

TOC

Q.7 Design a DFA over the alphabet $\Sigma = \{0, 1\}$ and accepts the language having an strings ending with at least two 0's.

↳ Let M be required DFA

where, $M = \{Q, \Sigma, \delta, q_0, F\}$



For 00, 010,
100, 1010,
001111

where,

$$Q = \{q_0, q_1, q_2\}$$

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

$$q_0 = q_0$$

$$F = q_2$$

Transition Function $\delta: \Rightarrow Q \times \Sigma \Rightarrow Q$

$$\delta(q_0, 1) = q_0$$

$$\delta(q_0, 0) = q_1$$

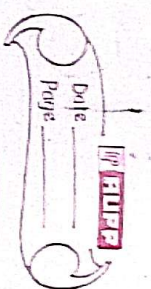
$$\delta(q_1, 1) = q_1$$

$$\delta(q_1, 0) = q_2$$

$$\delta(q_2, 1) = q_1$$

$$\delta(q_2, 0) = q_2$$

Q15 is only correct answer.

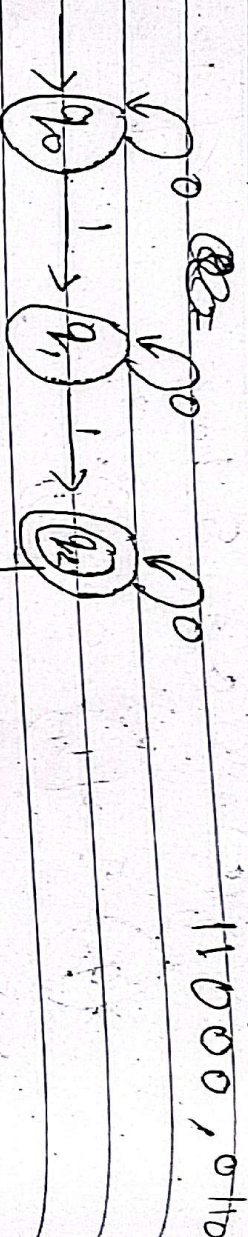


Q.7 with exactly two 1's

Let make regular DFA

$M = \{Q, \Sigma, \delta, q_0, F\}$

where,



$Q = \{q_0, q_1, q_2, q_3\}$

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

$q_0 = q_0$

$F = q_3$

Transition Function:

$\delta(q_0, 0) = q_0$

$\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_0$

$\delta(q_1, 1) = q_2$

$\delta(q_2, 0) = q_1$

$\delta(q_2, 1) = q_3$

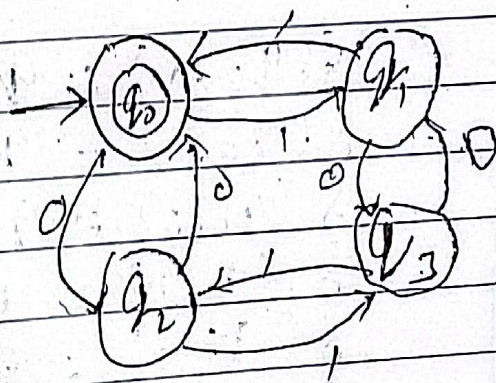
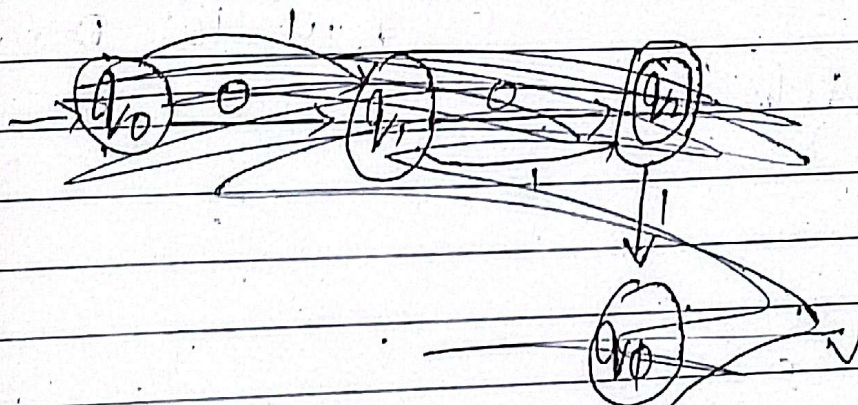
$\delta(q_3, 0) = q_3$

$\delta(q_3, 1) = q_2$

(Q) Design a DFA over the alphabet $\Sigma = \{0, 1\}$ and accepts the language having strings with number of 1's is even and number of 0's is even.

$L = \{00, 11, 0011, 1010, 110011, 001100, \dots\}$

So let the required DFA
 $M = \{Q, \Sigma, \delta, q_0, F\}$
 where



$\delta(q_0, 1) = q_1$
 $\delta(q_0, 0) = q_3$
 $\delta(q_1, 1) = q_2$
 $\delta(q_1, 0) = q_0$
 $\delta(q_2, 1) = q_3$
 $\delta(q_2, 0) = q_1$
 $\delta(q_3, 1) = q_0$
 $\delta(q_3, 0) = q_2$

$q_0 = q_0$
 $F = q_0$
 $Q = \{q_0, q_1, q_2, q_3\}$
 $\Sigma = \{0, 1\}$
 transition function

Q. Design a DFA over the alphabet $\Sigma = \{0, 1\}$ and accept the language having an string with no. of ones odd number of 0's odd.

Let me be the DFA which accepts the described language

Sol.

Let the

regular DFA

$Q = \{q_0, q_1, q_2, q_3\}$

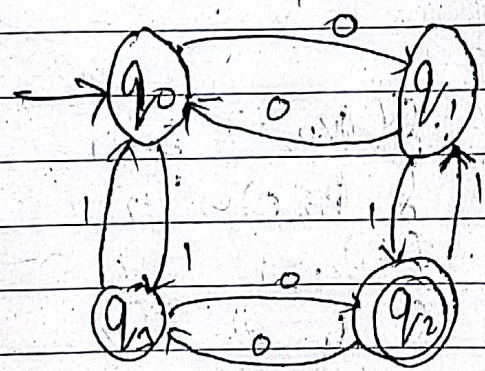
where,

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

$q_0 = q_0$

$F = q_2$

$L = \{01, 001, 10, 1110, 0010, 101010, 000111, 11100, 100110, \dots, 01, 10, 1101, 0010, \dots\}$



Transition function

$\delta: Q \times \Sigma \rightarrow Q$

$$\delta(q_0, 1) = q_3$$

$$\delta(q_0, 0) = q_1$$

$$\delta(q_1, 0) = q_0$$

$$\delta(q_1, 1) = q_2$$

$$\delta(q_2, 1) = q_1$$

$$\delta(q_2, 0) = q_3$$

$$\delta(q_3, 1) = q_0$$

$$\delta(q_3, 0) = q_2$$

Design a DFA that accepts the strings over the alphabet, $\Sigma = \{0, 1\}$ that either start with 01 ~~and~~ or end with 01. And test your design with two strings.

Let M be the DFA which accepts the described language.

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

Where,

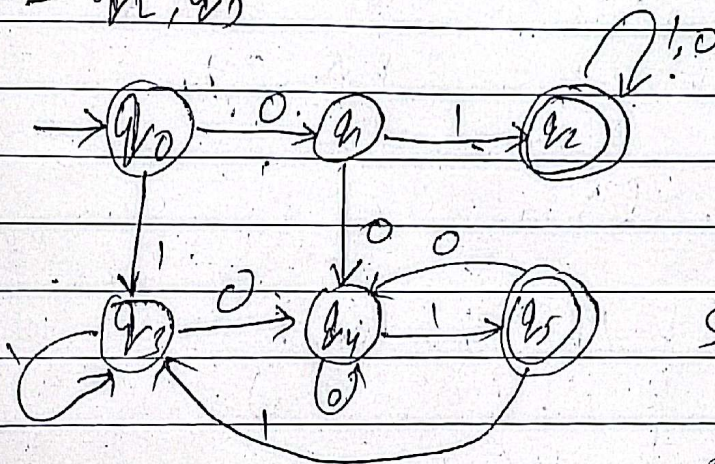
$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

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

$$q_0 = q_0$$

$$F = \{q_2, q_5\}$$

$$L = \{01, 0101, 010001, 001, 1001, 010101, \dots\}$$



Transition diagram

Transition function

$$\delta: Q \times \Sigma \rightarrow Q$$

State \ Σ	0	1
$\rightarrow q_0$	q_1	q_3
q_1	q_2	q_2
$* q_2$	q_3	q_2
q_3	q_4	q_3
q_4	q_4	q_5
$* q_5$	q_4	q_5

* And Check the string '0110' is accepted by your machine/DFA or not.

↳ Now,

Let Check

$\delta(q_0, 0110)$ $\vdash M(q_1, 110)$

$\vdash M(q_2, 10)$

$\vdash M(q_2, 0)$

$\vdash M(q_2, \epsilon)$

Hence After facing all inputs our machine halts at q_2 and q_2 is the final state. Hence the given string is accepted by the DFA. *Q5

* And check (1100)

$\delta(q_0, 1100) \vdash M(q_3, 100)$

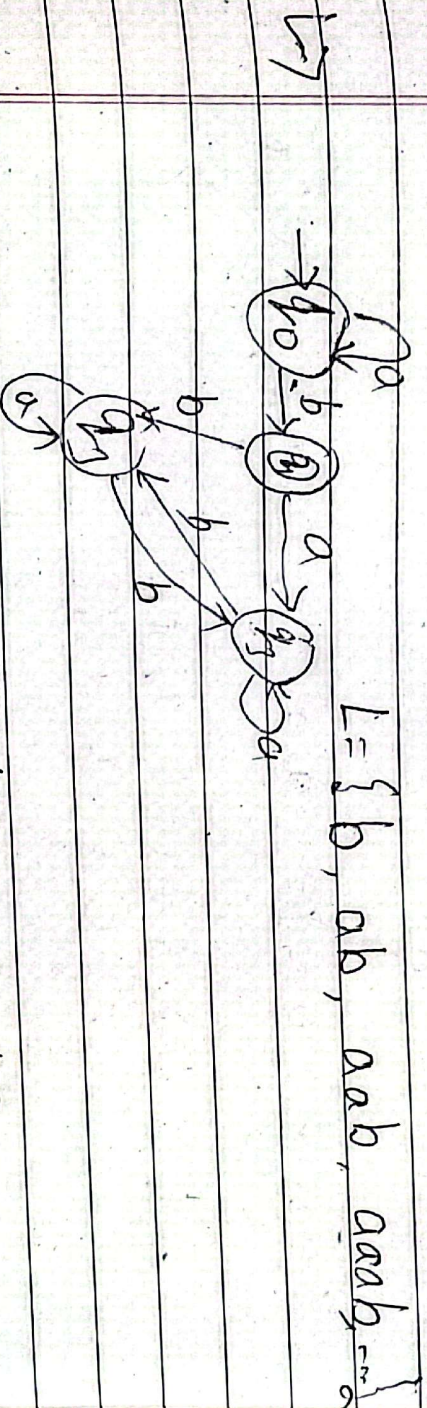
$\vdash M(q_3, 00)$

$\vdash M(q_4, 0)$

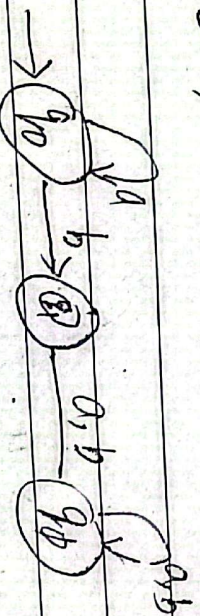
$\vdash (q_4, \epsilon)$

After facing all inputs our machine halts at q_4 and q_4 is not our final state. Hence the given string is not accepted by the DFA. *Q6

Q1 Design a DFA that accept the language defined by $L = \{a^n, b; n \geq 0\}$ over the alphabet $\Sigma = \{a, b\}$.



