(23) Page No. 01 Date

NAME: PRERNA SUNIL JADHAV
SAP ID: 60004220127
CLASS: S.Y. B.Tech (Computer) (B)
COURSE:

TUTORIAL 01:

Q1 Design FSM for strings which accept set of string on $\Sigma = \{a, b\}$ starting with prefix 'ab'.

Soln

Step 1: $q_0 = \{q_s\}$
$Q = \{q_s, q_a, q_{ab}, q_u\}$
$\Sigma = \{a, b\}$
$F = \{q_{ab}\}$

Step 2: Logic

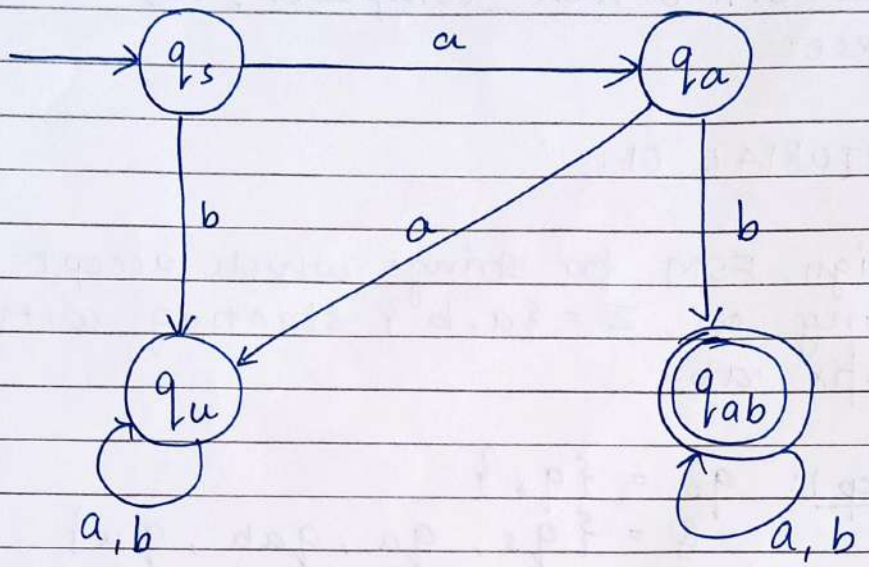
$q_s \rightarrow$ initial state
$q_a \rightarrow$ starting with a
$q_{ab} \rightarrow$ starting with ab
$q_u \rightarrow$ trap / dead state.

Step 3: Transition Table

Q \ Σ	a	b
$\rightarrow q_s$	q_a	q_u
q_a	q_u	$*q_{ab}$
$*q_{ab}$	$*q_{ab}$	$*q_{ab}$
q_u	q_u	q_u

Q \ Σ	a	b
q_s	n	n
q_a	n	y
$*q_{ab}$	y	y
q_u	n	n

Step 4: Transition diagram



Step 5:

Simulation

$\delta(q_s, abbb)$
$\delta(q_a, bbb)$
$\delta(q_{ab}, bb)$
$\delta(q_{ab}, b)$
$\delta(*q_{ab})$

⇒ Accept

$\delta(q_s, baab)$
$\delta(q_u, aab)$
$\delta(q_u, ab)$
$\delta(q_u, b)$
$\delta(q_u)$

⇒ Reject

Q2 Design a FSM for strings which accept set of all strings on $\Sigma = \{a, b\}$ with no more than 3 b's.

Soln

Step 1:

$q_0 = \{q_s\}$

$Q = \{q_s, q_b, q_{bb}, q_{bbb}\}$

$F = \{q_{bbb}\}, q_s, q_b, q_{bb}, q_{bbb}\}$

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

Step 2:

$q_s \rightarrow$ intial state or contains 0 'b's.

$q_b \rightarrow$ contains 1 'b'

$q_{bb} \rightarrow$ contains 2 'b's'.

$q_{bbb} \rightarrow$ contains 3 'b's'.

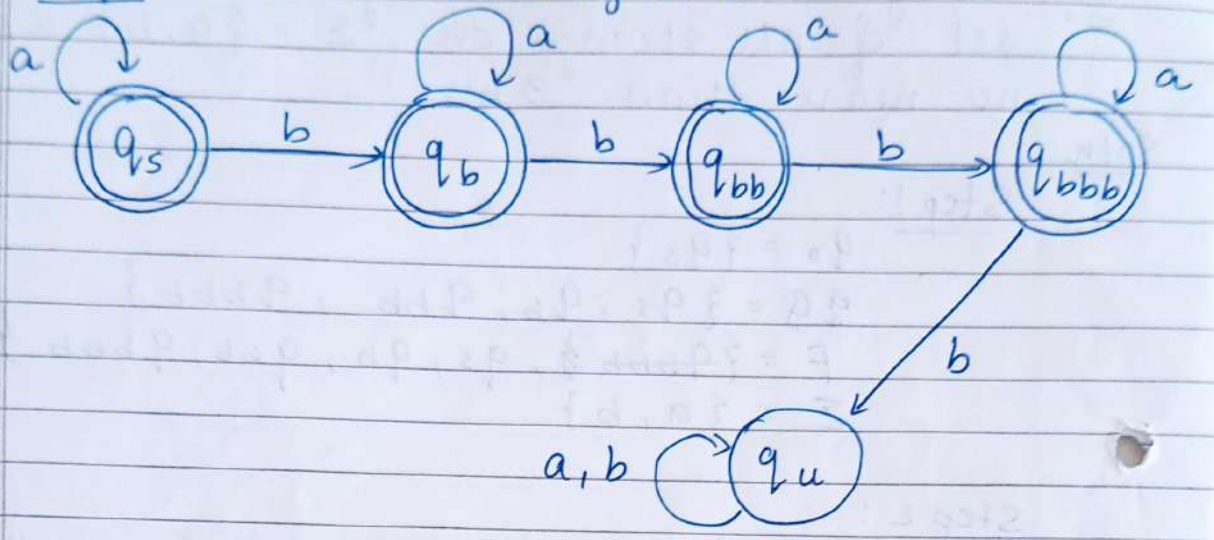
$q_u \rightarrow$ dead / trap state

Step 3: Transition Table

Q \ Σ	a	b
→ * q_s	q_s	q_b
* q_b	q_b	q_{bb}
* q_{bb}	q_{bb}	q_{bbb}
* q_{bbb}	q_{bbb}	q_u
q_u	q_u	q_u

Q \ Σ	a	b
→* q_s	Y	Y
* q_b	Y	Y
* q_{bb}	Y	Y
* q_{bbb}	Y	n
q_u	n	n

Step 4 : Transition Diagram



Step 5 : Simulation

$\delta(q_s, ababbaa)$
$\delta(q_s, babbaa)$
$\delta(q_b, abbaa)$
$\delta(q_b, bbaa)$
$\delta(q_{bb}, baa)$
$\delta(q_{bbb}, aa)$
$\delta(q_{bbb}, a)$
$\delta(*q_{bbb})$

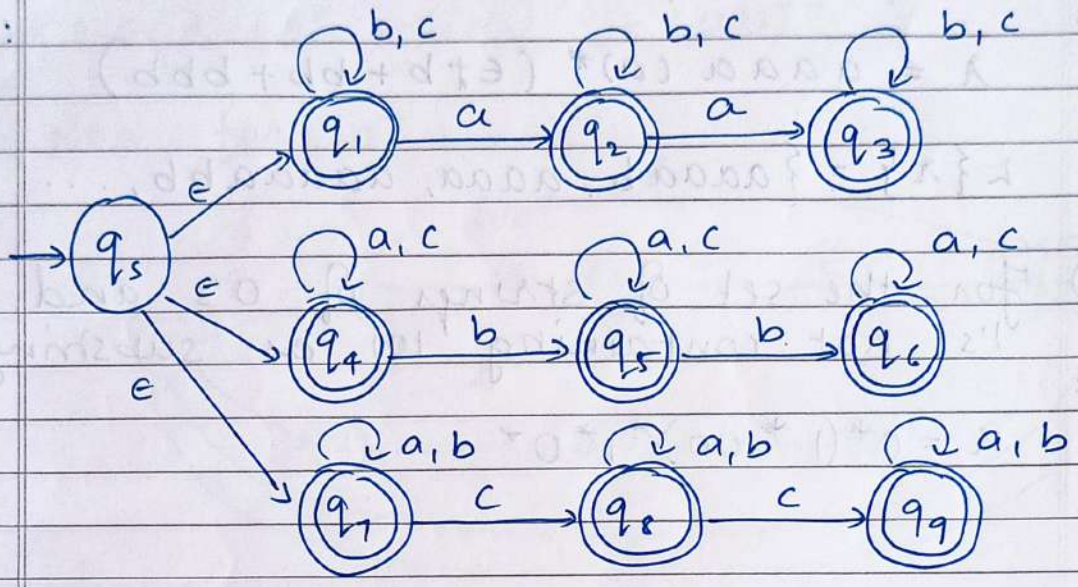
⇒ Accept

$\delta(q_s, ababbb)$
$\delta(q_s, babbb)$
$\delta(q_b, abbb)$
$\delta(q_b, bbb)$
$\delta(q_{bb}, bb)$
$\delta(q_{bbb}, b)$
$\delta(q_u)$

⇒ Reject

Q3 Design a FSM for $\Sigma = \{0, 1, 2\}$ and $L = \{w \mid$ some characters from Σ appear at most twice in $w\}$

Soln:



Transition Table:

S \ I	0 = a	1 = b	2 = c
q_5	ϵ	ϵ	ϵ
q_1	q_2	q_1	q_1
q_2	q_3	q_2	q_2
q_3	–	q_3	q_3
q_4	q_4	q_5	q_4
q_5	q_5	q_6	q_5
q_6	q_6	–	q_6
q_7	q_7	q_7	q_8
q_8	q_8	q_8	q_9
q_9	q_9	q_9	–

Q4 Write R.E. for the language

(a) $L = \{a^n b^m \mid n \geq 4 , m \leq 3\}$

Soln:

$$r = aaaa\,(a)^*\,(\epsilon + b + bb + bbb)$$

$$L\{r\} = \{aaaab, aaaa, aaaaabb, \ldots\}$$

(b) For the set of strings of 0's and 1's not containing '101' as substring

Soln:

