with rightmost 1 → mark both as X
3	**XXX01100XXX**	Match Leftmost 0 with rightmost 0 → mark both as X
4	**XXXX110XXXX**	Match Leftmost 0 with rightmost 0 → mark both as X
5	**XXXX110XXXX**	Match Leftmost 1 with rightmost 0 → mismatch!
	Rejected	

⇨ **Computation Steps for Input** 11011011

Step	Current Tape	Action Taken
0	**11011011**	Initial input
1	**X101101X**	Match Leftmost 1 with rightmost 1 → mark both as X
2	**XX0110XX**	Match Leftmost 1 with rightmost 1 → mark both as X
3	**XXX11XXX**	Match Leftmost 0 with rightmost 0 → mark both as X
4	**XXXXXXXX**	Match Leftmost 1 with rightmost 1 → mark both as X
	Accepted	

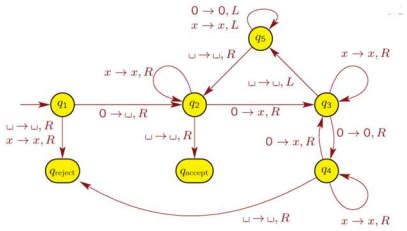
Problem 9: Trace the execution of this Turing machine with the string **00000000** as input.



Solutions:

q_0 0 0 0 0 0 0 0 0 ␣ ␣ q_1 0 0 0 0 0 0 0 ␣ ␣ × q_2 0 0 0 0 0 0 ␣ ␣ × 0 q_3 0 0 0 0 0 ␣ ␣ × 0 × q_2 0 0 0 0 ␣ ␣ × 0 × 0 q_3 0 0 0 ␣ ␣ × 0 × 0 × q_2 0 0 ␣ ␣ × 0 × 0 × 0 q_3 0 ␣ ␣ × 0 × 0 × 0 × q_2 ␣ ␣ × 0 × 0 × 0 q_4 × ␣		␣ × 0 × 0 × 0 q_4 × ␣ ␣ × 0 × 0 × q_4 0 × ␣ ␣ × 0 × 0 q_4 × 0 × ␣ ␣ × 0 × q_4 0 × 0 × ␣ ␣ × 0 q_4 × 0 × 0 × ␣ ␣ × q_4 0 × 0 × 0 × ␣ ␣ q_4 × 0 × 0 × 0 × ␣ q_4 ␣ × 0 × 0 × 0 × ␣ ␣ q_1 × 0 × 0 × 0 × ␣ ␣ × q_1 0 × 0 × 0 × ␣
␣ × q_1 0 × 0 × 0 × ␣ ␣ × × × q_2 0 × 0 × ␣ ␣ × × × 0 × q_3 0 × ␣ ␣ × × × 0 × × × q_2 ␣ q_4 ␣ × × × 0 × × × ␣ ␣ × × q_1 0 × × × × ␣ ␣ × × × × × × × q_2 ␣ q_4 ␣ × × × × × × × ␣ ␣ × × × × × × × q_1 ␣ ␣ × × × × × × × ␣ q_A		**To better understand, follow the slide simulations**

Problem 10: Trace the computation for the input 0000␣ of the above state diagram



➪ **Step-by-step execution of the Turing machine** on input 0000␣ in **tabular format**:

Step	State	Tape	Head Pos	Action
1	q1	0000␣	0	Read 0 → Write ␣, move R go to q2
2	q2	␣000␣	1	**0 → x,** move **R→ q3**
3	q3	␣x00␣	2	**0 → 0,** move **R→ q4**
4	q4	␣x00␣	3	**0 → x,** move **R→ q3**
5	q3	␣x0x␣	4	**␣ → ␣,** move **L→ q5**
6	q5	␣x0x␣	3	**x → x,** move **L→ q5**
7	q5	␣x0x␣	2	**0 → 0,** move **L→ q5**
8	q5	␣x0x␣	1	**x → x,** move **L→ q5**
9	q5	␣x0x␣	0	**␣ → ␣,** move **R→ q2**
10	q2	␣x0x␣	1	**x → x,** move **R→ q2**
11	q2	␣x0x␣	2	**0 → x,** move **R→ q3**
12	q3	␣xxx␣	3	**x → x,** move **R→ q3**
13	q3	␣xxx␣	4	**␣ → ␣,** move **L→ q5**
14	q5	␣xxx␣	3	**x → x,** move **L→ q5**
15	q5	␣xxx␣	2	**x → x,** move **L→ q5**
16	q5	␣xxx␣	1	**x → x,** move **L→ q5**
17	q5	␣xxx␣	0	**␣ → ␣,** move **R→ q2**
18	q2	␣xxx␣	1	**x → x,** move **R→ q2**
19	q2	␣xxx␣	2	**x → x,** move **R→ q2**
20	q2	␣xxx␣	3	**x → x,** move **R→ q2**
21	q2	␣xxx␣	4	**␣ → ␣,** move **R→ q_{accept}**

Final State: q_{accept}

- The machine **accepts** the input.
- Tape content: ␣ **x x x** ␣

Accepted

Problem 11: Consider a Turing machine with the following transitions.

State	Input	**δ** (State, Symbol, Move)
Qo	a	Q1,#,R
Q0	#	Qaccept, #,R
Q1	a	Q1, a, R
Q1	b	Q2,x,R
Q1	x	Q1, x, R
Q2	a	Q3,x,R
Q2	b	Q2, b, R
Q2	x	Q2, x, R
Q3	a	Q4,a,L
Q3	#	Q6,#,L
Q4	a	Q4,a,L
Q4	b	Q4,b,L
Q4	x	Q4,x,L
Q4	#	Q5,#,R
Q5	a	Q1,x,R
Q5	x	Q5,x,R
Q6	x	Q6,x,L
Q6	#	Qaccept, #,R

Here 'Q_0' is the start state, 'Q_{Accept}' is the accept state. Trace the execution of this Turing machine with the string **aabbaa#** as input. Note that '#' represents the blank symbol.