Q2, Trap state / dead state



Q1 NFA
 Design NFA which accepts the set of strings $w \in \{a, b\}^*$ where each string contains ^{either} two consecutive a's or two consecutive b's.

eg $L = \{aa, bb, ababaa, abaab, aabba, \dots\}$

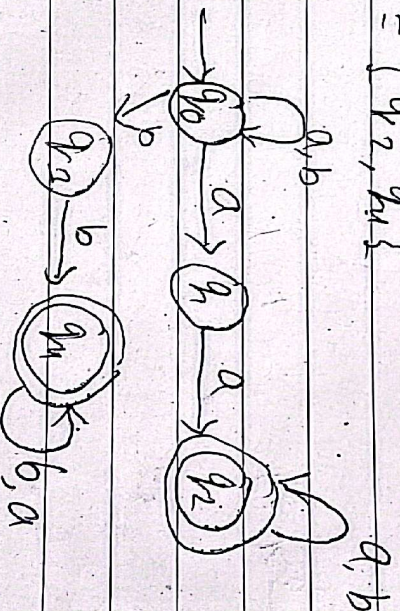
Let, the required NFA be

$$M = \{Q, \Sigma, q_0, F, \delta\}$$

$$\Sigma = \{a, b\}$$

$$q_0 = q_0$$

$$F = \{q_2, q_4\}$$



State \ Σ	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	q_2	$\emptyset$
$*q_2$	q_2	q_2
q_3	$\emptyset$	q_4
$*q_4$	q_4	q_4

$$ab^*a = a \cdot b^* \cdot a$$

$$ab^*a = a \cdot b^* \cdot a$$

$$= aa$$

Q. Design an NDFA or NFA accepts the language over $\Sigma = \{a, b\}$ where each string contains either ab^*a or b^*aa .

Ex: $L = \{aa, abba, aba, abbbba, bbbbaa, baa, \dots\}$

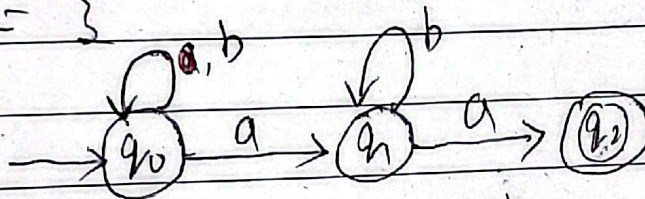
Let the required NDFA be

$$M = \{Q, \Sigma, q_0, F, \delta\}$$

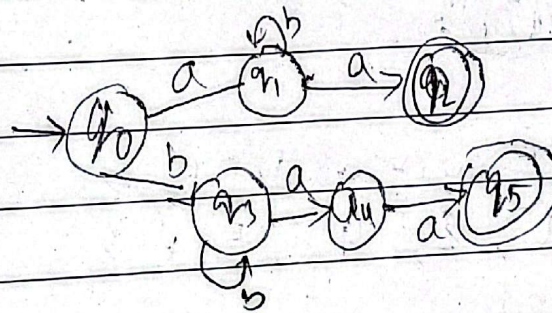
$$\Sigma = \{a, b\}$$

$$q_0 = q_0$$

$$F = \{ \}$$



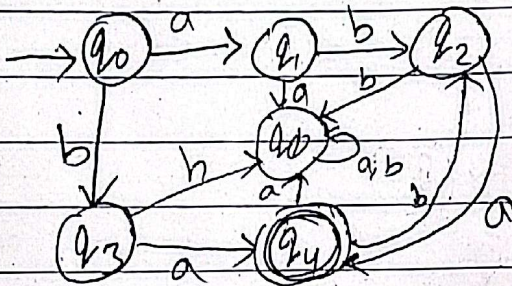
or,



20/05/2020

Q. Design a DFA that accepts the language over the alphabet $\Sigma = \{a, b\}^*$, w contains neither 'aa' nor 'bb' as a substring. And test your design with the string 'abaabb'.

Ex $L = \{ab, abab, ba, baba, babab, \dots\}$
 $\{aa, bb, aaba, bbab, ababba, \dots\}$
 $\{x, x, x, x, x\}$



Let, DFA requires DFA be

$M = \{Q, \Sigma, q_0, F, \delta\}$

$\Sigma = \{a, b\}$

$q_0 = q_0$

$F = \{q_2, q_4\}$

State \ Σ	a	b
$\rightarrow q_0$	q_1	q_3
q_1	q_3	q_2
$*q_2$	q_4	q_0
q_3	q_4	q_1
$*q_4$	q_1	q_3

let check,

$\delta(q_0, abab) \vdash M(q_1, bab)$
 $\vdash M(q_2, ab)$
 $\vdash M(q_4, b)$
 $\vdash M(q_2, \epsilon)$

Ans,

check,

$\delta(q_0, abaabb) \vdash M(q_1, baabb)$
 $\vdash M(q_2, aabb)$
 $\vdash M(q_4, abb)$
 $\vdash M(q_\phi, bb)$
 $\vdash M(q_\phi, b)$
 $\vdash M(q_\phi, \epsilon)$

After seeing all inputs our machine
 halts at q_ϕ and q_ϕ is not final
 state. Hence the given string is not
 accepted by the DFA $*q_2 \& q_4$.

Equivalence of DFA and NFA

- 1) For each NFA, there is an equivalent deterministic finite automata (DFA)
- 2) NFA can always be converted into Σ^* equivalent deterministic are:

Conversion steps:

- i) Observe all the transition from starting state q_0 for every symbol in Σ .
- ii) For new states from (i) check all transition for every Σ .
- iii) Repeat step (ii) till new steps are appeared.

Any language accepted by Σ^* NFA can also be accepted by some DFA. To do this Σ^* algorithm that takes any NFA and convert it into a DFA. The algorithm which is describe above is known as subset construction algorithm.

- 3) Formally, we say that two FA M_1 and M_2 (NFA and DFA) are equivalent if and only if $L(M_1) = L(M_2)$. Thus two automata are considered to be equivalent, if they accept the same language, even though they may use the different method to do so.

Example :-

Given, NFA

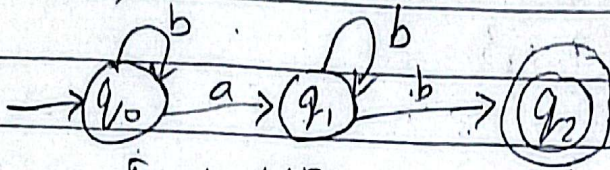


Fig: NFA

Convert into DFA.

Sol.

Step-1]

For initial state q_0 , new state, Not possible
 For DFA, $\{q_0\}$. DFA

$$\delta'(\{q_0\}, a) = \delta(q_0, a) = q_1 \text{ (new state)}$$

$$\delta'(\{q_0\}, b) = \delta(q_0, b) = q_0$$

Step-2] For new state q_1 ,

$$\delta'(\{q_1\}, a) = \delta(q_1, a) = \emptyset$$

$$\delta'(\{q_1\}, b) = \delta(q_1, b) = \{q_1, q_2\} \text{ new state}$$

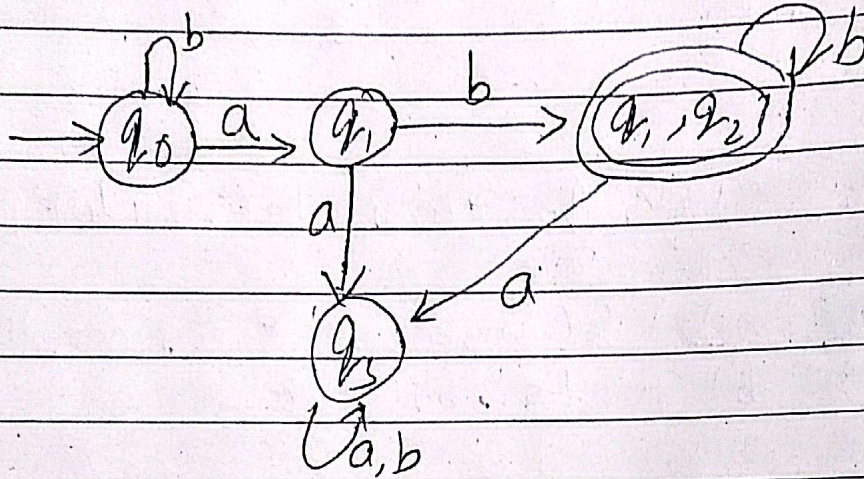
Step-3] For new state $\{q_1, q_2\}$

$$\delta'(\{q_1, q_2\}, a) = \delta(q_1, a) \cup \delta(q_2, a) = \emptyset \cup \emptyset = \emptyset$$

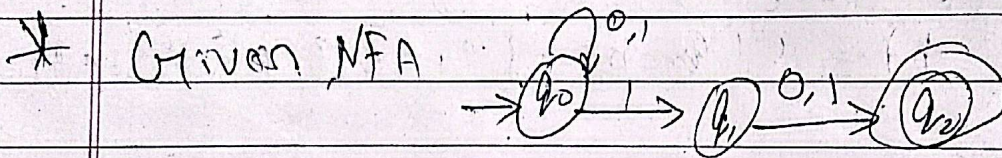
$$\begin{aligned}
 \delta'(\{q_1, q_2\}, b) &= \delta(q_1, b) \cup \delta(q_2, b) \\
 &= \{q_1, q_2\} \cup \{\emptyset\} \\
 &= \{q_1, q_2\}
 \end{aligned}$$

Now, our transition table: For DFA,

State \ I/P	a	b
$\rightarrow q_0$	q_1	q_0
q_1	$\emptyset = q_3$	$\{q_1, q_2\}$
$*\{q_1, q_2\}$	$\emptyset = q_3$	$\{q_1, q_2\}$
q_3	q_3	q_3



Final : DFA Figure.



Convert into equiv. DFA.

Soln Step-1 : For initial state q_0 , new state for DFA $\{q_0\}$

$$\delta'(\{q_0\}, 0) = \delta(q_0, 0) = q_0$$

$$\delta'(\{q_0\}, 1) = \delta(q_0, 1) = \{q_0, q_1\} \text{ new state}$$

Step-2 : For new state $\{q_0, q_1\}$

$$\delta'(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = q_0 \cup q_2$$

$$= \{q_0, q_2\} \text{ new state}$$

$$\begin{aligned}
 \delta'(\{q_0, q_1\}, 1) &= \delta(q_0, 1) \cup \delta(q_1, 1) \\
 &= \{q_0, q_1\} \cup q_2 \\
 &= \{q_0, q_1, q_2\} \text{ new state.}
 \end{aligned}$$

Step-3 : For new state $\{q_0, q_2\}$ and $\{q_0, q_1, q_2\}$

$$\begin{aligned}
 \delta'(\{q_0, q_2\}, 0) &= \delta(q_0, 0) \cup \delta(q_2, 0) = q_0 \cup \emptyset = q_0 \\
 \delta'(\{q_0, q_2\}, 1) &= \delta(q_0, 1) \cup \delta(q_2, 1) = \{q_0, q_1\} \cup \emptyset \\
 &= \{q_0, q_1\}
 \end{aligned}$$

And,

$$\begin{aligned}
 \delta'(\{q_0, q_1, q_2\}, 0) &= \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0) \\
 &= q_0 \cup q_2 \cup \emptyset \\
 &= \{q_0, q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \delta'(\{q_0, q_1, q_2\}, 1) &= \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1) \\
 &= \{q_0, q_1\} \cup q_2 \cup \emptyset \\
 &= \{q_0, q_1, q_2\}
 \end{aligned}$$

Now, our transition table for DFA,

State/In	0	1
$\rightarrow q_0$	q_0	$\{q_0, q_1\}$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	q_0	$\{q_0, q_1\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

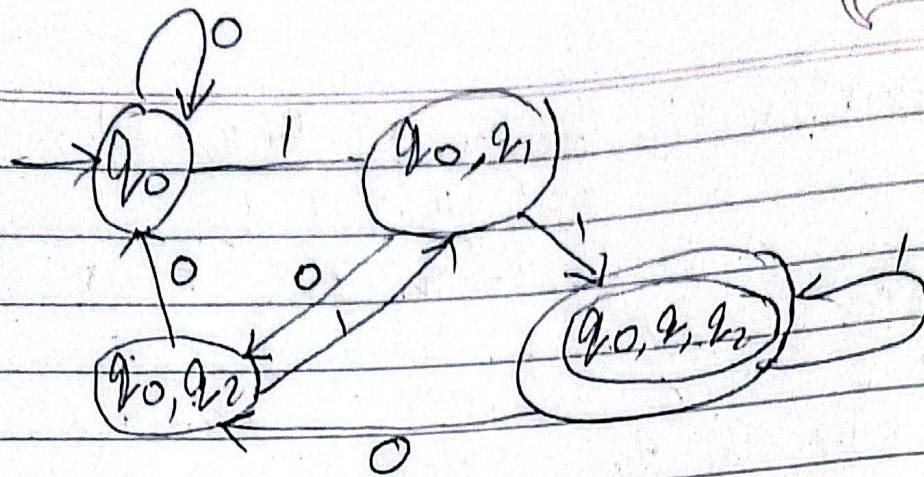


Fig: DFA

Regular Expression :- 2082101/30

- ↳ An expression used to generate strings for finite automata is called regular expression.
- ↳ It is a formula for representing a language in terms of a form combined using 3-operation.
- ↳ It describes a language exclusively by means of single symbol and combined perhaps with symbol $\cup$ and $\cap$ possibly with the help of parenthesis.
- ↳ The r.e over an alphabet Σ or a strings over the alphabet $\Sigma \cup \{\epsilon\} \cup \{^*\}$ that can be obtained as follows.
 - 1) $\emptyset$ and each number of Σ is regular Expression.
 - 2) If α and β are regular Expression then so is $(\alpha\beta)$

3) If α and β are regular expression so is $(\alpha \cup \beta)$

4) If α is regular expression then so is α^* . Nothing is r.e unless it follow from (1) through (4).