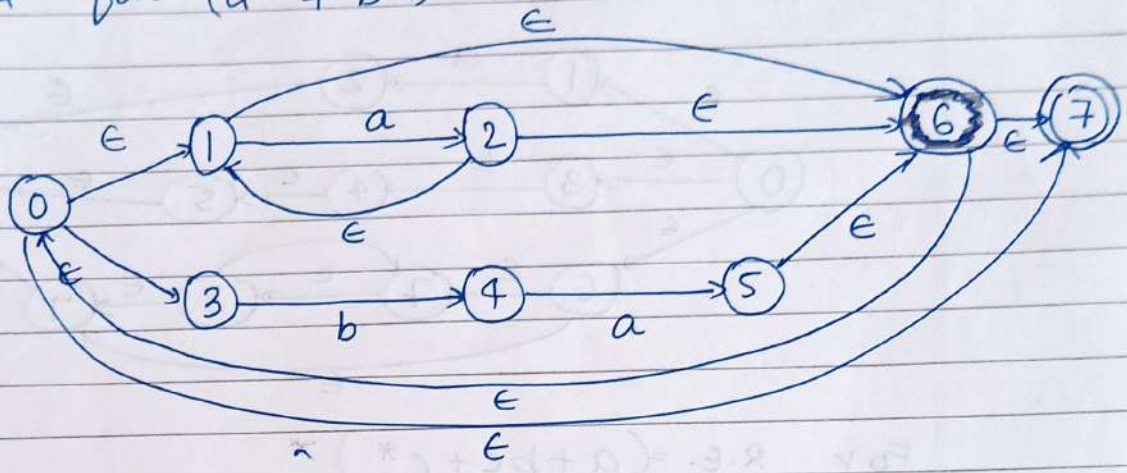
$$r = 0^*(1^*00)^*1^*0^*$$

Q5 Convert the following RE to NFA

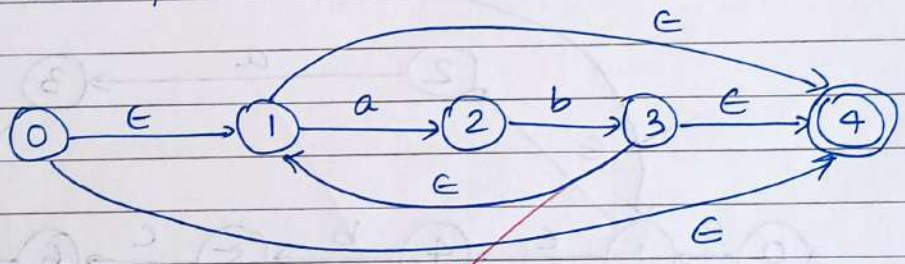
Soln:

(a) R.E. = $(a^* + ba)^* + (ab)^*$

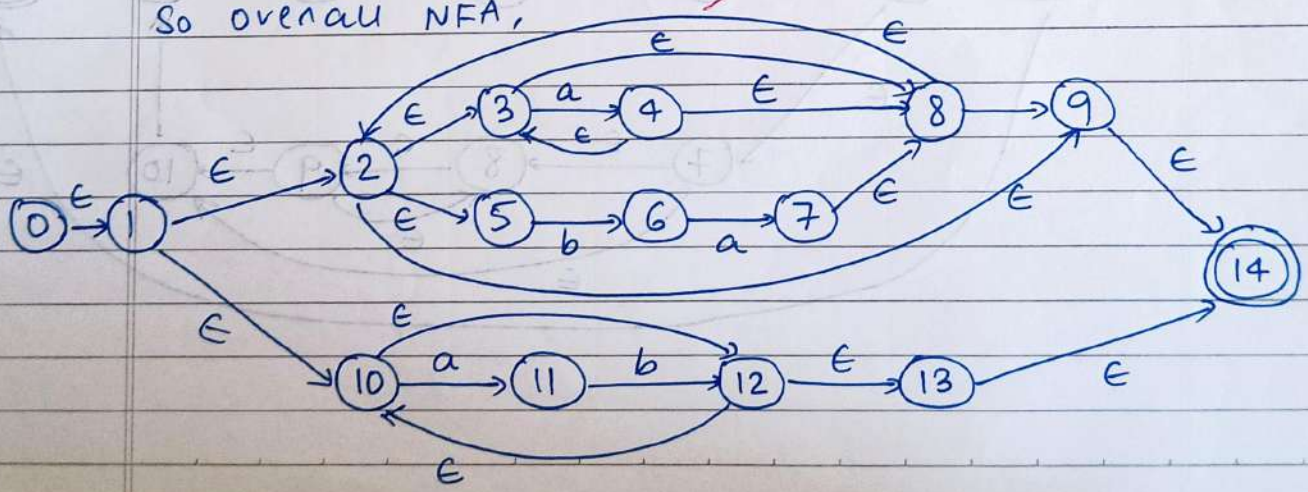
NFA for $(a^* + ba)^*$ is



Now, NFA for $(ab)^*$



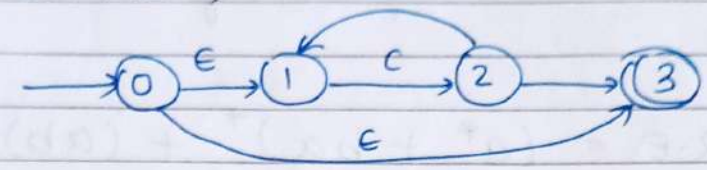
So overall NFA,



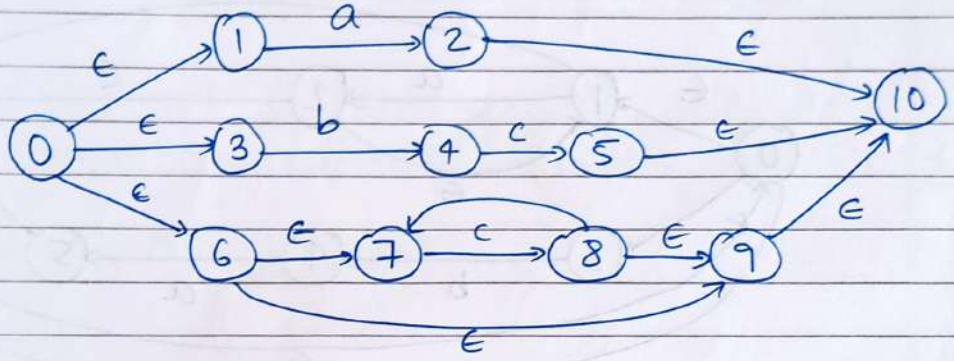
(b) R.E. = $(a + bc + c^*)^*$

Soln

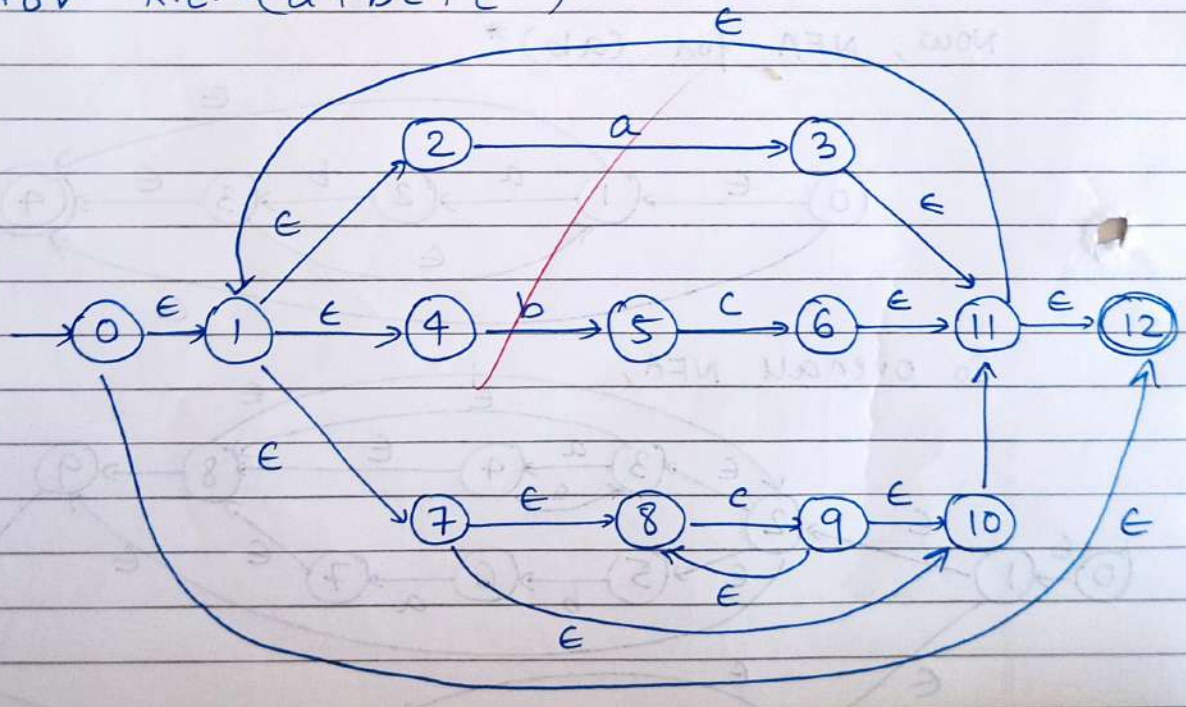
For c^*



For $a + bc + c^*$



For R.E. $= (a + bc + c^*)^*$

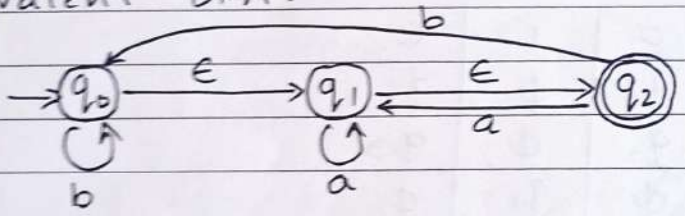


23 pages

NAME : PRERNA SUNIL JADHAV
SAP ID : 60004220127
COURSE: FORMAL LANGUAGES AND AUTOMATA THEORY
TUTORIAL (DJ19CET402)

TUTORIAL 2

Q1 Convert the following NFA with Epsilon to equivalent DFA.



Soln

x	y = ε - closure (x)	δ(y, a)	δ(y, b)
A {q_0}	{q_0, q_1, q_2}	{q_1}	{q_0}
B {q_1}	{q_1, q_2}	{q_1}	{q_0}

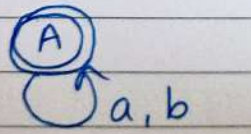
Q \ Σ	a	b
A*	B	A
B*	B	A

can be merged

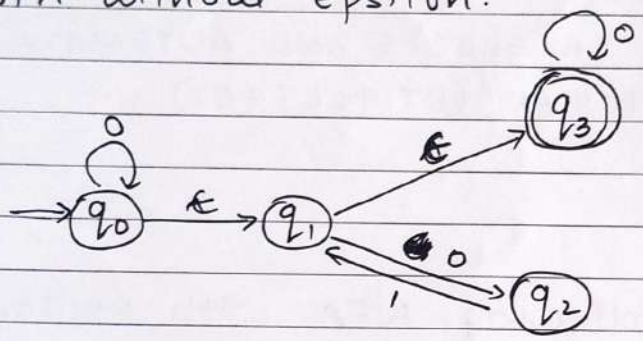
As the transition A & B is same & as both of them are final & states, we can merge them

Q \ Σ	a	b
A*	A	A

DFA transition diagram:



Q2 Convert the following NFA with Epsilon to NFA without Epsilon.