⇨ **Step-by-step execution of the Turing machine** on input `aabbaa#` in **tabular format**:

Step	State	Tape	Head Pos	Action
1	Q0	**a** a b b a a #	0	Read `a` → Write `#`, move R go to Q1
2	Q1	**#** **a** b b a a #	1	**`a`** → **a**, move **R**→ **Q1**
3	Q1	# **a** **b** b a a #	2	**`b`** → **x**, move **R**→ **Q2**
4	Q2	# a **x** **b** a a #	3	**`b`** → **b**, move **R**→ **Q2**
5	Q2	# a x **b** **a** a #	4	**`a`** → **x**, move **R**→ **Q3**
6	Q3	# a x b **x** **a** #	5	**`a`** → **a**, move **L**→ **Q4**
7	Q4	# a x b **x** **a** #	4	**x** → **x**, move **L**→ **Q4**
8	Q4	# a x **b** **x** a #	3	**b** → **b**, move **L**→ **Q4**
9	Q4	# a **x** **b** x a #	2	**x** → **x**, move **L**→ **Q4**
10	Q4	# **a** **x** b x a #	1	**`a`** → **a**, move **L**→ **Q4**
11	Q4	**#** **a** x b x a #	0	**#** → **#**, move **R**→ **Q5**
12	Q5	**#** **a** x b x a #	1	**`a`** → **x**, move **R**→ **Q1**
13	Q1	# **x** **x** b x a #	2	**x** → **x**, move **R**→ **Q1**
14	Q1	# x **x** **b** x a #	3	**`b`** → **x**, move **R**→ **Q2**
15	Q2	# x x **x** **x** a #	4	**x** → **x**, move **R**→ **Q2**
16	Q2	# x x x **x** **a** #	5	**`a`** → **x**, move **R**→ **Q3**
17	Q3	# x x x x **x** **#**	6	**#** → **#**, move **L**→ **Q6**
18	Q6	# x x x x **x** #	5	**x** → **x**, move **L**→ **Q6**
19	Q6	# x x x **x** **x** #	4	**x** → **x**, move **L**→ **Q6**
20	Q6	# x x **x** **x** x #	3	**x** → **x**, move **L**→ **Q6**
21	Q6	# x **x** **x** x x #	2	**x** → **x**, move **L**→ **Q6**
22	Q6	# **x** **x** x x x #	1	**x** → **x**, move **L**→ **Q6**
23	Q6	**#** **x** x x x x #	0	**#** → **#**, move **R**→ **Q_{accept}**

Final State: Q_{accept}

- The machine **accepts** the input.
- Tape content: **`# x x x x x #`**

Accepted

Problem 12: In the Transition Function, Q_0 is the start state, Q_A is the accept state and Q_R is the reject state. Trace the execution of this Turing machine with the string **010001B** as input. Note that 'B' represents the blank symbol.

Transition Table:

State	Symbol	δ(State, Symbol)
Q_0	0	(Q_1, B, R)
Q_0	1	(Q_4, B, R)
Q_0	B	(Q_A, B, R)
Q_1	0	$(Q_1, 0, R)$
Q_1	1	$(Q_1, 1, R)$
Q_1	B	(Q_2, B, L)
Q_2	0	(Q_3, B, L)
Q_2	1	(Q_R, B, R)
Q_2	B	(Q_A, B, R)
Q_3	0	$(Q_3, 0, L)$
Q_3	1	$(Q_3, 1, L)$
Q_3	B	(Q_0, B, R)
Q_4	0	$(Q_4, 0, R)$
Q_4	1	$(Q_4, 1, R)$
Q_4	B	(Q_5, B, L)
Q_5	0	(Q_R, B, R)
Q_5	1	(Q_3, B, L)
Q_5	B	(Q_A, B, R)

⇨ **Step-by-step execution of the Turing machine** on input `010001B` in **tabular format**:

Step	State	Tape	Head Pos	Action
1	Q_0	**0** 1 0 0 0 1 B	0	0 → B, move R → Q_1
2	Q_1	**B** **1** 0 0 0 1 B	1	1 → 1, move R → Q_1
3	Q_1	B **1** **0** 0 0 1 B	2	0 → 0, move R → Q_1
4	Q_1	B 1 **0** **0** 0 1 B	3	0 → 0, move R → Q_1
5	Q_1	B 1 0 **0** **0** 1 B	4	0 → 0, move R → Q_1
6	Q_1	B 1 0 0 **0** **1** B	5	1 → 1, move R → Q_1
7	Q_1	B 1 0 0 0 **1** **B**	6	B → B, move L → Q_2
8	Q_2	B 1 0 0 0 **1** **B**	5	1 → B, move R → QR (Reject)

- **Problem 13:** Read the algorithm carefully. Then design the Implementation Level Strategy for the algorithm. Describe how you are going to use the tape for designing the Turing Machine.

  Algorithm:

  For every ‘a’

  replace next ‘a’ from left by X

  replace next two ‘b’ from left by Y

  End for

  if

  for each ‘a’ there is one or more than two ‘b’, then REJECT

  Else if

  All ‘a’ and ‘b’are marked, then ACCEPT.