Eg. Let $L = \{w \in \{a,b\}^* ; w \text{ has even no. of } a\text{'s followed by odd no. of } b\text{'s}\}$

$\rightarrow L = \{b, aa, aab, aabbb, aaaaab, aaaaabb, \dots\}$
 $= (aa)^*(bb)^*b$ This is the regular expression.

Eg: Let $L = \{w \in \{0,1\}^* ; w \text{ has string of } 0\text{'s and } 1\text{'s ending with } 00\}$

$\rightarrow L = \{00, 100, 1100, 0100, 110100, 11100, \dots\}$
 $RE = (001)^* 00$

Regular operator :-

Basically, there are three operators that used to generate the languages that are regular union :-

If L_1 & L_2 are only 2 regular languages
 $L_1 \cup L_2 = \{s \mid s \in L_1 \text{ or } s \in L_2\}$

For eg: $L_1 = \{00, 11\}$, $L_2 = \{01, 10\}$
 $L_1 \cup L_2 = \{00, 11, 01, 10\}$

Concatenation :-

$$L_1 \cdot L_2 = \{ l_1 \cdot l_2 \mid l_1 \in L_1 \text{ \& } l_2 \in L_2 \}$$

eg :- $L_1 = \{00, 11\}$ and $L_2 = \{ \epsilon, 10 \}$

$$L_1 \cdot L_2 = \{00, 11, 0010, 1110\}$$

$$L_2 \cdot L_1 = \{00, 1000, 1011, 11\}$$

$$\text{So } L_1 \cdot L_2 \neq L_2 \cdot L_1$$

Precedence of regular operator.

- 1> The star operator is highest precedence.
- 2> Next precedence is taken by precedence operator.

3> Finally Union are taken.

→ Regular ~~operator~~ language :-

Let Σ be an alphabet, the class of reg. language over Σ is defined as :-

- ϕ is a regular language representing empty language.
- $\{ \epsilon \}$ is a reg. language representing language of empty string.
- For each $a \in \Sigma$ $\{a\}$ is a reg. language
- If $L_1, L_2, \dots, L_n$ is a R.L. then so is $L_1 \cup L_2 \cup \dots \cup L_n$.
- If $L_1, L_2, L_3, \dots, L_n$ is a R.L. then so is $L_1 \cdot L_2 \cdot L_3 \dots L_n$.

→ If L is a R.L. then so is L^*

NOTE: A regular expression E is just expression not a language. We should use $L(E)$ when we want to refer to the language that E denotes.

• Regular language:

— The relation between regular expression and the language they represent is established by a function L such that if α is any regular expression, then $L(\alpha)$ is language represented.

— The function L is defined as follows:

1) $L(\emptyset) = \emptyset$ and $L(a) = \{a\}$ for each $a \in \Sigma$

2) If α and β are regular expressions then
 $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$

3) If α and β are regular expressions then
 $L(\alpha\beta) = L(\alpha)L(\beta)$

4) If α is regular expression then
 $L(\alpha^*) = L(\alpha)^*$

Q. What language is represented by the following expression?

$L((a \cup b)^* a)$

→ $L((a \cup b)^*) \cup L(a)$

→ $(L(a) \cup L(b))^* \{a\}$

→ $(\{a\} \cup \{b\})^* \{a\}$

= $\{a, b\}^* \{a\}$

⇒ $L = \{w \in \{a,b\}^* ; w \text{ ends with } a\}$

2081/02/04

* Identities theorem for Regular Expression

- | | |
|---|--|
| 1. $\emptyset + R = R$ | 7. $RR^* = R^*R$ |
| 2. $\emptyset R + R\emptyset = \emptyset$ | 8. $(R^*)^* = R^*$ |
| 3. $\epsilon R = R\epsilon = R$ | 9. $\epsilon + RR^* = \epsilon + R^*R = R^*$ |
| 4. $E^*E \& \emptyset^* = E$ | 10. $(p\emptyset)^*p = p(\emptyset p)^*$ |
| 5. $R + R = R$ | 11. $(p+q)^* = (p^*q^*)^* = (p^* + q^*)^*$ |
| 6. $R^*R^* = R^*$ | 12. $(p+q)R = pR + qR \& R(p+q) = Rp + Rq$ |

* Conversion of Regular Expression to FA:

* Subset method is used to obtain FA from the given regular expression.

Step 1: construct transition diagram for a given regular expression by using NFA with ϵ -moves

Step 2: Convert NFA with ϵ to NFA without ϵ

Step 3: Convert NFA to Equivalent DFA.

* Union operator can be eliminated by introducing parallel edges between two states.

* Concatenation (dot) operator can be

eliminated by introducing new states between states.

Closure (*) can be eliminated by introducing self loops on states.

Some basic rules are as :-

1. $\emptyset \Rightarrow \rightarrow (q_0)$

atb
union

2. $a \Rightarrow \rightarrow (q_0) \xrightarrow{a} (q_f)$

ab
concatenation.

3. $a + b \Rightarrow \rightarrow (q_0) \xrightarrow{b} (q_1) \xrightarrow{a} (q_f)$

$\wedge^*$ $\rightarrow$ closure

4. $ab \Rightarrow \rightarrow (q_0) \xrightarrow{a} (q_1) \xrightarrow{b} (q_f)$

5. $ab^* \Rightarrow \rightarrow (q_0) \xrightarrow{a} (q_f) \xrightarrow{b}$

6. $(ab)^* \Rightarrow \rightarrow (q_0) \xrightarrow{a} (q_1) \xrightarrow{b} (q_0)$

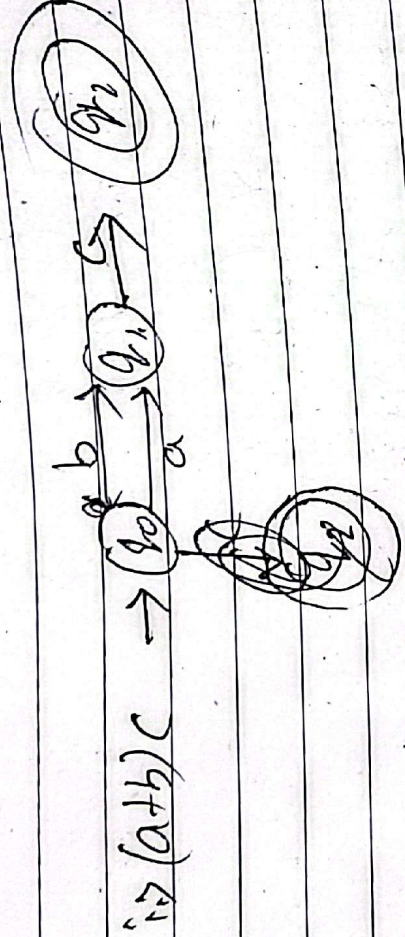
7. $(ab + ba)^* \Rightarrow \rightarrow (q_0) \xrightarrow{a} (q_1) \xrightarrow{b} (q_0) \xrightarrow{a} (q_2) \xrightarrow{b} (q_0)$

8. $(a + b)^* \Rightarrow \rightarrow (q_0) \xrightarrow{a, b} (q_0)$

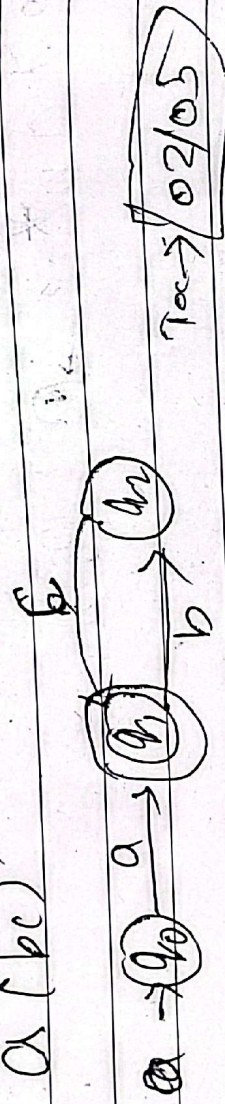
conversion of

eg:-

i.e. ba^*b

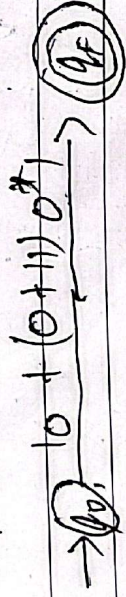


i.e. $a^*(bc)^*$

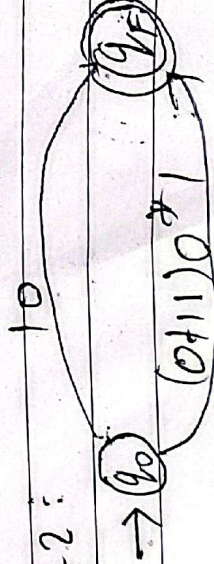


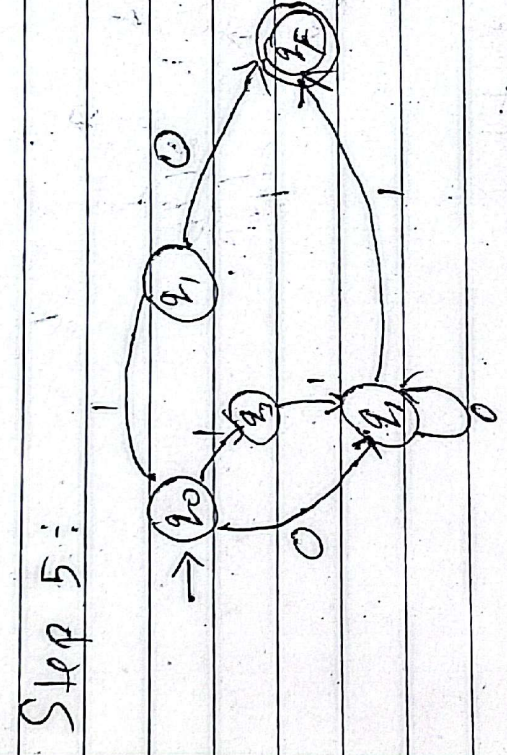
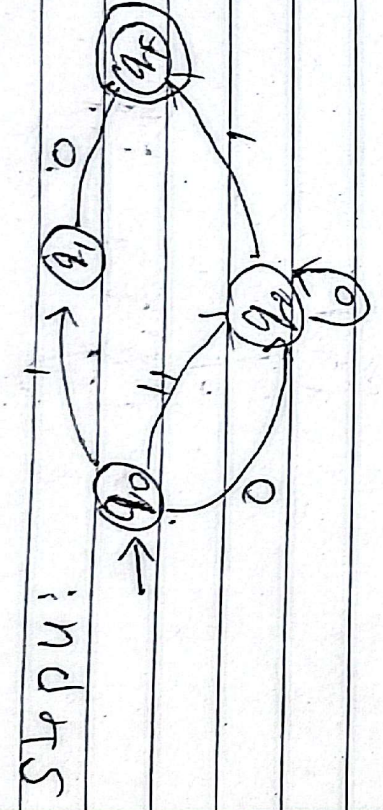
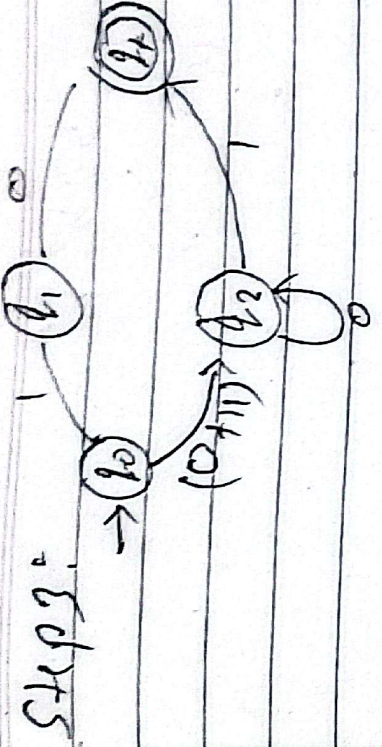
Q. Design a FA from the Regular expression
~~10 + (0+1)0*~~

sol Step 1:

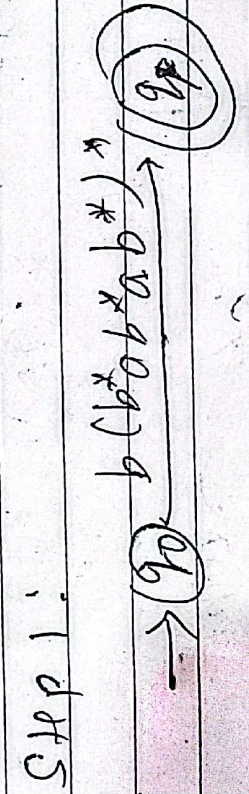


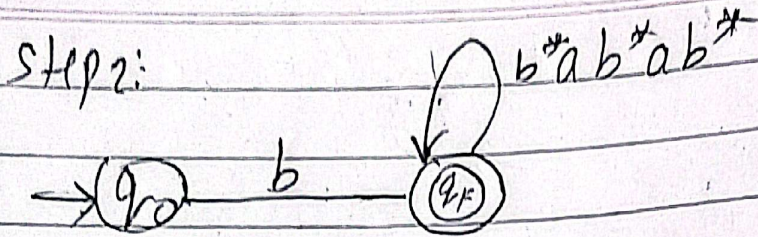
Step-2:



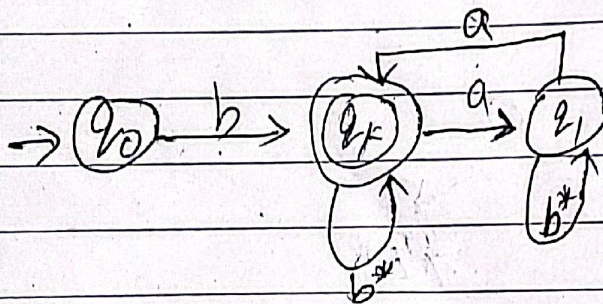
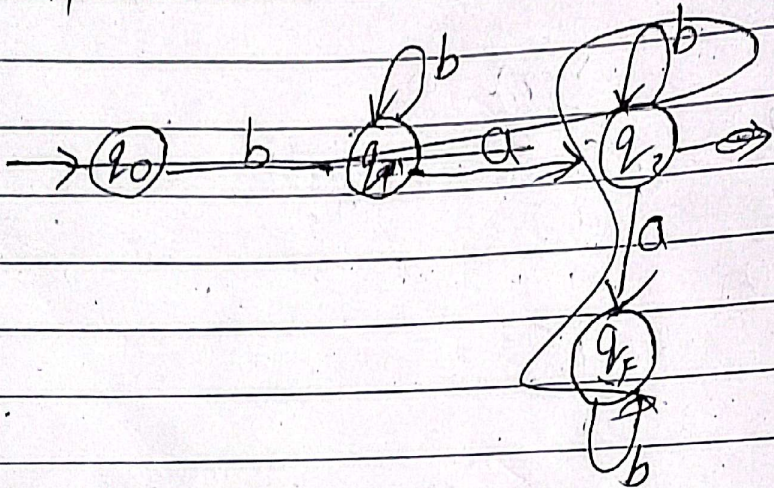


2. ~~3~~ $b(b^*ab^*ab^*)^*$



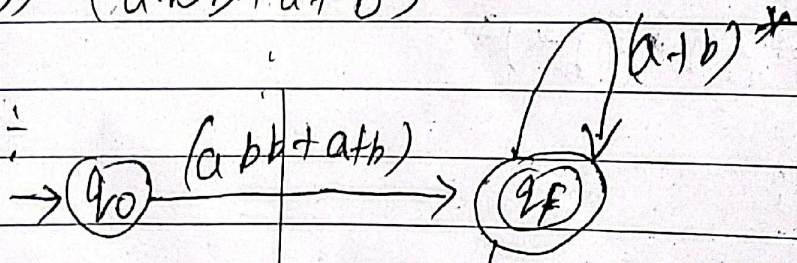


Step 3:

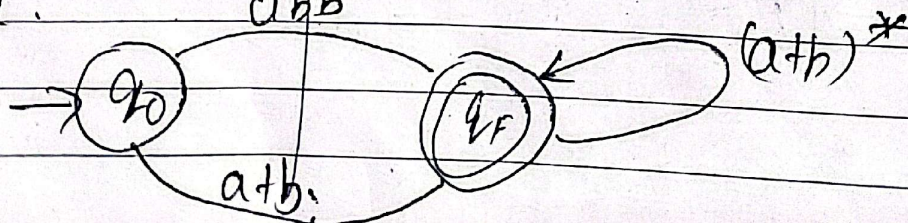


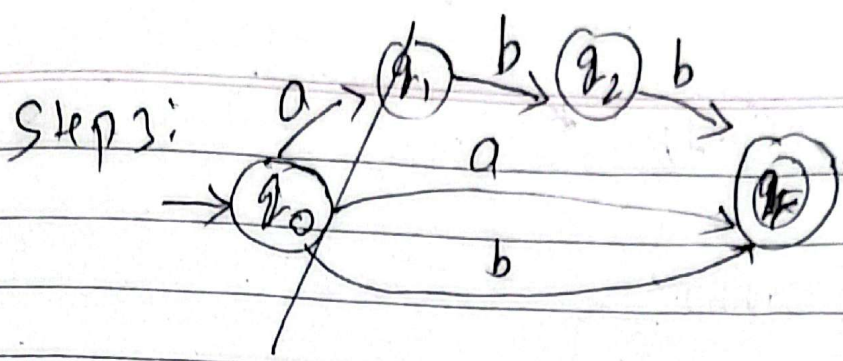
$(a+b)^*(a.b.b+a.b)$

Step 1:

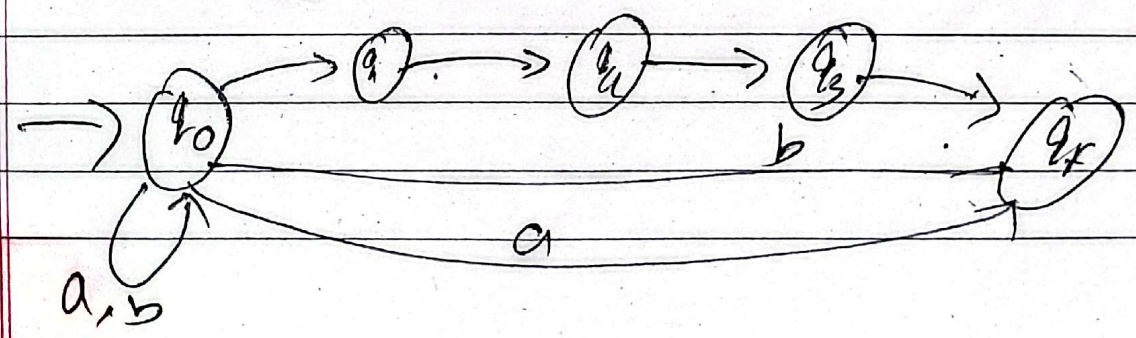
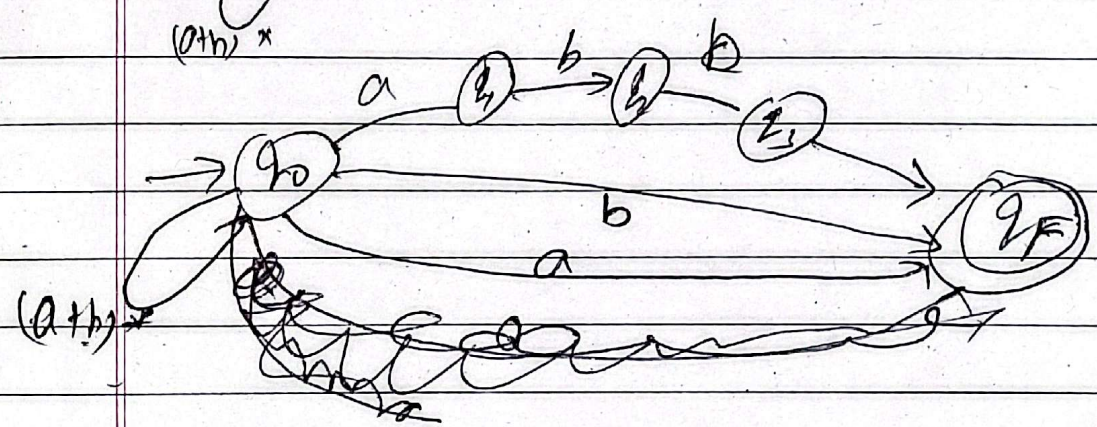
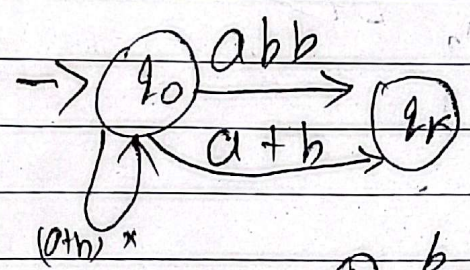
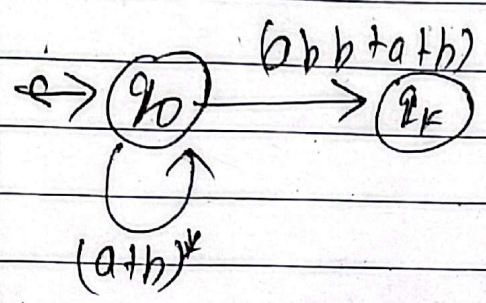


Step 2:

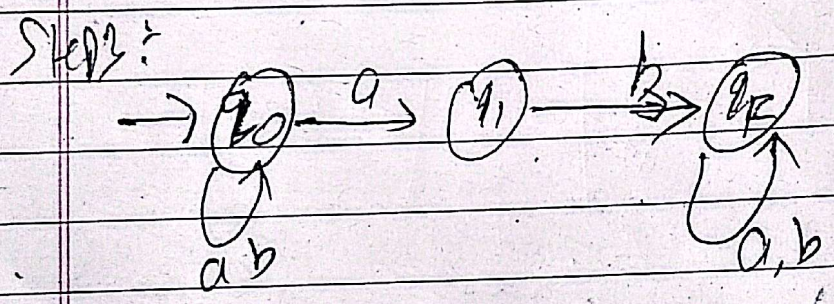
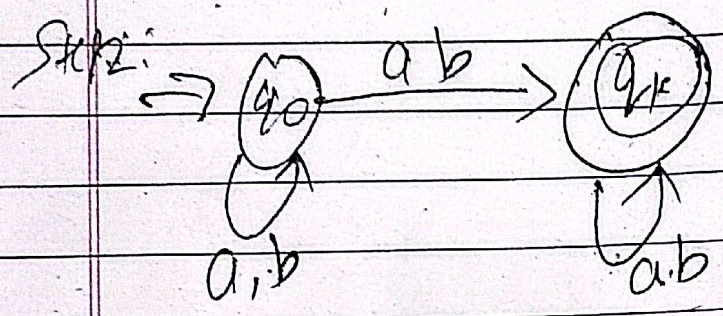
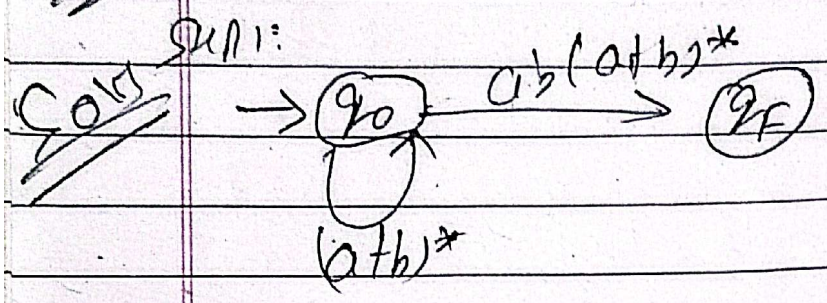




594 ✓



Q-7 $(a+b)^* a b (a+b)^*$



Properties of Regular language:

The properties of R.L can be broadly classified as two parts.