Soln

Σ / Q	0	1	ε
q_0	q_0	ϕ	q_1
q_1	q_2	ϕ	q_3
q_2	ϕ	q_1	ϕ
q_3	q_3	ϕ	ϕ

Hence, ϵ-closure$(q_0) = \{q_0, q_1, q_3\}$

ϵ-closure$(q_1) = \{q_1, q_3\}$

ϵ-closure$(q_2) = \{q_2\}$

ϵ-closure$(q_3) = \{q_3\}$

$\delta(q_0, 0) = \epsilon$-closure$(\delta(\delta^\wedge(q_0, \epsilon), 0))$

$= \epsilon$-closure$(\delta(q_0, q_1, q_3), 0))$

$= \epsilon$-closure$(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_3, 0))$

$= \epsilon$-closure$(q_0 \cup q_2 \cup q_3)$

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

$\delta(q_0, 1) = \epsilon\text{-closure}(\delta(\delta'(q_0, \epsilon), 1))$
$= \epsilon\text{-closure}(\delta(q_0, q_1, q_3), 1)$
$= \epsilon\text{-closure}(\delta(q_0, 1) \cup (q_1, 1) \cup (q_3, 1))$
$= \epsilon\text{-closure}(\phi \cup \phi \cup \phi)$
$= \{\phi\}$

$\delta(q_1, 0) = \epsilon\text{-closure}(\delta(\delta'(q_1, \epsilon), 0))$
$= \epsilon\text{-closure}(\delta(q_1, q_3), 0)$
$= \epsilon\text{-closure}(\delta(q_1, 0) \cup \delta(q_3, 0))$
$= \epsilon\text{-closure}(q_2 \cup q_3)$
$= \{q_2, q_3\}$

$\delta(q_1, 1) = \epsilon\text{-closure}(\delta(\delta'(q_1, \epsilon), 1))$
$= \epsilon\text{-closure}(\delta(q_1, q_3), 1)$
$= \epsilon\text{-closure}(\delta(q_1, 1) \cup \delta(q_3, 1))$
$= \epsilon\text{-closure}(\phi \cup \phi)$
$= \{\phi\}$

$\delta(q_2, 0) = \epsilon\text{-closure}(\delta(\delta'(q_2, \epsilon), 0))$
$= \epsilon\text{-closure}(\delta(q_2, 0))$
$= \{\phi\}$

$\delta(q_2, 1) = \epsilon\text{-closure}(\delta(\delta'(q_2, \epsilon), 1))$
$= \epsilon\text{-closure}(\delta(q_2), 1)$
$= \epsilon\text{-closure}(\delta(q_2, 1))$
$= \epsilon\text{-closure}(q_1)$
$= \{q_1, q_3\}$

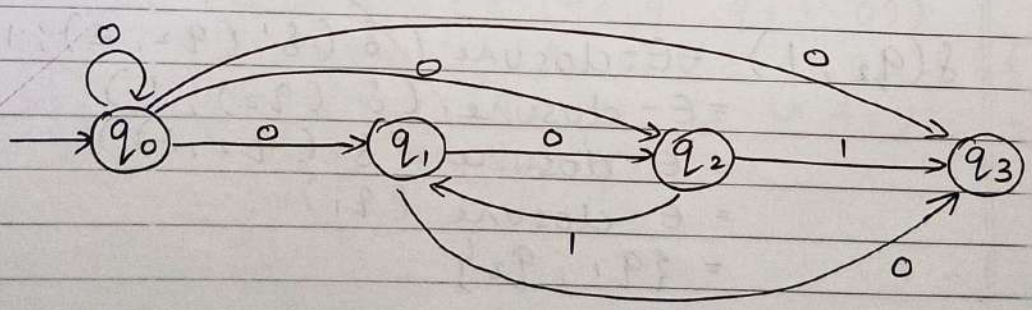
$\delta(q_3, 0) = \epsilon\text{-closure}(\delta(\delta'(q_3, \epsilon), 0))$
$= \epsilon\text{-closure}(\delta(q_3), 0)$
$= \epsilon\text{-closure}(\delta$
$= \epsilon\text{-closure}(q_3)$
$= \{q_3\}$

$\delta(q_3, 1) = \epsilon\text{-closure}(\delta'(q_3, \epsilon), 1)$
$= \epsilon\text{-closure}(\delta'(q_3), 1)$
$= \epsilon\text{-closure}(\delta'(q_3, 1))$
$= \epsilon\text{-closure}(\phi)$
$= \{\phi\}$

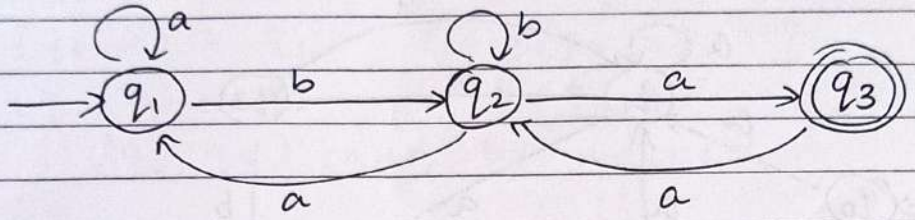
State transition function:

Q \ Σ	0	1
q_0	$\{q_0, q_1, q_2, q_3\}$	ϕ
q_1	$\{q_2, q_3\}$	ϕ
q_2	ϕ	$\{q_1, q_3\}$
q_3	$\{q_3\}$	ϕ

Transition Diagram:



Q3 Construct RE for the given NFA / DFA using Arden's Theorem



Soln The above state diagram is given as-

$q_1 = \epsilon + q_1 \cdot a + q_2 \cdot a$ ... (1)

$q_2 = q_1 \cdot b + q_2 \cdot b + q_3 \cdot a$ ... (2)

$q_3 = q_2 \cdot a$ ... (3)

Sub. eq (3) in (2)

$q_2 = q_1 \cdot b + q_2 \cdot b + q_2 a \cdot a$

$R = Q + RP$

$\therefore q_2 = q_1 b (b + aa)^*$ ... (4)

... By Arden's theorem

Sub eq (4) in (1)

$q_1 = \epsilon + q_1 \cdot a + q_1 b(b + aa)^* \cdot a$

$q_1 = \epsilon + (a + b(b + aa)^* a) q_1$

$R = Q + RP$

$\therefore q_1 = \epsilon (a + b(b + aa)^* \cdot a)^*$ ... By Arden's theorem

$\therefore q_1 = (a + b(b + aa)^* \cdot a)^*$ ... (5)

Sub (5) in (4)

$\therefore q_2 = (a + b(b + aa)^* \cdot a)^* \cdot b(b + aa)^*$ .. (6)

Sub (6) in (3)

$\therefore q_3 = (a + b(b + aa)^* \cdot a)^* \cdot b(b + aa)^* \cdot a$

for (q_2, q_3)

$\delta(q_2, a) = q_1$ $\quad$ $\delta(q_2, b) = q_2$

$\delta(q_3, a) = q_1$ $\quad$ $\delta(q_3, b) = q_4$

Since (q_2, q_4) is marked

Mark (q_2, q_4) is marked as 2

Now, check again for

(q_0, q_1)

$\delta(q_0, a) = q_1$ $\quad$ $\delta(q_0, b) = q_2$

$\delta(q_1, a) = q_1$ $\quad$ $\delta(q_1, b) = q_3$

Now (q_2, q_3) is marked

So Mark (q_0, q_1) as 3

for (q_0, q_2)

$\delta(q_0, a) = q_1$ $\quad$ $\delta(q_0, b) = q_2$

$\delta(q_2, a) = q_1$ $\quad$ $\delta(q_2, b) = q_2$

$\therefore (q_0, q_2)$ cant be marked

for (q_1, q_2)

$\delta(q_1, a) = q_1$ $\quad$ $\delta(q_1, b) = q_3$

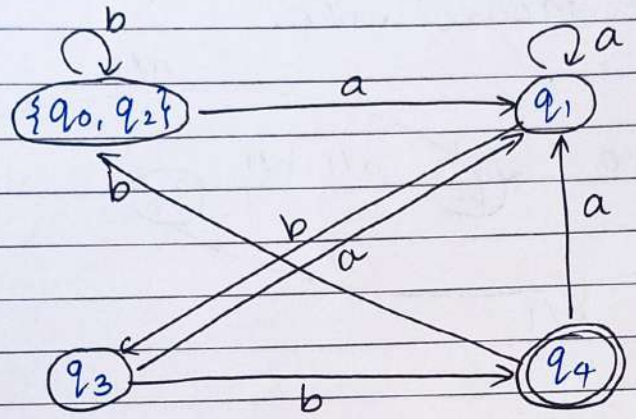
$\delta(q_2, a) = q_1$ $\quad$ $\delta(q_2, b) = q_2$

Now (q_2, q_3) is marked

Mark (q_1, q_2) as 3

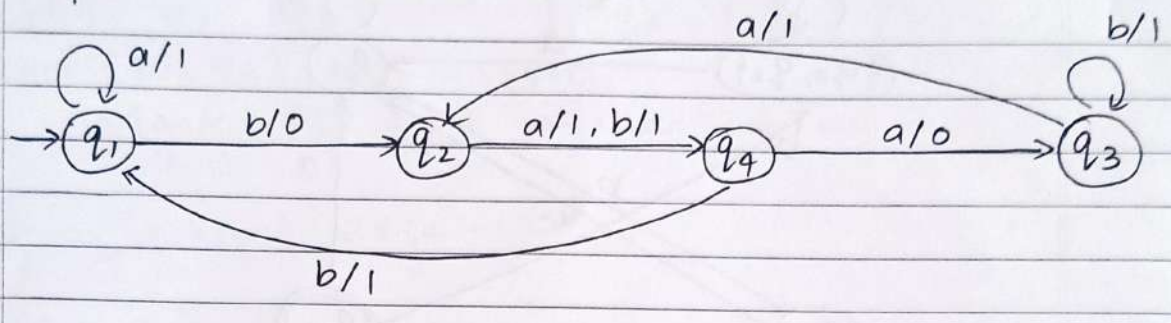
$\therefore$ Merging (q_0, q_2)

Minimized DFA is as follows:



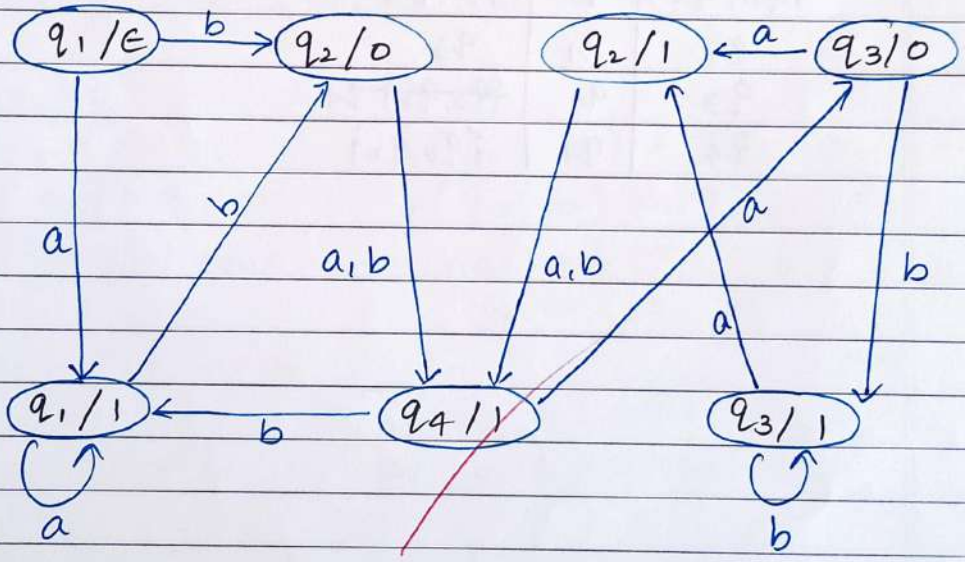
Q \ Σ	a	b
{q0, q2}	q1	{q0, q2}
q1	q1	q3
q3	q1	~~{q0, q2}~~ q4
q4	q1	{q0, q2}

Q5 Convert the following Mealy m/c into equivalent Moore m/c.



Soln

Equivalent Moore machine is as follows:



(24)

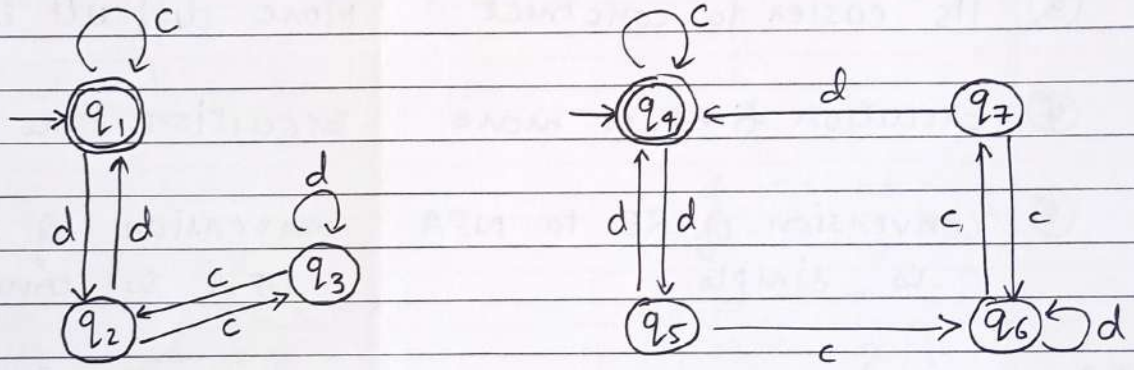
NAME: PRERNA SUNIL JADHAV

SAP ID: 60004220127

COURSE: FORMAL LANGUAGE AND AUTOMATA THEORY
TUTORIAL (DJ19CET402)

TUTORIAL 3

Q1 Consider the two Deterministic Finite Automata (DFA) and check whether they are equivalent or not.



Soln

states	c	d
$\{q_1, q_4\}$	$\{q_1, q_4\}$ FS FS	$\{q_2, q_5\}$ IS IS
$\{q_2, q_5\}$	$\{q_3, q_6\}$ IS IS	$\{q_1, q_4\}$ FS FS
$\{q_3, q_6\}$	$\{q_2, q_7\}$ IS IS	$\{q_3, q_6\}$ IS IS
$\{q_3, q_7\}$	$\{q_2, q_6\}$ IS IS	$\{q_3, q_4\}$ IS FS

Thus on comparing the transition of the states $\{q_3, q_7\}$ on input d it leads to a state $\{q_3, q_4\}$ in which q_4 is a final state & q_3 is an intermediate state. Hence, they are equivalent

Q2

	NFA	DFA
①	NFA stands for Non-Deterministic finite automata	DFA stands for Deterministic finite automata
②	It can use empty string transition	It cannot use empty string transition
③	Its easier to construct	More difficult to construct
④	Execution time is more	Execution time is less
⑤	Conversion of RE to NFA is simple	Conversion of RE to DFA is complex.

Q3

	Moore machine	Mealy machine
①	Output is associated with state	Output is associated with transition
②	If input changes output doesnt change	If input changes output also changes
③	Asynchronous output	Synchronous output/state
④	Easy to design	Difficult to design
⑤	It has often a larger no. of states compared	Can be implemented with a fewer no. of states

Q4 '+*-xyxy' → The string to be derived

Left Most Derivation is given as follows:

$E \rightarrow +EE$ .... [E → +EE]
$E \rightarrow +*EEE$ .... [E → *EE]
$E \rightarrow +*-EEEE$ .... [E → -EE]
$E \rightarrow +*-xEEE$ .... [E → x]
$E \rightarrow +*-xyEE$ .... [E → y]
$E \rightarrow +*-xyxE$ .... [E → x]
$E \rightarrow +*-xyxy$ .... [E → y]

Right Most Derivation

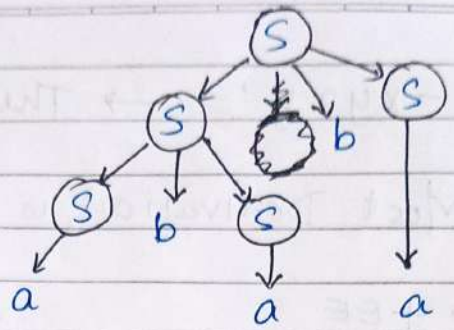
$E \rightarrow +EE$ .... [∵ E → +EE]
$E \rightarrow +Ey$ .... [∵ E → y]
$E \rightarrow +*EEy$ .... [∵ E → *EE]
$E \rightarrow +*Exy$ .... [∵ E → x]
$E \rightarrow +*-EExy$ .... [∵ E → -EE]
$E \rightarrow +*-Eyxy$ .... [∵ E → y]
$E \rightarrow +*-xyxy$ .... [∵ E → x]

Q5 a) $S \rightarrow SbS/a$

LMD

$S \rightarrow SbS$
$S \rightarrow SbSbS$ .... [∵ S → SbS]
$S \rightarrow abSbS$ .... [∵ S → a]
$S \rightarrow ababS$ .... [∵ S → a]
$S \rightarrow ababa$ .... [∵ S → a]

Left Parse Tree



ababa

RMD

$S \rightarrow SbS$

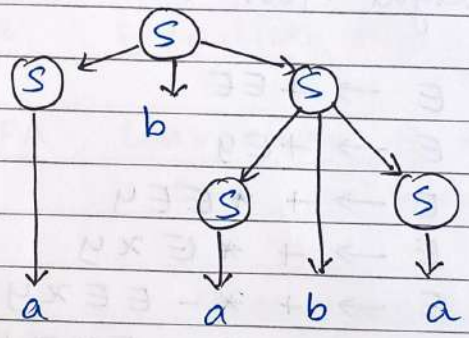
$S \rightarrow SbSbS$ .... [$\because S \rightarrow SbS$]

$S \rightarrow SbSba$ .... [$\because S \rightarrow a$]

$S \rightarrow Sbaba$ .... [$\because S \rightarrow a$]

$S \rightarrow ababa$ .... [$\because S \rightarrow a$]

Right Parse Tree



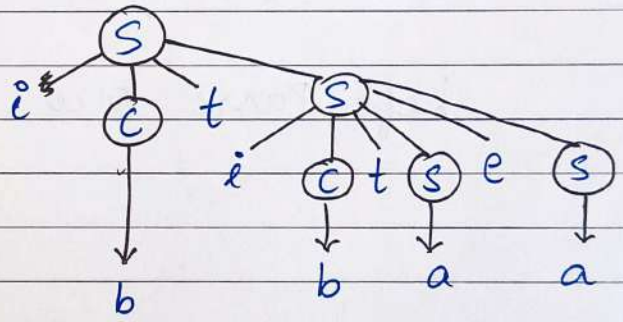
ababa

Now, Since Left & Right Parse trees are not the same for the sequence 'ababa' Thus the grammar is ambigious.

Q5
(b) $S \rightarrow iCtS \qquad C \rightarrow b$
$S \rightarrow iCtSeS$
$S \rightarrow a$

RMD

$S \rightarrow iCtS$
$S \rightarrow iCtiCtSeS$ .... [$\because S \rightarrow iCtSeS$]
$S \rightarrow iCtiCtSea$ .... [$\because S \rightarrow a$]
$S \rightarrow iCtiCtaea$ .... [$\because S \rightarrow a$]
$S \rightarrow iCtibtaea$ .... [$\because C \rightarrow b$]
$S \rightarrow ibtibtaea$ .... [$\because C \rightarrow b$]

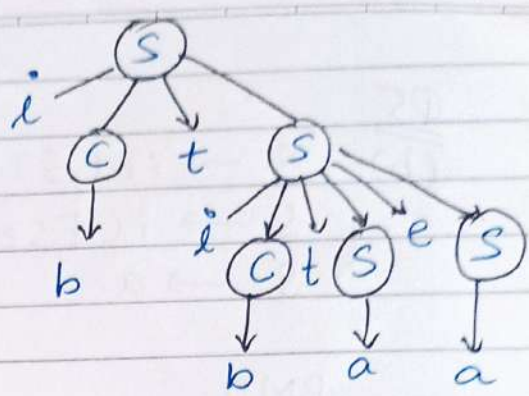


ibtibtaea

LMD

$S \rightarrow iCtS$
$S \rightarrow ibtS$ .... [$\because C \rightarrow b$]
$S \rightarrow ibtiCtSeS$ .... [$\because S \rightarrow iCtSeS$]
$S \rightarrow ibtibtSeS$ .... [$\because C \rightarrow b$]
$S \rightarrow ibtibtaeS$ .... [$\because S \rightarrow a$]
$S \rightarrow ibtibtaea$ .... [$\because S \rightarrow a$]