1> Closure properties

2> Decision properties

1> Closure properties of Regular Languages:

→ If a set of regular languages are combined using an operator, then the resulting language is also regular.

→ Closure property is a statement that a certain ~~property~~ operation on languages when applied to a language in a class produces a result is also in the class.

→ The regular language is closed under.

1> Union

2> Concatenation

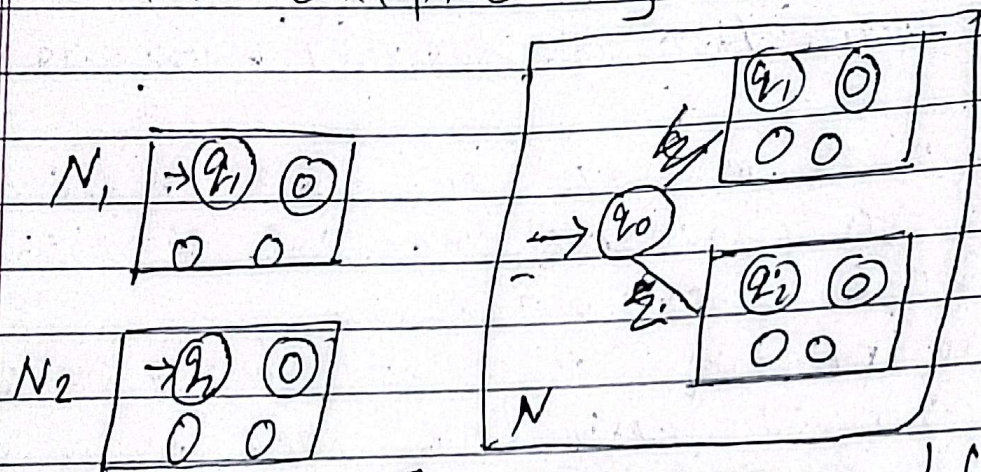
3> Kleene star

4> Complementation

1> Union:

→ The set of regular languages is closed under the union operation.

→ If L_1 and L_2 are regular then $L_1 \cup L_2$ is also a regular language. ~~let~~
 let A_1 & A_2 are two r.l. Now prove $A_1 \cup A_2$ are.
 Here, idea is take two NFA's N_1 and N_2 for A_1 and A_2 and combine them to one new NFA N such it accepts input i.e either accepted by N_1 and N_2 .



The NFA accepts $L(N_1) \cup L(N_2)$

let $N_1 = \{Q_1, \Sigma, \delta_1, q_1, F_1\}$ recognizes A_1
 $N_2 = \{Q_2, \Sigma, \delta_2, q_2, F_2\}$ recognise A_2

Now,

$N = (Q, \Sigma, \delta, q_0, F)$ is such that
 $L(N) = A_1 \cup A_2$

Here,

$Q = Q_1 \cup Q_2 \cup \{q_0\}$; q_0 is the start state $A \cup N$

$F = F_1 \cup F_2$

$$\begin{aligned} \delta(q_0, a) &= \delta_1(q, a) \text{ if } q \in Q_1 \\ &= \delta_2(q, a) \text{ if } q \in Q_2 \\ &= \{q_1, q_2\} \text{ if } q = q_0 \text{ and } a = \epsilon \\ &= \emptyset, \text{ if } q = q_0 \text{ and } q \neq \epsilon \end{aligned}$$

Pumping Lemma

To prove that a language is not regular using pumping lemma, follow the below steps:

(We prove using contradiction)

- 1) Assume that a language A is regular.
- 2) It has to have a pumping length (say p)
- 3) All strings longer than ' p ' can be pumped $|S| \geq p$.
- 4) Now find a string ' S ' in A s.t. $|S| \geq p$
- 5) Divide S into xyz .
- 6) Show that $xy^iz \notin A$ for some i .
- 7) Then consider all ways that S can be divided into xyz .
- 8) Show that none of these can satisfy all the three pumping conditions at the same time.
- 9) S cannot be pumped = contradiction.

Q) Using pumping lemma show that the language $A = \{a^n b^n \mid n \geq 0\}$ is not regular.

→ proof:

Assume that A is regular language
let pumping length = p

Our string, $S = a^p b^p$

let $p = 7$

$S = a^7 b^7 \Rightarrow aaaaaaabbbbbbb$

Case 1: Assume y is in 'a' part.

$\underbrace{aaaaaa}_x \underbrace{a}_y \underbrace{bbbbbb}_z$

Case 2: Assume y is in 'b' part

$\underbrace{aaaaaa}_x \underbrace{bb}_y \underbrace{bbbbb}_z$

Case 3: Assume y is in 'ab' part.

$\underbrace{aaaaa}_x \underbrace{ab}_y \underbrace{bbbbb}_z$

let $i = 2$

pumping the value of i
from case (I)

The resulted string is

case I xy^2z

$\underbrace{aaaaaaaaa}_x \underbrace{aa}_y \underbrace{bbbbbb}_z \notin A$

again, from case II

over string is xy^2z

$\underbrace{aaaaaa}_x \underbrace{bbb}_y \underbrace{bbbbb}_z \notin A$

Again, from case III

over string is xy^2z

$a a a a a a a b b a a b b b b b b b \notin A$

Here, all the three conditions are not satisfied and hence there is contradiction.

Hence we can say that the language A is not regular.

Imp Q7

Using pumping lemma prove that the language

$A = \{xy \mid y \in \{0,1\}^*\}$ is not regular

Solⁿ

Assume that A is regular language

Let pumping length = p .

our String, $S = 0101$

Let $p = 5$

$S = 01^p 01^p / 0^p 1^p 0^p 1^p$

Case 1: Assume x is in '1' part

01111101111

--- continues

Chapter-1

Q.1) Generate the string 00101 from the given grammar.

$$S \rightarrow AID$$

$$\begin{aligned}
 \Rightarrow S &\rightarrow 0AID \quad (\text{using u.r production } A \rightarrow 0A) \\
 &\rightarrow 00AID \quad (\text{using u.r production } A \rightarrow 0A) \\
 &\Rightarrow 001D \quad (\text{using production } A \rightarrow \epsilon)
 \end{aligned}$$

$$\begin{aligned}
 &\rightarrow 0010D \quad (\text{using production u.r } D \rightarrow 0D) \\
 &\rightarrow 00101D \quad (\text{using production u.r } D \rightarrow 1D)
 \end{aligned}$$

$$\begin{aligned}
 &\rightarrow 00101 \quad (\text{using production u.r } D \rightarrow \epsilon)
 \end{aligned}$$

Q.2) Consider the following grammar.

$$S \rightarrow aAS | aS | \epsilon$$

$$A \rightarrow SbA | ba$$

Generate the string aabaa using both left most and rightmost derivation.

$$\begin{aligned}
 S &\rightarrow aAS \\
 &\rightarrow aaAS
 \end{aligned}$$

left most derivation

$S \rightarrow ass$

$\rightarrow aaass \quad (S \rightarrow aAs)$

$\rightarrow aabass \quad (A \rightarrow ba)$

$\rightarrow aabasss \quad (s \rightarrow ass)$

$\rightarrow aabaass \quad (s \rightarrow \epsilon)$

$\rightarrow aabaas \quad (s \rightarrow \epsilon)$

$\rightarrow aabaa \quad (s \rightarrow \epsilon)$

~~$S \rightarrow aAs$~~

~~$\rightarrow aaass \quad (s \rightarrow ass)$~~

~~$\rightarrow aabass$~~

Right most derivation

~~$S \rightarrow ass$~~

~~$\rightarrow asaAs \rightarrow s \rightarrow aAs$~~

~~$\rightarrow asaAs \rightarrow (s \rightarrow \epsilon)$~~

~~$\rightarrow aasaass \rightarrow (s \rightarrow ass)$~~

~~$\rightarrow aabaasss \rightarrow (s \rightarrow ass)$~~

~~$\rightarrow aaba$~~

$\rightarrow aabaass \rightarrow (A \rightarrow ba)$

$\rightarrow aabass \rightarrow (s \rightarrow \epsilon)$

$\rightarrow aabaa \rightarrow (s \rightarrow \epsilon)$

$\rightarrow aabaa$

$S \rightarrow ass$

$\rightarrow asaAs \quad (s \rightarrow aAs)$

$\rightarrow asaAass \quad (s \rightarrow ass)$

$\rightarrow asaAas \quad (s \rightarrow \epsilon)$

$\rightarrow asaAa \quad (s \rightarrow \epsilon)$

$\rightarrow asabaa \quad (A \rightarrow ba)$

$\rightarrow aabaa \quad (s \rightarrow \epsilon)$

Q.1) Let $U = \{V, \tau, p, S\}$
 where, $V = \{S\}$
 $\tau = \{a, b\}$
 $S = \{S\}$
 $p = \{S \rightarrow asb, S \rightarrow ab\}$

Find the language $L(U)$ generated by the above CFG

Sol.

- i) $S \rightarrow ab$
- ii) $S \rightarrow asb$
 $\rightarrow aabb$
- iii) $S \rightarrow asb$
 $\rightarrow aasbb$
 $\rightarrow aaasbbb$

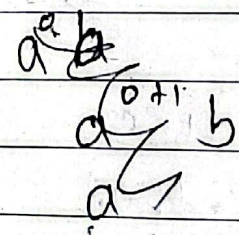
$$\begin{aligned}
 &a^{n-1} s b^{n-1} \\
 &a^{n-1} ab b^{n-1} \\
 &a^{n-1+1} b^{n-1+1} \\
 &a^n b^n \quad \checkmark
 \end{aligned}$$

Here, a language of the given grammar
 is $L(U) = \{a^n b^n, n \geq 1\} \quad \checkmark$

Q.2

over some : n Q.13
 $p = \{S \rightarrow aas, S \rightarrow b\}$

- i) $S \rightarrow b$
- ii) aas
 $\rightarrow aab$
- iii) aas
 $\rightarrow aaaaas$
 $\rightarrow aaaaaab$



$\{b, aab, aaaaab, \dots\}$

Hence the language of the given grammar
 is $L(U) = \{a^{2n}b, n \geq 0\}$
 $V = \{S, A\}$

Q7 $P \rightarrow \{S \rightarrow aA \mid \epsilon$
 $A \rightarrow bs\}$

Ex
 i) $S \rightarrow aA$
 $\rightarrow abs$
 $\rightarrow ab \quad (S \rightarrow \epsilon)$

ii) $A \rightarrow bs$
 $\rightarrow baA$
 $\rightarrow babs \quad (S \rightarrow \epsilon)$
 $\rightarrow bab$

iii) $S \rightarrow aA$
 $\rightarrow abs$
 $\rightarrow abaA$
 $\rightarrow ababs$
 $\rightarrow abab \quad (S \rightarrow \epsilon)$

$\{ \underline{ab}, \underline{abab}, \underline{ababab}, \dots \}$

Hence, the languages of the given grammar
 is $L(U) = \{(ab)^n, n \geq 0\}$

Ambiguity in Grammar

A grammar $G = \{V, T, P, S\}$ is said to be ambiguous, if there is a string $w \in L(G)$ for which we can derive two or more distinct derivation tree rooted at S and yielding w .

In other words, a grammar is ambiguous, if it can produce more than one leftmost or more than one rightmost derivation for the same string in the language of the grammar.

Example: $S \rightarrow AB|aAb$

$A \rightarrow a|Aa$

$B \rightarrow b$

For New string aab .

We have two leftmost derivation as:

i. $S \rightarrow AB$

$\rightarrow Aab$

$\rightarrow aab$

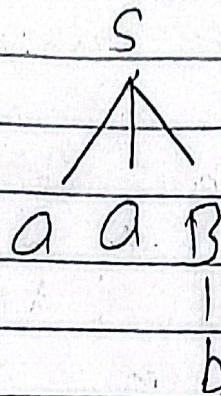
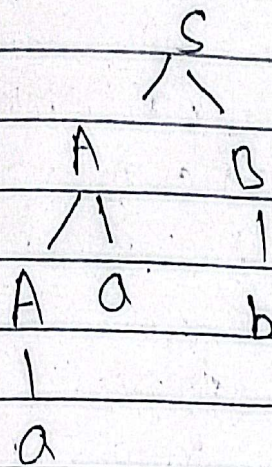
$\rightarrow aab$

ii. $S \rightarrow$

aAp

$\rightarrow aab$

Parse tree



* Simplification of Grammar :-

1. Eliminate useless symbols.
2. Eliminate empty (ϵ) productions.
3. Eliminate unit productions.

In CFG, sometimes all the production rules and symbols are not needed for the derivation of strings. Besides this there may also be some NULL/empty productions and unit productions.

Elimination of these productions and symbols is called Simplification of CFG.

1. Eliminating useless symbols.

- ↳ Useless symbols are those variables or terminales that do not appear in any derivation of a terminal string from the start symbol.
- ↳ Elimination of useless symbols include identifying whether or not the symbol is "generating" and "reachable."
- ↳ A symbol x is generating if $x \xrightarrow{*} w$ for some terminal string w .
- ↳ A symbol x is reachable if there is derivation $S \xrightarrow{*} \alpha x \beta$ for some α and β .
- ↳ Thus we eliminate non-generating and non-reachable symbols.

Eg: Consider a grammar rules:

$S \rightarrow abs / abA / abd$

$A \rightarrow cd$

$B \rightarrow ab$

$C \rightarrow dc$

In above rules $C \rightarrow dc$ is useless, because the variable 'c' will not occur in derivation of any strings.

The other productions are written in such a way that variable 'c' can never reach from the start symbol 'S'.

Production $B \rightarrow ab$ is also useless, because there is no way it will ^{ever} terminate, then it can never produce a string.

We then try to identify all the variable that can never be reached from the starting variable such as var 'c'.

We then remove all the production in which variable 'c' occurs.

Then below grammar is now free of useless / production symbols.

$$S \rightarrow abs / abA$$

$$A \rightarrow cd$$

2) Eliminating Empty (ϵ) production.

- ↳ A grammar is said to have empty production if there is a production of the form $A \rightarrow \epsilon$
- ↳ Here we have to discover nullable variable.
- ↳ A variable A is nullable if $A^+ \rightarrow \epsilon$

Procedure for Removal:

- Step-1: To remove $A \rightarrow \epsilon$, look for all productions whose right side contains A .
- Step-2: Replace each occurrence of A in each of these productions with ϵ .
- Step-3: Add the resultant productions to the grammar.

Ex: Remove (ϵ) null productions from the following grammar.

$$S \rightarrow ABAC$$

$$A \rightarrow aA | \epsilon$$

$$B \rightarrow bB | \epsilon$$

$$C \rightarrow c$$

~~So~~ Here we have two null (ϵ) production,
 i.e. $A \rightarrow \epsilon$
 $B \rightarrow \epsilon$

$\therefore$ Both A and B are nullable.

To eliminate

$$A \rightarrow \epsilon$$

production which contain A are:

$$S \rightarrow ABAC$$

$$A \rightarrow aA$$

$$(1) S \rightarrow ABAC$$

$$S \rightarrow BAC | ABC | BC \quad \checkmark$$

$$(2) A \rightarrow aA$$

$$A \rightarrow a$$

Now, we get:

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

$$A \rightarrow aA | a$$

$$B \rightarrow bB | \epsilon$$

$$C \rightarrow c$$

Similarly, $B \rightarrow \epsilon$

productions which contain B are

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

$$B \rightarrow bB$$

Then,

$$(i) S \rightarrow ADAC$$

$$S \rightarrow AAC$$

$$(ii) S \rightarrow BAC$$

$$\rightarrow S \rightarrow AC$$

$$(iii) S \rightarrow ABC$$

$$S \rightarrow AC$$

$$(iv) S \rightarrow BC$$

$$S \rightarrow C$$

$$(v) D \rightarrow bB$$

$$D \rightarrow b$$

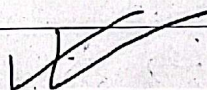
NOW, OUR Final CFG is

$$S \rightarrow ADAC / BAC / ABC / DC / AAC / AC / C$$

$$A \rightarrow AA / a$$

$$D \rightarrow bB / b$$

$$C \rightarrow c$$



3.7 Eliminating Unit Productions:

Any production rule of the form $A \rightarrow B$ where A, B are non-terminals, i.e. may be both variables is called Unit Production.

Procedure for removal:

Step 1: To remove $A \rightarrow B$, add production

$A \rightarrow x$ to the grammar rule whenever

$B \rightarrow x$ occurs in the grammar.

[$x \in$ Terminal and it can be nulls]

Step 2: Delete $A \rightarrow B$ from the grammar.

Step 3: Repeat from Step 1 until all

Unit productions are removed.

Example: Remove unit productions from

the grammar where production rules

are given by:

$S \rightarrow XY$

$X \rightarrow a$

$Y \rightarrow z/b$

$Z \rightarrow M$

$M \rightarrow N$

$N \rightarrow a$

Soln Here we have 3 unit productions
i.e. $y \rightarrow z$, $z \rightarrow M$, $M \rightarrow N$

Since we have $M \rightarrow a$ we replace with
 $M \rightarrow a$

Again, we have $M \rightarrow a$, it can be written
as $z \rightarrow a$

also, we have $z \rightarrow a$, we can replace
 $x \rightarrow z$ with $x \rightarrow a$.

Now, we get,

$$S \rightarrow xy$$

$$x \rightarrow a$$

$$y \Rightarrow a/b$$

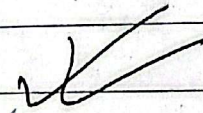
$$N \rightarrow a$$

Remove unreachable symbol,

$$S \rightarrow xy$$

$$x \rightarrow a$$

$$y \rightarrow a/b$$



Normal Form of CFG:

↳ If certain restrictions are applied to the production rules of CFG, then it is said to be in Normal Form.

↳ There are two types of normal forms:

1) Chomsky Normal Form (CNF)

2) Greibach Normal Form (GNF)

1) Chomsky Normal Form (CNF)

A CFG is said to be in CNF if its production rules are of the form:

Non-terminal $\rightarrow$ Exactly one terminal

Non-terminal $\rightarrow$ Exactly two non-terminals

Thus a grammar in CNF is one which should not have ϵ -production and useless symbol.

eg: $S \rightarrow a$

$S \rightarrow AB$ #

Conversion of CFG to CNF:

↳ To convert a given CFG to its equivalent CNF, substitution method is used.

→ First replace all terminal symbols by Non-Terminal Symbol. It is possible only after adding the respective production rule to the given CFG.

eg: $S \rightarrow aA$, to convert it into CNF,
we replace 'a' by Non-terminal let say C ,
i.e. $S \rightarrow CA$
 $C \rightarrow a$

So we add this rule to the given grammar.
we continue until all the production rules are connected.

Q.7

Construct the following CFG to its equivalent CNF.

$$S \rightarrow aAS/as$$

$$A \rightarrow aBb$$

$$B \rightarrow bB/bAa$$

soln

For First rule:

$$S \rightarrow aAS \text{ gives } S \rightarrow C_1AS \text{ where, } C_1 \rightarrow a$$

$$S \rightarrow C_1AS \text{ gives } S \rightarrow C_1C_2 \text{ where, } C_2 \rightarrow AS$$

Second rule:

$$S \rightarrow C_1S$$

Third rule: A.

$$A \rightarrow aBb \text{ gives } A \rightarrow C_1BC_3 \text{ where } C_3 \rightarrow b$$

$A \rightarrow C_1 B C_3$ gives $A \rightarrow C_1 C_4$ where $C_4 \rightarrow B C_3$

Again,

Fourth rule

$B \rightarrow b B$ gives $B \rightarrow C_2 B$

Fifth rule

$B \rightarrow b A a$ gives $B \rightarrow C_3 A C_1$

$B \rightarrow C_3 A C_1$ gives $B \rightarrow C_3 C_5$ where $C_5 \rightarrow A C_1$

Let $C_i = (V', \Sigma', p', S')$ be the equivalent CAF
where,

$$V' = \{ S, A, B, C_1, C_2, C_3, C_4, C_5 \}$$

$$\Sigma' = \{ a, b \}$$

$$p' = \{ S \rightarrow C_1 C_2 / C_1 S$$

$$A \rightarrow C_1 C_4$$

$$B \rightarrow C_2 C_5 / C_2 B$$

$$C_1 \rightarrow a$$

$$C_2 \rightarrow A s$$

$$C_3 \rightarrow b$$

$$C_4 \rightarrow B C_2$$

$$C_5 \rightarrow A C_1 \}$$

$$S' = \{ S \} \quad \checkmark$$

2) Greibach Normal Form (GNF)

→ A CFG is said to be in CNF, if its productions rules are of the form.
where $\alpha \rightarrow \alpha \beta^*$

where,

$\alpha, \beta \in V$ and $a \in T$

Non-Terminal $\rightarrow$ exactly one terminal

OR

Non-Terminal $\rightarrow$ one terminal followed by any number of non-terminals.

eg:-

$S \rightarrow a, S \rightarrow aA, S \rightarrow aAB$ etc.

Example:-

$S \rightarrow AB$

$A \rightarrow aA \mid bB \mid b$

$B \rightarrow b$

Algorithm / Rules to convert CNF to GNF

Step 1:- Check if the given CFG has unit production or null production.
Remove if there is any.

Step 2: Check whether the CFG is already in CNF and convert it to CNF if it is not.

Step 3: If there is left recursion remove it.

Solve

Replace A in $S \rightarrow AB$

$S \rightarrow aAB$ ($A \rightarrow aA$)

$S \rightarrow bBB$ ($A \rightarrow bB$)

$S \rightarrow bB$ ($A \rightarrow b$)

Now our production rules:

$S \rightarrow aAB / bBB / bB$

$A \rightarrow aA / bB / b$

$B \rightarrow b$

§ Pushdown Automata (Introduction)

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

A stack is a way we arrange elements one on top of another.

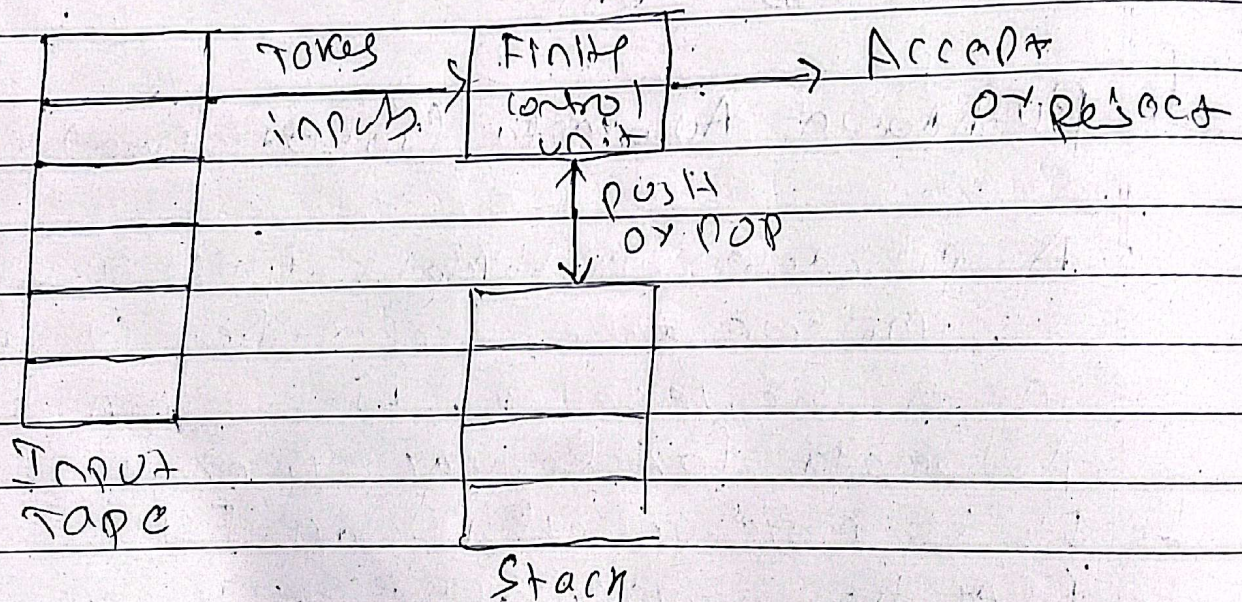
A stack does two basic operations.

push : A new element is added at the top of the stack.

pop : The top element of the stack is read and removed.