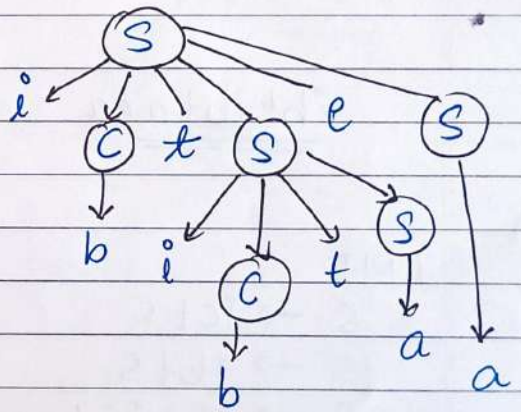
Left Parse Tree



ibtibtaea

S → ictses ... [∵ ~~S~~ c → b]
S → ibtses .... [∵ S → icts]
S → ibtictses .... [∵ ~~S~~ c → b]
S → ibtibtses .... [∵ S → a]
S → ibtibtaes .... [∵ S → a]
S → ibtibtaea

Left Parse Tree



ibtibtaea

Now, since there are two possible left parse trees for the same sequence.
∴ The given grammar is Ambigious

23

NAME: PRERNA SUNIL JADHAV
SAP ID: 60004220127
COURSE: FORMAL LANGUAGE AND AUTOMATA THEORY

TUTORIAL 4

Q1 Simplify the following grammars:

a)

$S \rightarrow aA \mid bB$
$A \rightarrow aA \mid a$
$B \rightarrow bB$
$D \rightarrow ab \mid Ea$
$E \rightarrow aC \mid d$

Soln

Productions D, E are not reachable from the starting production
∴ They are useless production and thus removed from the grammar
New grammar:
$S \rightarrow aA \mid bB$
$A \rightarrow aA \mid a$
$B \rightarrow bB$
Production B is a recursive grammar and it cannot be terminated, so remove Production B and its occurence in S.
New grammar:
$S \rightarrow aA$
$A \rightarrow aA \mid a$

b)

$S \rightarrow AB \mid AC$
$A \rightarrow aA \mid bAa \mid a$
$B \rightarrow bbA \mid aB \mid AB$
$C \rightarrow aCAa \mid aD$
$D \rightarrow aD \mid bC$

Soln

Productions C, D are recursive among themselves

∴ We remove productions C, D and their occurence

$S \rightarrow AB$
$A \rightarrow aA \mid bAa \mid a$
$B \rightarrow bbA \mid aB \mid AB$

Q2 State and explain Pumping Lemma for Regular Languages

Soln

- Pumping Lemma is used to prove that a language is not regular.
- If A is a regular language, then A has a pumping length 'P' such that any string 'S' where $|S| \geq P$ may be divided into 3 parts, $S = xyz$, such that the following conditions must be true

i) $xyz \in A$ for every $i \geq 0$
ii) $|y| > 0$
iii) $|xy| \leq p$

Example: $A = \{a^n b^n \mid n \geq 0\}$

Soln: Assume that A is regular

Pumping length = $p = 7$

$\therefore S = aaaaaaabbbbbbb$

As $|y| > 0$

$|xy| \leq p$

We will check for $xy^iz \in a$

$x = aa$

$y = aaaa$

$z = abbbbbbb$

→ Lets assume $i = 1$

$\because xy^iz = xy^1z$

$\therefore aa(aaaa)^1 abbbbbbb$

Count no. of a's and b's

$|a| = 7$

$|b| = 7$

→ assume $i = 2$

$\therefore xy^iz = xy^2z$

$\therefore \underbrace{aa}_{x} \underbrace{(aaaaaaaa)}_{y^2} \underbrace{a\ bbbbbbb}_{z}$

count no. of a's & b's

$|a| = 11$, $|b| = 7$

as $a \neq b$

Our intial assumption fails. As we had initially assumed $A = a^n b^n$ to be regular.

$A = a^n b^n$ is not regular.

Q3 Convert the following CFG to CNF

a.

$S \rightarrow aB \mid bA$
$A \rightarrow a \mid aS \mid bAA$
$B \rightarrow b \mid aS \mid aBB$

Soln

There are no useless, null & unit productions

∴ No simplification is possible

Old Production	CNF Solution
$A \rightarrow a$	$A \rightarrow a$
$B \rightarrow b$	$B \rightarrow b$
$A \rightarrow aS$	$C_1 \rightarrow a$
	$A \rightarrow C_1 S$
$B \rightarrow aS$	$B \rightarrow C_1 S$
$S \rightarrow aB$	$S \rightarrow C_1 B$
$S \rightarrow bA$	$C_2 \rightarrow b$
	$S \rightarrow C_2 A$
$A \rightarrow bAA$	$C_3 \rightarrow AA$
	$A \rightarrow C_2 C_3$
	$C_4 \rightarrow BB$
$B \rightarrow aBB$	$B \rightarrow C_1 C_4$

∴ Final Production are: $S \rightarrow C_2A \mid C_1B$

$A \rightarrow a \mid C_1 S \mid C_2 C_3$

$B \rightarrow b \mid C_1 S \mid C_1 C_4$

$C_1 \rightarrow a$

$C_2 \rightarrow b$

$C_3 \rightarrow AA$, $C_4 \rightarrow BB$

b.

$S \rightarrow AB \mid a$
$A \rightarrow aab$
$B \rightarrow Ac$

Soln

There are no useless, null or unit productions
∴ No simplification is possible

Old production	CNF solution
$S \rightarrow a$	$S \rightarrow a$
$B \rightarrow Ac$	$C_1 \rightarrow c$
	$B \rightarrow A C_1$
$A \rightarrow aab$	$C_2 \rightarrow b$
	$C_3 \rightarrow a$
	$C_4 \rightarrow C_3 C_3$
	$A \rightarrow C_4 C_2$
$S \rightarrow AB$	$S \rightarrow AB$

∴ Final productions are:

$S \rightarrow AB \mid a$
$A \rightarrow C_4 C_2$
$B \rightarrow AC_1$
$C_1 \rightarrow c$
$C_2 \rightarrow b$
$C_3 \rightarrow a$
$C_4 \rightarrow C_3 C_3$

c. $S \rightarrow aBa \mid abba$
$A \rightarrow ab \mid AA$
$B \rightarrow aB \mid a$

Soln

Since A is not reachable we remove A.

New Production:

$S \rightarrow aBa \mid abba$

$B \rightarrow aB \mid a$

Old production	CNF solution
$B \rightarrow a$	$B \rightarrow a$
$B \rightarrow aB$	$C_1 \rightarrow a$
	$B \rightarrow C_1 B$
$S \rightarrow aBa$	$C_2 \rightarrow C_1 B$
	$S \rightarrow C_2 C_1$
$S \rightarrow abba$	$C_3 \rightarrow b$
	$C_4 \rightarrow C_1 C_3$
	$C_5 \rightarrow C_3 C_1$
	$S \rightarrow C_4 C_5$

∴ Final Productions are:

$S \rightarrow C_2 C_1 \mid C_4 C_5$
$B \rightarrow a \mid C_1 B$
$C_1 \rightarrow a$
$C_2 \rightarrow C_1 B$
$C_3 \rightarrow b$
$C_4 \rightarrow C_1 C_3$
$C_5 \rightarrow C_3 C_1$

d. $S \rightarrow aACa$
$A \rightarrow B \mid a$
$B \rightarrow C \mid c$
$C \rightarrow Cc \mid \epsilon$

Soln

→ $C \rightarrow \epsilon$ is a null production

old	New
$B \rightarrow C$	$B \rightarrow \epsilon$
$S \rightarrow aACa$	$S \rightarrow aAa$
$C \rightarrow Cc$	$C \rightarrow c$

New Production:
$S \rightarrow aAa \mid aACa$
$A \rightarrow B \mid a$
$B \rightarrow \epsilon \mid C$
$C \rightarrow c$

→ $B \rightarrow \epsilon$ is a NULL production

old	New
$A \rightarrow B$	$A \rightarrow \epsilon$

New Productions:
$S \rightarrow aAa \mid aACa$
$A \rightarrow a \mid \epsilon$
$B \rightarrow C$
$C \rightarrow c$

→ $A \rightarrow \epsilon$ is a NULL production

old	New
$S \rightarrow aACa$	$S \rightarrow aCa$
$S \rightarrow aAa$	$S \rightarrow aa$

New production

$S \rightarrow aACa \mid aAa \mid aCa \mid aa$

$A \rightarrow a$

$B \rightarrow c$

$C \rightarrow c$

∵ B is not reachable we remove B

$S \rightarrow aACa \mid aAa \mid aCa \mid aa$

$A \rightarrow a$

$C \rightarrow c$

∴ We now convert to CNF

Old	New
$A \rightarrow a$	$A \rightarrow a$
$C \rightarrow c$	$C \rightarrow c$
$S \rightarrow aACa$	$C_1 \rightarrow a$
	$C_2 \rightarrow C_1 A$
	$C_3 \rightarrow C C_1$
	$S \rightarrow C_2 C_3$
$S \rightarrow aAa$	$S \rightarrow C_2 C_1$
$S \rightarrow aCa$	$S \rightarrow C_1 C_3$
$S \rightarrow aa$	$S \rightarrow C_1 C_1$

∴ New productions: $S \rightarrow C_2 C_3 \mid C_2 C_1 \mid C_1 C_3 \mid C_1 C_1$

$A \rightarrow a$

$C \rightarrow c$

$C_1 \rightarrow a$

$C_2 \rightarrow C_1 A$

$C_3 \rightarrow C C_1$

Q4 Write down the steps to convert a CFG to CNF and GNF. State and explain the principle behind CNF and GNF

soln

i. Chomskey's Normal form (CNF):
A context free grammar without useless, unit & null productions is said to be in CNF, if
a) $A \rightarrow BC$, where $A, B, C \in V$
b) $A \rightarrow a$, where $a \in T$, $A \in V$

→ Steps for converting a CFG to CNF
1) Eliminate all the NULL, UNIT & USELESS productions
2) Add to the solution, the productions, those are already in CNF
3) For the remaining non-CNF production :-
a) Replace the terminal by some variable
b) Limit the no. of variables on the right hand side of the production to 2.

ii. Greibach Normal form (GNF)
The rules for a CFG to be in GNF are as follows:
Variable → One Terminal
OR
variable → 1 Terminal followed by n no. of terminals.

i.e, $A \to a$
$A \to a\alpha$
where $\alpha \to$ any number of variable

→ Steps for converting a CFG to GNF
1) Eliminate NULL, UNIT & USELESS Productions.
2) Use any combinations of Rule & Rule2 to obtain the GNF
Rule1: Let $A \to B\alpha$ be some 'A' production and let $B \to \beta_1 \mid \beta_2 \mid \beta_n$ be some 'B' production then, we can write A production as follow:-
$A \to \beta_1 \alpha \mid \beta_2 \alpha \mid \ldots \beta\alpha$
Rule 2: Let $A \to A\alpha_1 \mid A\alpha_2 \mid \ldots A\alpha_r$ be some A productions and Let $A \to \beta_1 \mid \beta_2 \mid \ldots \beta_s$ be some remaining A productions, then introduce a new variable B such that:
$B \to \alpha_i \mid \alpha_i B \quad 1 \leq i \leq r$
$A \to \beta_i \mid \beta_i B \quad 1 \leq i \leq s$

Q5 Consider the following Grammar G. convert it into GNF

$S \to AA \mid BC$

$A \to a \mid SA$ ....(3)

$B \to a$ ....(1)

$C \to c$ ....(2)

Soln

No simplification is possible

$B \to a$ } Both are already in GNF form

$C \to c$

consider

$S \to BC$

$S \to aC$

∴ This is in GNF

Consider

$S \to AA \mid aC$

from (3), from rule 1

$S \to aA \mid SAA$

$S \to SAA \mid aA \mid aC$

By rule 2:

$B \to \alpha \mid \alpha B$

$A \to \beta \mid \beta B$

∴ $D \Longrightarrow AA \mid AAD$

$S \to aA \mid aAD \mid aC \mid aCD$

$A \to aA \mid aADA \mid aCA \mid aCDA \mid a$

D is not in GDF

(12)

We substitute A in D

$D \rightarrow aA \mid aADAA \mid aAAD \mid aADAAD \mid aCAA \mid aCAAD \mid aCDAA \mid aCDAAD \mid aA \mid aAD$

∴ Final Production:

$S \rightarrow aA \mid aAdaC \mid aCD$

$A \rightarrow aA \mid aADA \mid aCA \mid aCDA \mid a$

$B \rightarrow a$

$C \rightarrow c$

$D \rightarrow aA \mid aADAA \mid aAAD \mid aADAAD \mid aCAAD \mid aCDAA \mid aCDAAD \mid aA \mid aAD$

NAME: PRERNA JADHAV
SAP ID: 60004220127
COURSE: FLAT

TUTORIAL 5

Q1 What is left linear grammar & right linear grammar How to convert LLG to RLG & vice versa. Explain with suitable example.

→ The language accepted by a FA can be described using a set of production rules known as Regular Grammar.

→ Types.

1) Right linear grammar : All the non-terminals on the right hand side exist at the rightmost place. i.e., right ends

Eg: $A \rightarrow aB$

$S \rightarrow 00B$

2) Left linear grammar: All the non-terminals on the right hand side exists at the leftmost place i.e left ends

Eg: $A \rightarrow Ba$

$S \rightarrow B00$

→ LLG to RLG

Shift all the left handed nonterminal to the right.

Eg: LLG

$A \rightarrow Ba$

$B \rightarrow ab$

RLG

$A \rightarrow abaB$ OR

$B \rightarrow \epsilon$

$A \rightarrow abB$

$B \rightarrow a$

→ RLG to LLG

Shift all the right handed non terminals to left

Eg: RLG

$A \to aB$

$B \to ab$

LL G

$A \to Baab$ OR

$B \to \epsilon$

$A \to Bab$

$B \to a$

Q2 Differentiate b/n NPDA and DPDA

	NPDA	DPDA
1)	A pushdown automata $M = (Q, \Sigma, F, \delta, q_0, Z_0, f)$ is said to be NPDA if Q - Finite set of states Σ = Input language δ = Transition mapping $q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to Q \times \Gamma^*$ q_0 = Initial state $q_0 \in Q$ Z = Stack start symbol $Z \in \Gamma$ F = Finite set of final states $F \subseteq Q$	A pushdown automata $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, f)$ is said to DPDA if Q = finite set of states Σ = Input alphabet Γ = Stack alphabet δ = Transition function $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to$ finite subset of $Q \times \Gamma^*$ q_0 = Initial state $q_0 \in Q$ Z = Stack start symbol $Z \in \Gamma$ F = finite set of final state $F \subseteq Q$
2)	No restriction, for every $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$ & $b \in \Gamma$ $\delta(q, a, b)$ can have more than one element	Subject to restriction that for every $q \in Q$ $a \in \Sigma \cup \{\epsilon\}$ $b \in \Gamma(a)$ $\delta(q, a, b)$ contains at most one element

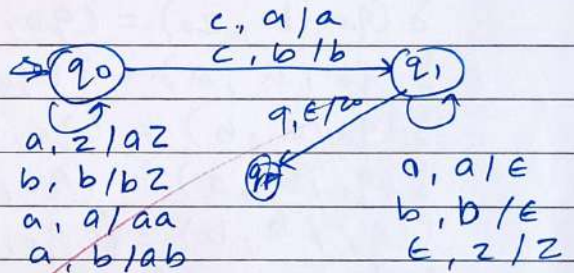
Q3 Design a PDA to accept the string
$L = \{ w c w^r \mid w \in \{a, b\}^* \}$

→ Logic:
For every a (before c) push 1 'a'
For every b (before c) push 1 'b'
For c, just bypass it & change the state
Now after c, pop 1 'a' when encounter 1 'a'
& pop 1 'b' when encounter 1 'b'

→ Implementation:

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, a, b) = (q_0, ab)$
$\delta(q_0, b, a) = (q_0, ba)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, c, Z_0) = (q_F, Z_0)$
$\delta(q_0, c, a) = (q_1, a)$
$\delta(q_0, c, b) = (q_1, b)$
$\delta(q_1, a, a) = (q_1, \epsilon)$
$\delta(q_1, b, b) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, Z_0) = (q_1, Z_0)$

→ Diagram



$Q = \{q_0, q_1, q_F\}$
$\Sigma = \{a, b, c\}$
$\Gamma = \{a, b\}$
$q_0 = q_0$
$Z_0 = Z_0$
$F = \{q_F\}$

Simulation

$\delta(q_0, abcba, Z_0)$
$\vdash \delta(q_0, bcba, aZ_0)$
$\delta(q_0, cba, baZ_0)$
$\delta(q_1, ba, baZ_0)$
$\delta(q_1, a, aZ_0)$
$\delta(q_F, \epsilon, Z_0)$
Accepted.

Q9 Design a PDA to accept the string

$L = \{ww^R \mid w \in \{a,b\}^*\}$

→ This is the case of NPDA

→ If there is an occurrence of double letter (aa, bb) there are 2 possibilities

1) The middle of the string is reached so we start popping respective symbols from stack

2) The middle of the string is not reached so we continue pushing the symbol on stack

→ Implementation

$\delta(q_0, \epsilon, Z_0) = \{(q_F, \epsilon)\}$

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$

$\delta(q_0, b, Z_0) = (q_0, bZ_0)$

$\delta(q_0, a, a) = \{(q_0, aa), (q_1, \epsilon)\}$

$\delta(q_0, a, b) = (q_0, ab)$

$\delta(q_0, b, a) = (q_0, ba)$

$\delta(q_0, b, b) = \{(q_0, bb), (q_1, \epsilon)\}$

$\delta(q_1, a, a) = (q_1, \epsilon)$

$\delta(q_1, b, b) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, Z_0) = (q_F, Z_0)$

~~Simul~~ $M =$

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

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

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

$q_0 = q_0$

$Z_0 = Z_0$

$F = \{q_F\}$

Simulation

$\delta(q_0, abaaba, Z_0)$
$\vdash \delta(q_0, baaba, Z_0)$
$\vdash \delta(q_0, aaba, bZ_0)$
$\vdash \delta(q_0, aba, abaZ_0)$

After this 2 transition occurs

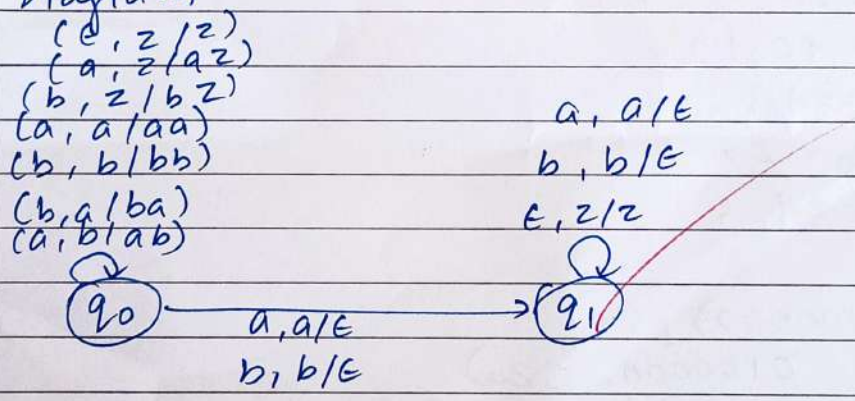
[Not middle]

$\vdash \delta(q_0, ba, aabaZ_0)$
$\vdash \delta(q_0, a, baabaZ_0)$
$\vdash \delta(q_0, \epsilon, abaabaZ_0)$
Rejected

[middle]

$\delta(q_1, ba, baZ_0)$
$\delta(q_1, a, aZ_0)$
$\delta(q_E, \epsilon, Z_0)$
Accepted.

→ Diagram



Q5 PDA for $S \to 0BB$, $B \to 0S \mid 1S \mid 0$
Check if 010^7 is accepted or not

→ Grammar is already in GNF

$M = \{Q, \Sigma, \delta, \Gamma, q_0, Z_0, F\}$

δ:

$\delta(q_0, \epsilon, Z_0) = \{(q_1, SZ_0)\}$
$\delta(q_1, 0, S) = \{(q_1, BB)\}$
$\delta(q_1, 1, B) = \{(q_1, 1S)\}$
$\delta(q_1, 0, B) = \{(q_1, S), (q_1, \epsilon)\}$
$\delta(q_1, \epsilon, Z_0) = \{(q, Z_0)\}$

$M = Q = \{q_0, q_1, q_f\}$
$\Gamma = \{Z_0, S, B\}$
$\Sigma = \{0, 1\}$
$q_0 = \{q_0\}$
$Z_0 = Z_0$
$F = \{q_f\}$

$\delta(q_0, 01000000, Z_0)$
$\vdash \delta(q_1, 0100000, SZ_0)$
$\vdash \delta(q_1, 100000, BBZ_0)$
$\vdash \delta(q_1, 00000, SBBZ_0)$
$\vdash \delta(q_1, 000, BBBZ_0)$
$\vdash \delta(q_1, 00, BBZ_0)$
$\vdash \delta(q_1, 0, SBZ_0)$
$\vdash \delta(q_1, \epsilon, Z_0)$
$\vdash \delta(q_f, \epsilon, Z_0)$

NAME: PRERNA SUNIL JADHAV
SAP ID: 60009220127
COURSE: FLAT

(20)

TUTORIAL 6

Q1 How is Turing m/c different from a PDA? Explain with example.