A pushdown Automata has 3 components.

- 1) An input tape
- 2) A Finite control unit
- 3) A stack with infinite size



Pushdown Automata (Formal Definition)

A pushdown Automata is formally defined by 7 Tuples as shown below.

$$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

Where,

Q = A finite set of states

Σ = A finite set of Input symbols.

Γ = A finite stack Alphabet.

δ = The transition function

q_0 = The start state

z_0 = The start stack symbol

F = The set of final / Accepting states.

δ takes as argument a triple $\delta(q, a, x)$

Where,

i) q is a state in Q

ii) a is either an input symbol in Σ or $a = \epsilon$

iii) x is a stack symbol, that is a member of Γ .

The output of δ is finite set of pair (p, Y)

Where,

p is new state

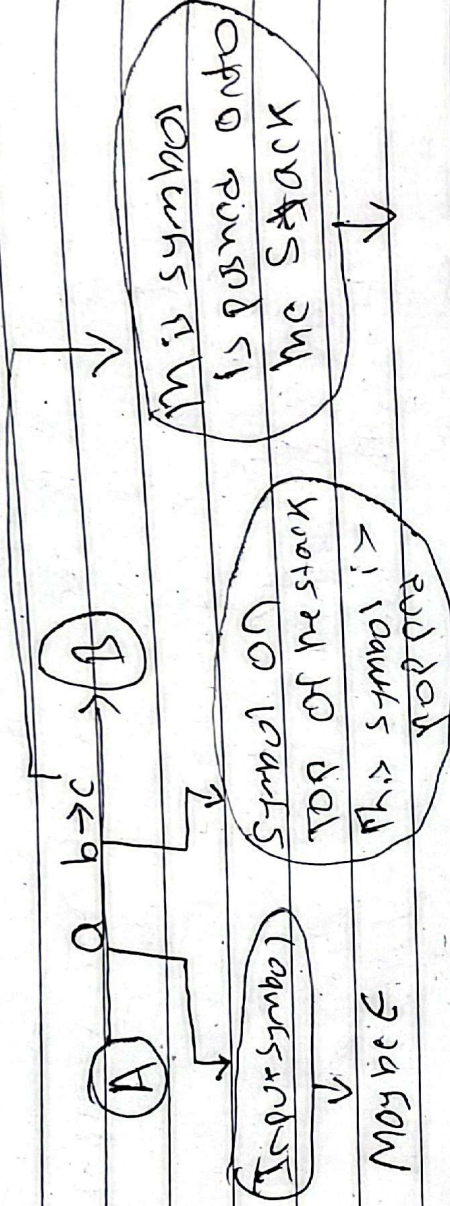
Y is a string of stack symbols that replaces x at the top of the stack.

Pushdown Automata (Graphical Notation)

Finite State Machine



Pushdown Automata

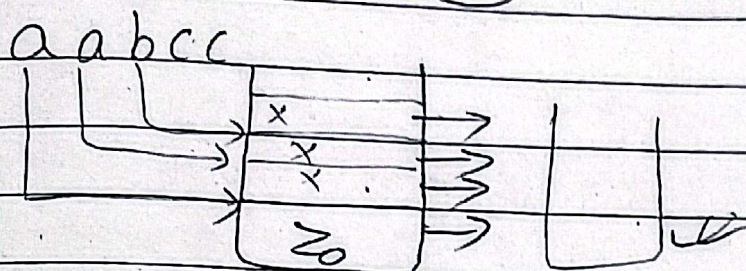
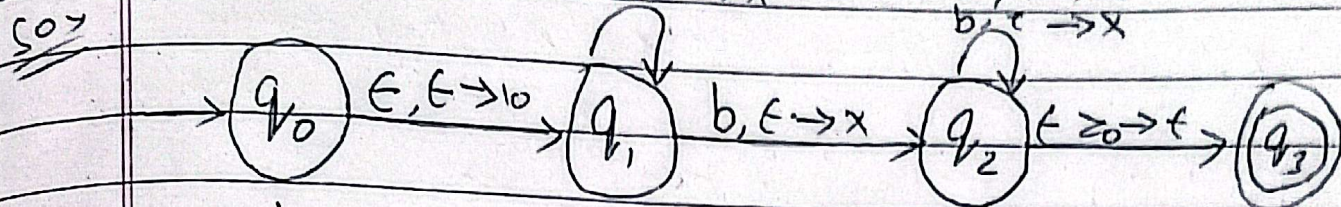


ε means nothing
ε means he

Stack is either read
or popped.

Q. Design a PDA for accepting the following language:

$$L = a^i b^j c^k / i, j, k > 0; i + j = k$$



Let, PDA $M = \{Q, \Sigma, \Gamma, \delta, q_0, F, Z\}$

where,

$$Q = \{q_0, q_1, q_2, q_3\}$$

$$\Sigma = \{a, b, c\}$$

$$\Gamma = \{z_0, x\}$$

$$q_0 = \{q_0\}$$

$$F = \{q_3\}$$

$$Z = \{z_0\}$$

$$\delta = \delta(q_0, \epsilon, z_0) \rightarrow (q_1, z_0)$$

$$\delta(q_1, a, z_0) \rightarrow (q_1, xz_0)$$

$$\delta(q_1, a, x) \rightarrow (q_1, xx)$$

$$\delta(q_1, b, x) \rightarrow (q_2, xx)$$

$$\delta(q_2, b, x) \rightarrow (q_2, xxx)$$

$$\delta(q_2, c, x) \rightarrow (q_2, xx)$$

$$\delta(q_2, \epsilon, z_0) \rightarrow (q_3, z_0)$$

IOC introduction

$$p(n) = 1 + 2 + \dots + n = \frac{n(n+1)}{2}$$

$$p(k) = 1 + 2 + \dots + k = \frac{k(k+1)}{2}$$

$$p(k+1) = 1 + 2 + \dots + k + (k+1) = \frac{(k+1)(k+2)}{2}$$

Q. $n^3 + 2n$ is divisible by 3.

For 1:

$$1^3 + 2 \times 1 = 3 \text{ is divisible by 3.}$$

For k:

$$3m = k^3 + 2k \text{ is divisible by 3}$$

For (k+1):

$$(k+1)^3 + 2(k+1)$$

$$k^3 + 1 + 3k^2 + 3k + 2(k+1)$$

$$k^3 + 1 + 3k(k+1) + 2k + 2$$

$$3m + 3k(k+1) + 3$$

$$3m + 3\{k(k+1) + 1\}$$

is divisible by 3 ✓

Equivalence of PDA and context free grammar (CFG):

* To show the languages defined by PDA are exactly the context free grammar.

↳ The main goal of equivalence is to prove that CFG and PDA accept the same class of language.

i) From CFG to PDA:

Given $CFG, G = (V, \Sigma, P, S)$, we construct a PDA M that accepts language generated by G .

The machine can be defined as

$$M = \{Q, \Sigma, \Gamma, V, \delta, q, S, \emptyset\}$$

where,

$$Q = \{q\} \text{ (is only the state in PDA)}$$

$$\Sigma = \Sigma$$

$$\Gamma = VUT \text{ (i.e. PDA uses terminals and variables of } G \text{ as stack symbol)}$$

$$z_0 = S$$

$$F = \emptyset$$

$$q_0 = q$$

δ can be defined as:

- i) $\delta(q, \epsilon, A) = (q, \alpha)$ where, $A \rightarrow \alpha$ is a production. p of G .
- ii) $\delta(q, a, a) = (q, \epsilon)$ for all $a \in \Sigma$ Here transition occurs in only one state.

Alternatively:

We can define PDA for CFG with two states p and q where p being start state.

Here, idea is that the stack symbol initially is support to be ϵ and at first PDA start with state p and reading ϵ , it insert start symbol ' S ' of CFG into stack and changes the state to q . Then all transition occurs in state q .

i.e. PDA can be defined as

$M = (\{p, q\}, \Sigma, V \cup T, \delta, \{q\}, p)$ i.e. the δ can be defined as

- $\delta(p, \epsilon, \epsilon) = (q, S)$; ' S ' is start symbol of G .
- $\delta(q, \epsilon, A) = (q, \alpha) : A \rightarrow \alpha$
- $\delta(q, a, a) = (q, \epsilon) : \text{for all } a \in \Sigma$

Example:

Let $G = (V, T, p \& s)$ where

$$p \rightarrow \varepsilon \rightarrow T \mid \varepsilon + T$$

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

$$F \rightarrow a \mid c \mid \varepsilon$$

We can define a PDA equiv to the grammar as.

$$M = (\{q_0\}, \{q_0, *, *, c\}, \{a, *, +, c\}, \{T, F\}, \delta, (\varepsilon, \emptyset))$$

Now with this PDA M ,

Check the string $a + (a * a)$ is accepted or not now,
defined δ as:

$$\delta(q_0, \varepsilon, F) = \delta(q_0, T), (q_0, F + T)$$

$$\delta(q_0, \varepsilon, T) = \delta(q_0, F), (q_0, T * F)$$

$$\delta(q_0, \varepsilon, F) = \delta(q_0, a), (q_0, c \mid \varepsilon)$$

$$\delta(q_0, a) = \delta(q_0, \varepsilon)$$

$$\delta(q_0, * *) = \delta(q_0, \varepsilon)$$

$$\delta(q_0, + +) = \delta(q_0, \varepsilon)$$

$$\delta(q_0, c, c) = \delta(q_0, \varepsilon)$$

$$\delta(q_0, \varepsilon, \varepsilon) = \delta(q_0, \varepsilon)$$

Now,

$$(q_0, a + (a * a), \epsilon) \vdash (q_0, a + (a * a), \epsilon * \tau)$$

$$\vdash (q_0, a + (a * a), \tau * \tau)$$

$$\vdash (q_0, a + (a * a), \tau * \tau)$$

$$\vdash (q_0, a + (a * a), \tau * \tau)$$

$$\vdash (q_0, a + (a * a), \tau * \tau)$$

$$\vdash (q_0, (a + 0), \tau)$$

$$\vdash (q_0, (a * a), F)$$

$$\vdash (q_0, (a * a), (F))$$

$$\vdash (q_0, (a * a), (F))$$

$$\vdash (q_0, (a * a), (T))$$

$$\vdash q_0, (a * a), (T * F)$$

$$\vdash q_0, (a * a), (F + F)$$

$$\vdash q_0, (a + a), (a * F)$$

$$\vdash q_0, (a + 0), (a + a)$$

$$\vdash (q_0 * a * a), (0, * 0)$$

$$\vdash (10, * a), (10 * a)$$

$$\vdash (q_0, a), (a) * a$$

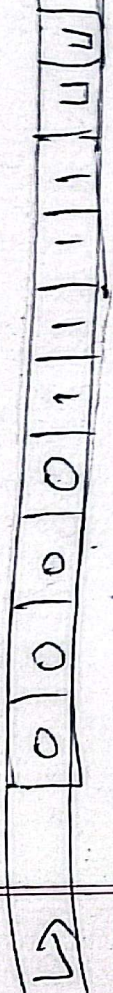
$$\vdash (q_0,) * a$$

$$\vdash (q_0, \epsilon, \epsilon)$$

Hence the string is accepted via empty stack.

Q7

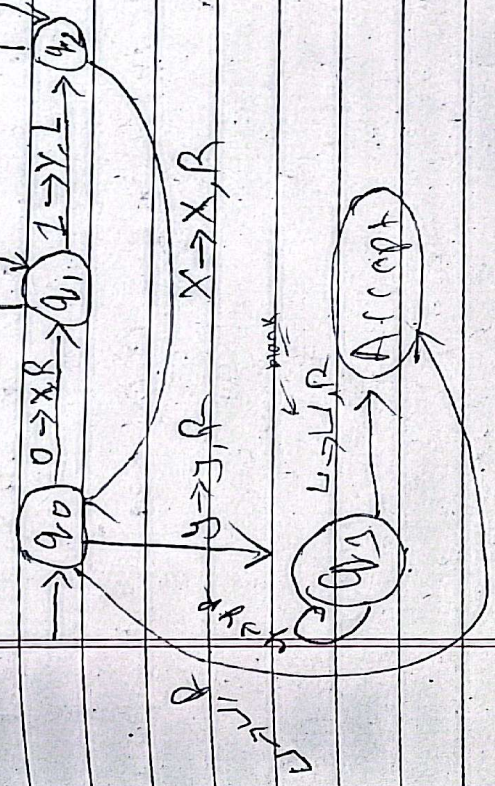
Design a Turing machine which recognizes the language $L = 0^n 1^n$



Algorithm:

- Change "0" to "x"
- Move RIGHT to first "1"
- Change "1" to "y"
- Move LEFT to leftmost "0"
- Repeat the above steps until no more "0"s
- Make sure no more "1"s remain.

$y \rightarrow y, R$
 $0 \rightarrow 0, R$
 $y \rightarrow y, L$
 $0 \rightarrow 0, L$



Q7

Design a Turing machine that

accepts the language $L = \{a^n b^n c^n / n \geq 1\}$

over the alphabet $\Sigma = \{a, b, c\}$

[2015 Spring, 2012 Spring, 2019 Fall, 38 marks]

So, eg: aaabbbccc

Approach for $a^n b^n c^n / n \geq 1$

1) Mark 'a' then move right.

2) Mark 'b' then move right

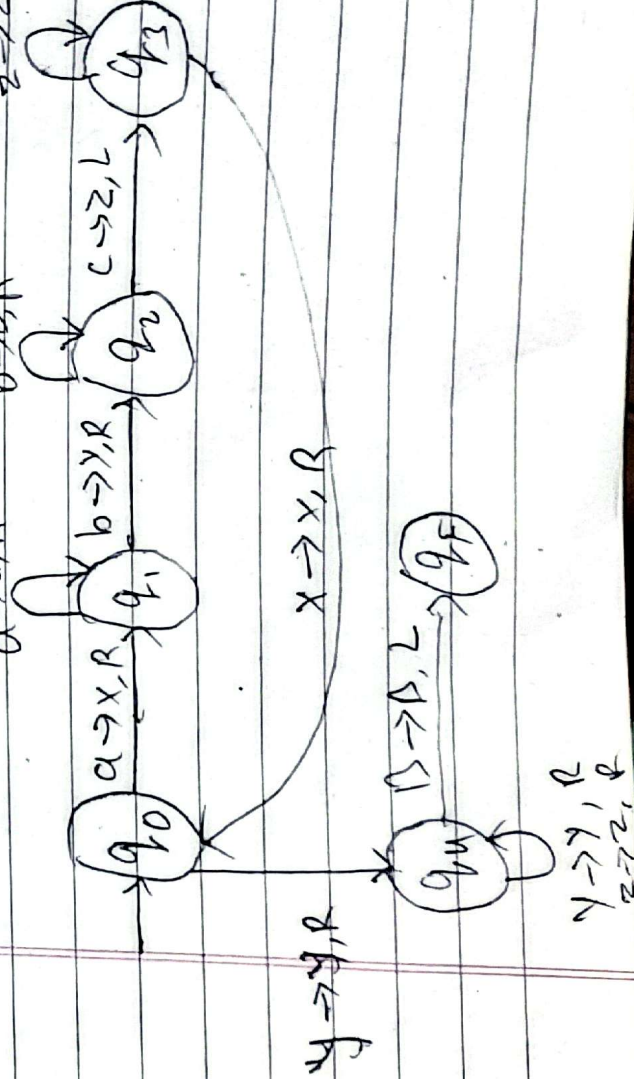
3) Mark 'c' then move left.

We come to 'a' left till we get 'x'

5) Repeat above steps till 'a', 'b' and 'c' are marked.

6) At last if everything is marked then many string is accepted.

$a \rightarrow a, L$
 $y \rightarrow x, R$
 $z \rightarrow z, R$
 $a \rightarrow a, R$
 $b \rightarrow b, R$
 $y \rightarrow y, L$
 $z \rightarrow z, L$



for C₂

$$a^n b^n c^n = a^n b^n c^n$$

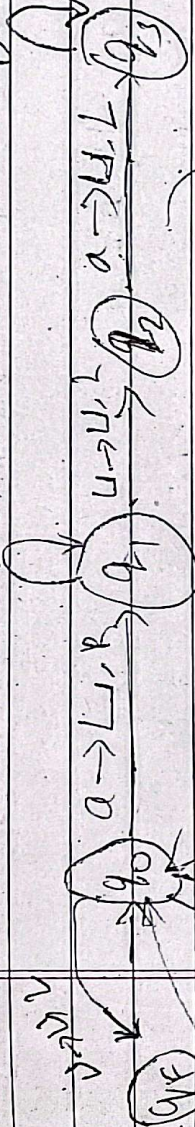

Q. 2. How

Q.1 Design a Turing Machine that accepts

Even polynomials over the alphabet.
 $\Sigma = \{a, b\}$

↳ $\Sigma \vdash a b a b a \text{ or } b a b b a b$

0-222

$$b \rightarrow b, l$$
$$a \geq 0, 2$$
$$b \rightarrow b, L$$


	0		a → g/r
--	---	--	---------

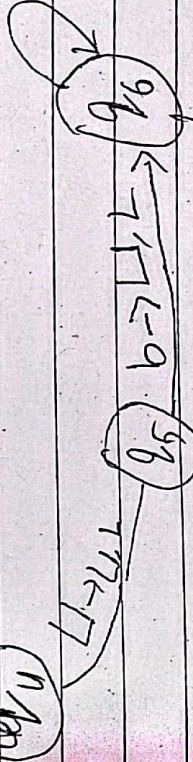
$a \rightarrow 9, 12$

$a \rightarrow 9, 12$

2. 2. 2. 2. 2.

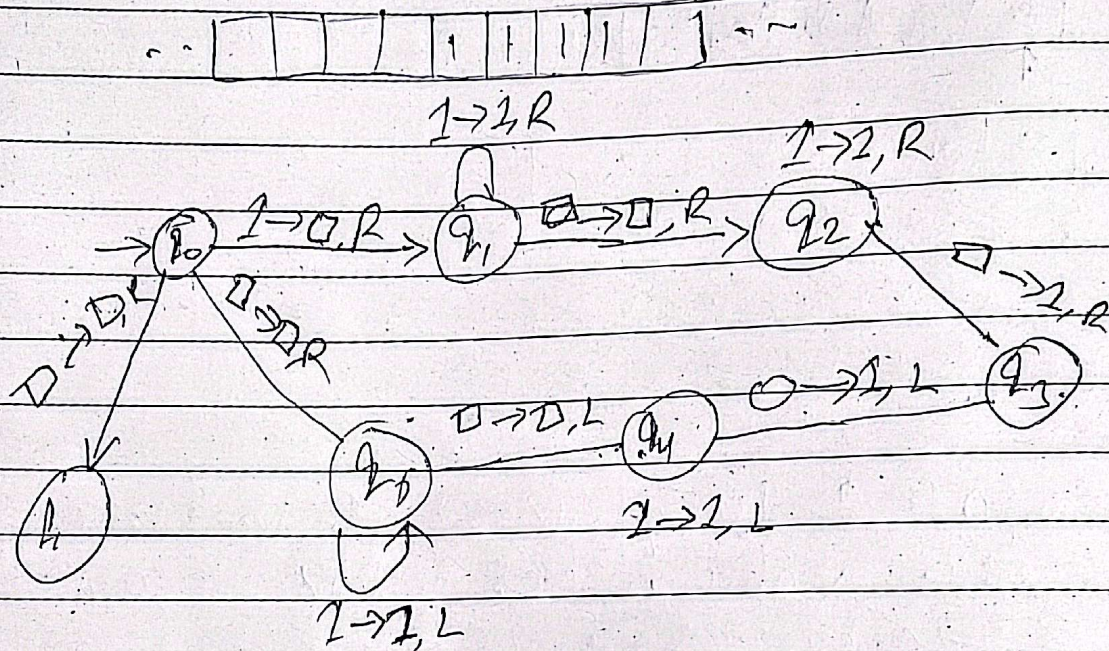
$$a \rightarrow 0, 2$$

7-26-66



UAW-R

Test input: 11



A Turing machine to decide
 $\{w \# w \mid w \in \{a, b, c\}^+\}$ [2017 Aug]

↳ ~~abbae~~ ~~ab~~

- Use a new symbol as x
- Replace each symbol into an x after it has been
a!

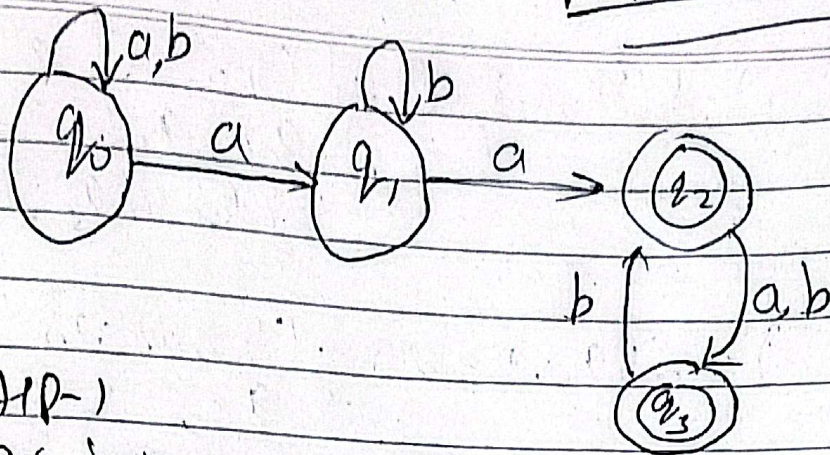
* Show that the function $f(n) = n+1$
is a Turing computable.
(2017, 2016, 2013, 2019, 2020)

7 examples.

7a

Converting NFA to DFA

Date: _____
Page: _____



STEP-1

→ For initial state q_0 , now state, not possible for DFA $\{q_0\}$ DFA.

~~Step-1~~ $\delta'(\{q_0\}, a) = \delta(q_0, a) = \{q_0, q_1\}$ new state
 $\delta'(\{q_0\}, b) = \delta(q_0, b) = q_0$

2) Step-2 for new state $\{q_0, q_1\}$

$\delta'(\{q_0, q_1\}, a) = \delta(\{q_0, q_1\}, a) = \delta(q_0, a) \cup \delta(q_1, a)$
 $= q_0 \cup q_2$
 $= \{q_0, q_2\}$ new state

$\delta'(\{q_0, q_1\}, b) = \delta(\{q_0, q_1\}, b) = \delta(q_0, b) \cup \delta(q_1, b)$
 $= q_0 \cup q_1$
 $= \{q_0, q_1\}$ ~~new state~~

3) Step-3

For new state $\{q_0, q_1, q_2\}$ & ~~$\{q_0, q_1, q_2\}$~~

$$\begin{aligned}\delta'(\{q_0, q_2\}, a) &= \delta(\{q_0, q_2\}, a) = \delta(q_0, a) \cup \delta(q_2, a) \\ &= \{q_0, q_1\} \cup \{q_3\} \\ &= \{q_0, q_1, q_3\} \text{ new state}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_0, q_2\}, b) &= \delta(\{q_0, q_2\}, b) = \delta(q_0, b) \cup \delta(q_2, b) \\ &= \{q_0\} \cup \{q_3\} \\ &= \{q_0, q_3\} \text{ new state}\end{aligned}$$

Step-4

$$\begin{aligned}\delta'(\{q_0, q_1, q_2\}, a) &= \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a) \\ &= \{q_0, q_1\} \cup \{q_2\} \cup \emptyset \\ &= \{q_0, q_1, q_2\}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_0, q_1, q_2\}, b) &= \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b) \\ &= \{q_0\} \cup \{q_3\} \cup \{q_2\} \\ &= \{q_0, q_1, q_2\}\end{aligned}$$

$10 + (0 + 11) 0^* 1$