→ PDA is not Deterministic where as Turing m/c is deterministic in nature.

→ It can access only the top of the stack as it works on the LIFO concept where as the turing m/c can access any position on the infinite tape.

→ PDA is used for designing the parsing phase of a compiler in syntax Analysis. Turing m/c is used for implementation in neural network.

→ ~~Depending~~ for example, tower of Hanoi problem is solved using PDA & Turing has its application in AI and robotics.

→ All applications in which is stack is involved are done using PDA.
where as Turing is very helpful in understanding complexity theory.

Q2 Design a Turing m/c to calculate difference of 2 unary

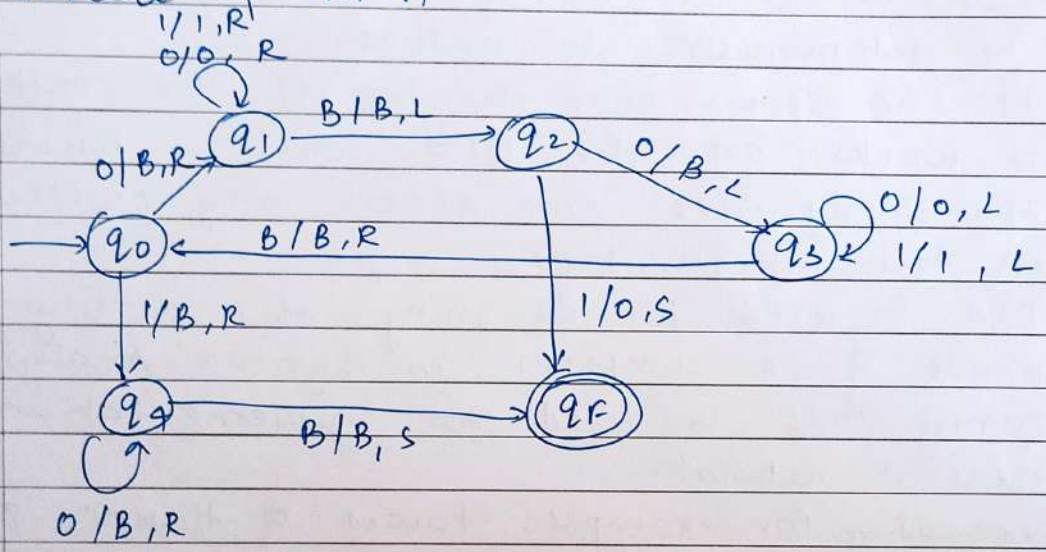
→ Logic: $m-n$ if $(m>n)$

0 if $(m<n)$ or $(m=n)$

if i/p → $B\,0^m\,|\,0^n\,B$

o/p → $B\,0^p\,B$

where $p = m-n$



Q \ Γ	0	1	B
q_0	$(q_1, 01B, R)$	$(q_4, 11B, R)$	
q_1	$(q_1, 010, R)$	$(q_1, 111, R)$	(q_2, BIB, L)
q_2	$(q_3, 01B, L)$	$(q_F, 110, S)$	
q_3	$(q_3, 010, L)$	$(q_3, 111, L)$	(q_0, BIB, R)
q_4	$(q_4, 01B, R)$		(q_F, BIB, S)
q_F			

Simulation

$m = 5 \quad n = 2$

(B00000100) → (BB00010BB)

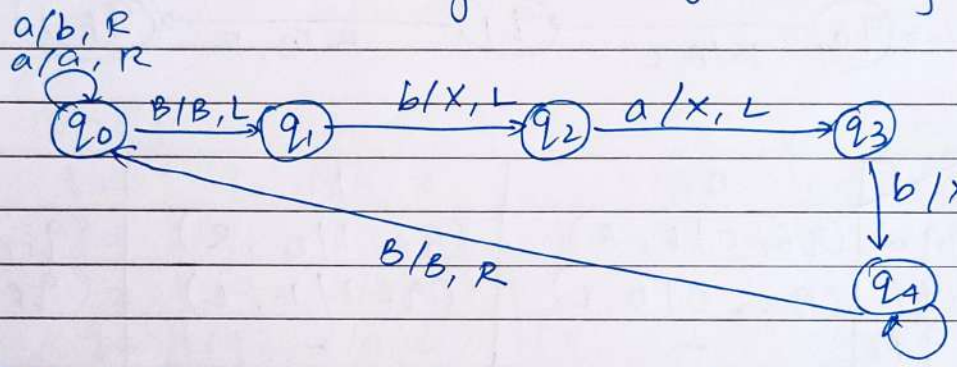
(B00001 0BB) → (BB0001BBB)

(BB0001BB) → (BB001B)

(BB001B) → (B000B)

Q3 Design a TM to accept string ending with 'bab' over $\Sigma = \{a, b\}$

→ Logic: Go to the end until blank is encountered go to left checking 'bab'.



Q \ F	a	b	X	B
q_0	$(q_0, a/a, R$	$(q_0, b/b, R$	$(q_0, X/X, R$	$(q_1, B/C, L)$
q_1		$(q_2, b/b, L$		
q_2	$(q_3, a/X, L)$			
q_3		$(q_4, b/X, L$		
q_4	$(q_4, a/a, L$	$(q_4, b/b, L$		$(q_0, B/B, R)$

Simulation:

$\delta(BababB) \rightarrow \delta(BabaXB)$

$\delta(BabaXB) \rightarrow \delta(BabXXB)$

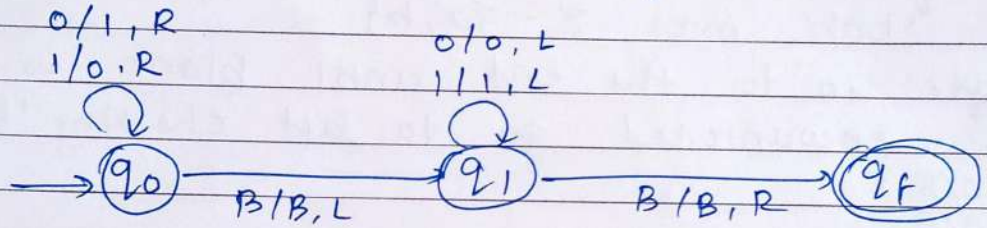
$\delta(BabXXB) \rightarrow \delta(BaXXXB)$

$\delta(BaXXXB) \rightarrow \delta(BBXXXB)$

Q4 Design a TM for 1's complement of a number over $\Sigma = \{0,1\}$

→

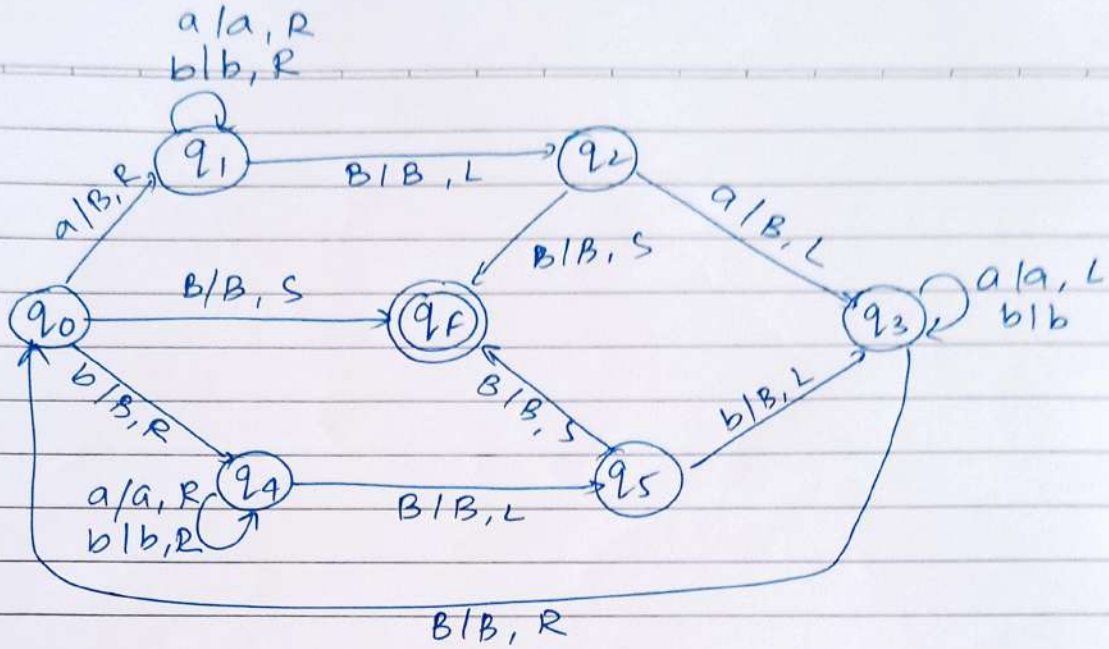
Logic:
Convert all 0's to 1's and all 1's to 0's and go right if B is encountered go left. Ignore 0's & 1's & go left till B & move right



Q \ F	0	1	B
q_0	$(q_0, 0/1, R)$	$(q_0, 1/0, R)$	$(q_1, B/B, L)$
q_1	$(q_1, 0/0, L)$	$(q_1, 1/1, L)$	$(q_F, B/B, S)$
q_F	-	-	-

Q5 Design a TM to accept string of the form WCW^R WaW^R over $\Sigma = \{a, b\}$

5



Q \ F	a	b	B
q_0	$(q_0, a/B, R)$	$(q_4, b/B, R)$	$(q_1, B/B, S)$
q_1	$(q_1, a/a, R)$	$(q_1, b/b, R)$	$(q_2, B/B, L)$
q_2	$(q_3, a/B, L)$		$(q_R, B/B, L)$
q_3	$(q_3, a/a, L)$	$(q_3, b/b, L)$	$(q_0, B/B, R)$
q_4	$(q_4, a/a, R)$	$(q_4, b/b, R)$	

Prerna Jadhav
60009220127 DATE:

TUTORIAL 7

Q1 State and explain the Halting problem

→ The Halting problem is about figuring out whether or not a given algorithm will halt (or stop) at some point in time.

→ When an algorithm is run for a given i/p, it will either halt after some time and return the o/p or it will keep on running infinitely.

→ The Halting problem helps us differentiate those 2 scenarios.

→ We cannot design a generalized algorithm that can accurately predict whether or not a given program will even halt.

→ The only way to find out is to run the algorithm & see if it halts.

Q2 What are universal Turing m/c?

→ Turing was inspired by the idea of connecting multiple Turing m/c.
He named this m/c as universal turing m/c

→ A UTM can imitate the behaviour of an arbitary Turing m/c over any collection of input symbol.

→ The i/p includes: Description of a machine on the tape
and
input data.

DATE:

→ The UTM can the simulate M on the rest of the i/p tape's content. As a result, a UTM can simulate any other m/c.

Q3 Whats are Recursive & Recursively Enumerable languages? Explain

→ Recursive Enumerable (RE) or Type 0:

- They are generated by type-0 grammar
- An RE can accepted or recognized by Turing m/c which means it will enter into final state for the strings of languages & may or may not enter into rejecting state for the strings which are not part of the language.
- It means TM can loop forever for the strings which are not a part of the language. RE language are also called as Turing recognizable languages.

→ A recursive language (subset of RE): It will enter into final state for the strings of language & rejecting state for the strings which are not part of the language eg: $L = \{a^n b^n c^n \mid n \geq 1\}$ is recursive because we can construct a turing m/c which will move to final state if the string is of the form $a^n b^n c^n$ else move to non-final state.

DATE:

Q4 Explain the variants of Turing m/c.

→ 1. multiple tape Turing m/c:
A TM that has more than one. Each tape has its own read & write head - used for more complex computation.

2. Non-deterministic TM: It can make multiple choices to the same pt during computation.

3. Two-way infinite tape: Its allowed to be infinite in both the directions

4. Multiple head turing m/c: can have more than one head scanning various cells of the tape

Q5 Explain church's Hypothesis.

→ Also known as church's thesis is a fundamental principle in the theory of computation.

→ So it states that any function that is effectively computable can be computed by TM

→ The hypothesis has been widely accepted in the field of computer science as it provides a formal definition of the concept of algorithm computation and a means of comparing the computation problem of different mode of computation.

DATE:

→ It also provides a foundation for the development of computation complexity theorem with the study of efficiency.

→ Any algorithm that can be solved by an effective method such as algorithm on a computer program can be solved by turing m/c.

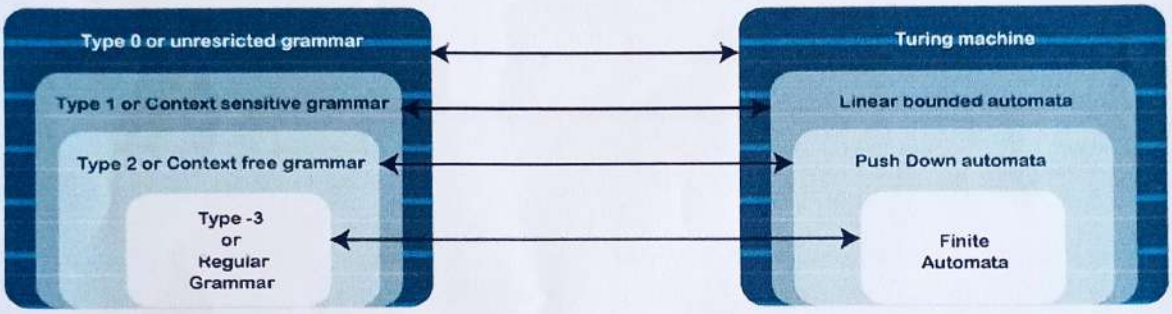
Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Academic Year: 2022-2023

Name	Prerna Sunil Jadhav
Sap Id	60004220127
Class	S. Y. B.Tech (Computer Engineering)
Course	Formal languages and Automata Theory Tutorial
Course Code	DJ19CET402
Tutorial No.	08

25 /proper

AIM. SIMULATE WORKING OF FA, PDA, AND TURING MACHINE WITH AN PROBLEM



SR. NO.	FINITE AUTOMATA	PUSH DOWN AUTOMATA	TURING MACHINE
1.	FA accepts the input if the machine end up in a final state.	PDA accepts the input if the machine end up in a final state with an empty stack.	A string is accepted by final state if the computation halts in a final state, but the TM need not read the entire input string to accept the string.
2.	Both deterministic (DFA) and nondeterministic (NFA) - Every state of DFA always has exactly one exiting transition arrow for each symbol in the alphabet while the NFA may has more. - In DFA, labels on transition arrows are from the alphabet while NFA can have an arrow with the label ε	Non-deterministic only - PDAs are non-deterministic. Allowed non-deterministic transitions – multiple transitions on same pop/input, transitions may but do not have to push or pop.	Deterministic only - Turing machine are deterministic. No lambda transitions allowed.

1. FINITE AUTOMATA

PROBLEM: Design an FSM to accept strings having even 0's and odd 1's

THEORY:

M = (Q, Σ, δ, q_0, F)

Where,

Q : Finite set called states { q_0, q_1, q_2, q_3, q_4 }

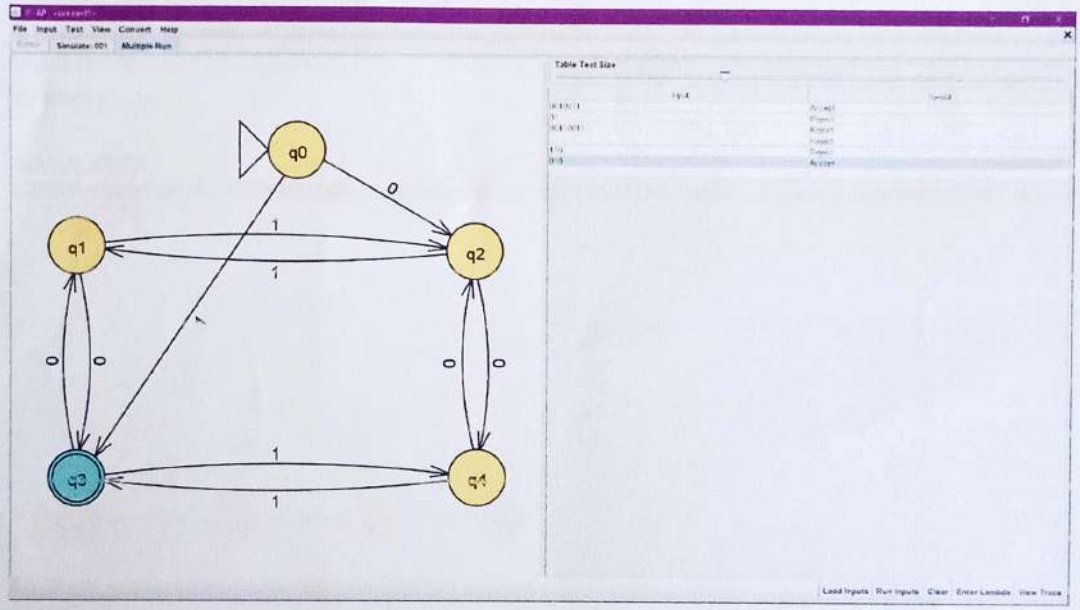
Σ : Finite set called alphabets.{ a, b }

δ : Q × Σ → Q is the transition function.

q_0 **∈ Q** is the initial state.

F : Final State - q_3

SIMULATION:



2. PUSH DOWN AUTOMATA

PROBLEM: Design a PDA to accept the following:

$L=\{ a^n b^m a^n \mid m, n >= 1 \}$

THEORY:

M = {Q, Σ, δ, F, q_0 , Z_0 }

A Pushdown Automata (PDA) can be defined as :

Q : Set of states { q_0 , q_1 , q_2 , q_3 }

Σ : Set of input symbols { a, b}

Γ : Set of pushdown symbols (which can be pushed and popped from stack)

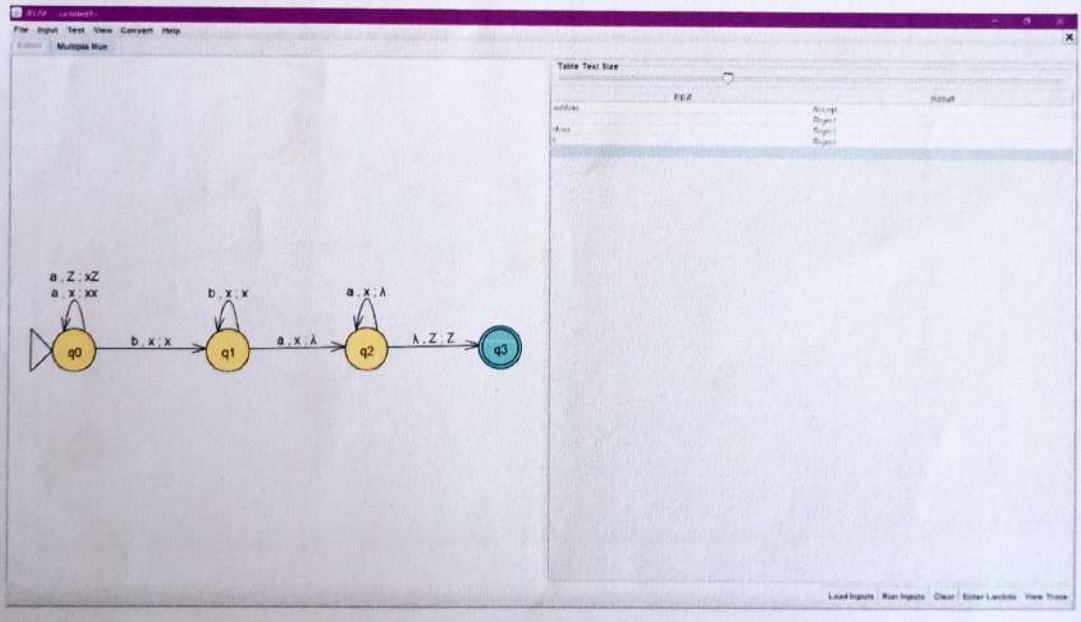
q_0 : Initial state

Z_0 : Initial pushdown symbol (which is initially present in stack)

F : Set of final states - q_3

δ : Transition function which maps **Q x {Σ ∪ ∈} x Γ** into **Q x Γ***. In a given state, PDA will read input symbol and stack symbol (top of the stack) and move to a new state and change the symbol of stack.

SIMULATION:



3. TURING MACHINE

PROBLEM: Design a TM to make a copy of a string over { 0, 1 }.

Input : ...BB1100#BB...

Output : ...BB1100#1100BB...

THEORY:

M = { Q, X, ∑, δ, q_0, B, F }

A Turing Machine can be defined as :

Q : Finite set of states. { q_0 , q_1 , q_2 , q_3 , q_4 }

X : Tape alphabet

∑ : Input alphabet

δ : Transition function where **δ : Q × X → Q × X × { Left_shift , Right_shift }**

q_0 : Initial state

B : Blank symbol

F : Set of Final states - q_4

SIMULATION: