Mealy and Moore Models T.O.C

The finite automata have binary output i.e o/p is 1 if input string is accepted, otherwise o/p is 0. Mealy and moore models are the one in which output can be chosen from some other alphabet.

A Moore machine is a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

1. Q is finite set of states
2. Σ is input alphabet
3. Δ is output alphabet mapping
4. δ is transition function $\Sigma \times Q$ into Q.
5. λ is output function mapping Q into Δ
6. q_0 is initial state.

A Mealy machine is 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where all symbols are same as Moore except λ which maps $Q \times \Sigma$ into Δ.

Note:
1. The difference between moore and mealy m/c is that in moore m/c, o/p ~~depends~~ depends only on states, but in mealy m/c, o/p depends both on states and input alphabet.
2. A finite automaton can be converted to a Moore machine by introducing $\Delta = \{0, 1\}$ and defining $\lambda(q) = 1$ if $q \in F$ and $\lambda(q) = 0$ if $q \notin F$.

Scanned by CamScanner

$Q_1 = \{q_3, q_4\}$ $Q_2 = \{q_0, q_1, q_2, q_5, q_6, q_7\}$

$\pi_0 = \{q_3, q_4\}$ $\{q_0, q_1, q_2, q_5, q_6, q_7\}$
$(5,6)$ $(7,6)$ $(1,2)$ $(4,3)$ $(4,3)$ $(3,6)$ $(6,6)$ $(4,6)$

$\pi_1 = \{q_3, q_4\}$ $\{q_0, q_6\}$ $\{q_1, q_2\}$ $\{q_5, q_7\}$
$(5,6)$ $(7,6)$ $(1,2)$ $(6,6)$ $(4,3)$ $(4,3)$ $(3,6)$ $(4,6)$

$\pi_2 = \{q_3, q_4\}$ $\{q_0\}$ $\{q_6\}$ $\{q_1, q_2\}$ $\{q_5, q_7\}$

$\pi_3 = \{q_3, q_4\}$ $\{q_0\}$ $\{q_6\}$ $\{q_1, q_2\}$ $\{q_5, q_7\}$

As $\pi_2 = \pi_3$, we stop.

The minimum state automaton M' is :-

$M' = (Q', \{a, b\}, \delta', \{q_0\}, \{q_3, q_4\})$

Q' are all subsets of π_3 and

δ

State/Σ	a	b
$[q_0]$	$[q_1, q_2]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_3, q_4]$	$[q_3, q_4]$
$[q_3, q_4]$	$[q_5, q_7]$	$[q_6]$
$[q_5, q_7]$	$[q_3, q_4]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_6]$

Scanned by CamScanner

(14)

Moore machine

Present state	Next state δ		Output
	q=0	q=1	λ
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

For input string 0111, Transition of states is

$q_0 \xrightarrow{0} q_3 \xrightarrow{1} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2$

o/p is: 0 0 0 1 0

For input string ∧(empty string), o/p is $\lambda(q_0)$. (here $\lambda(q_0) = 0$)

Mealy machine

Present state	Next state			
	q=0		q=1	
	state	o/p	state	output
→ q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

For input string 0011, Transition of states is

$q_1 \xrightarrow{0} q_3 \xrightarrow{0} q_2 \xrightarrow{1} q_4 \xrightarrow{1} q_3$

o/p is 0100

Scanned by CamScanner

Note: (1) In Mealy machine, we get an output of 1 when input is 1. But in Moore, we get output $\lambda(q_0)$ for input 1.

(2) For Moore machine, if input string is of length n, the output string is of length n+1. But in mealy machine if input string is of length n, output string is also of length n.

Procedure for transforming a mealy into a Moore machine

Q: Consider the Mealy machine described by the following transition table. Construct a Moore machine which is equivalent to the Mealy machine.

Mealy Machine

Present state	Next state			
	Input a=0		Input a=1	
	state	output	state	output
→ q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

Steps

(1) We look into next state column for all states, say q_i and determine the no. of different outputs associated with q_i.

Scanned by CamScanner

output 1 is associated with q_1
" 0+1 are " " q_2
" 0 is " " q_3
" 0+1 are " " q_4

(2) If q_i is associated with more than one output say n, then we split q_i into n different states.

Eg. we split q_2 into q_{20} + q_{21}
and we split q_4 into q_{40} and q_{41}

(3) The pair of states and outputs in the next state are rearranged as:-

Moore machine

Present state	Next state a=0	Next state a=1	Output
→ q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Here we observe that initial state q_1 is associated with output 1. This means that with input 1, we get an output of 1 if m/c starts at state q_1. Thus this Moore m/c accepts a null sequence which is not accepted by Mealy m/c. As in Mealy m/c, if input 1, then output is 1. To overcome this, either we neglect the response of Moore m/c to input 1 or we must add [illegible] starting state [illegible] to [illegible]

Scanned by CamScanner

are same as q_1, but output is 0.

Moore machine

Present state	Next state $a=0$	Next state $a=1$	Output
→ q_0	q_3	q_{20}	0
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Procedure for transforming a Moore machine into a Mealy Machine

(1) We have to define the output function λ for mealy machine as a function of present state and input symbol, we define λ' by:-

$$\lambda'(q, a) = \lambda(\delta(q, a))$$ for all states q and all input symbols a.

(2) The transition function is same as that of given Moore machine.

2. Construct a Mealy machine which is equivalent to the following Moore machine:-

Scanned by CamScanner

Moore machine

Present state	Next state		Output
	a=0	a=1	
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

Mealy machine

Present state	Next state			
	a=0		a=1	
	state	o/p	state	o/p
→ q_0	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0

Note: We can reduce the no. of states in any model by considering states with identical transitions. If 2 states have identical transitions, then we can delete one of them

Scanned by CamScanner

Q. Mealy Machine

Present state	Next state			
	a = 0		a = 1	
	state	output	state	output
→ q_1	q_1	0	q_2	0
q_2	q_1	0	q_3	1
q_3	q_1	0	q_3	1

Here q_2 and q_3 are identicals so we can delete one of them say q_3, we get: (and replace q_3 by q_2)

Present state	Next state			
	a = 0		a = 1	
	state	output	state	output
→ q_1	q_1	0	q_2	0
q_2	q_1	0	q_2	1

Scanned by CamScanner

Grammar

A Grammar G_1 is defined as quadruple

$$G_1 = (V, T, S, P)$$

where

V is finite set of variables

T is finite set of Terminal symbols

$S \in V$, is start variable

P is set of production rules / substitution rules.

Start Variable usually occurs on the L.H.S. of the topmost rule. We use a grammar to describe a language by generating each string of that language in the following manner;

1. Write down the start variable.
2. Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the R.H.S. of that rule.
3. Repeat step 2 until no variables remain.

Rules are of form $x \to y$

where $x \in (V \cup T)^+$ ~~left~~ / usually not null string

$y \in (V \cup T)^*$

we write $x \Rightarrow y$

or "x derives y"

"y is derived from x"

If $w_1 \Rightarrow w_2 \Rightarrow w_3 \cdots \Rightarrow w_n$

we say that w_1 derives w_n

& write $w_1 \overset{*}{\Rightarrow} w_n$

* indicates that an unspecified no. of steps can be taken to derive w_n from w_1.

Also $w \overset{*}{\Rightarrow} w$

Let $G = (V, T, S, P)$ be a grammar

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

is the language generated by G

If $w \in L(G)$, then the sequence

$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n \Rightarrow w$

is a derivation of sequence w.

The strings $S, w_1, w_2, \cdots w_n$ which contain variables as well as terminal symbols are called sentential forms of the derivation.

Scanned by CamScanner

~~Q. $G = (V, T, S, P)$~~

Q. Find language generated by:

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

$S \rightarrow Ab$

$A \rightarrow aAb$

$A \rightarrow \lambda$

~~$A \Rightarrow \lambda$~~

~~$A \Rightarrow aAb \Rightarrow ab$~~

~~$A \Rightarrow aAb \Rightarrow aabb$~~

$S \Rightarrow Ab \Rightarrow b$

~~$S \Rightarrow b$~~

$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow abb$

$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow aaAbbb$
$\Rightarrow aabbb$

$L(G) = \{b, abb, aabbb, \ldots\}$

$L(G) = \{a^n b^{n+1} : n \geq 0\}$

Scanned by CamScanner

Q: Find language.

$G_1 = (\{A, S\}, \{a, b\}, S, P_1)$

with P_1 as

$S \to aAb \mid \lambda$

$A \to aAb \mid \lambda$

$S \to aAb$

$S \to \lambda$

$A \to aAb$

$A \to \lambda$

$L(G_1) = \{\lambda, ab, aabb \ldots\}$

$S \Rightarrow \lambda$

$A \Rightarrow aAb \Rightarrow ab$

~~$S \Rightarrow aAb$~~ $A \Rightarrow aAb \Rightarrow aaAbb$

$\Rightarrow aabb$

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

For a language, there can be 2 grammars but for a grammar cannot have 2 languages.

Ambiguous grammar → to derive a string more than 1 form exist

Scanned by CamScanner

Q: Find language

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

with P given by:

$S \to asa \mid bsb \mid \lambda$

① $S \to asa$ $L(G) = \{\lambda, aa,$

② $S \to bsb$ $bb, aaaa,$

③ $S \to \lambda$ $bbbb \ldots\}$

Using ③

$S \Rightarrow \lambda$

② $S \Rightarrow aSa \Rightarrow aa$

① $S \Rightarrow bSb \Rightarrow bb$

$S \Rightarrow bSb \Rightarrow baSab \Rightarrow baaab$

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

$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aaaaa$

$S \Rightarrow bSb \Rightarrow bbSbb \Rightarrow bbbbb$

$\Rightarrow$

$L = \{w.w^R : w \in \{a, b\}^*\}$

$*$ indicates that string w consists of symbol a & b

Scanned by CamScanner

Q.
$G = (\{S\}, \{a, b\}, S, P)$

with P defined by:

① $S \to aSb$

② $S \to \lambda$

$S \Rightarrow aSb$
$\Rightarrow aaSbb$
$\Rightarrow a^2Sb^2$
$\Rightarrow a^nSb^n$
$\Rightarrow a^nb^n$

③ $S \Rightarrow \lambda$

④ $aSb \Rightarrow ab$ (using ② put this $S = \lambda$)

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$
(using ① $S = aSb$) (using ② $S = \lambda$)

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$
(sentential form)

$L(G) = \{\lambda, ab, aabb, aaabbb, \ldots\}$

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

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

Scanned by CamScanner

Ch - 1 to 8 Peter Linz 20

Find grammar for:

$L = \{a^n b^{n+1} : n \geq 1\}$

$L = \{abb, aabbb, aaabbbb, \ldots\}$

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

$A \to \lambda$

$S \to aAb$

$A \to Ab$

$S \Rightarrow aAbb \Rightarrow abb$

$S \Rightarrow aAbb \Rightarrow$

$A \to aAb$

$A \to Ab$

$S \to Ab$

$A \to aAb$

$A \to ab$

$S \to Ab \Rightarrow abb$

$S \Rightarrow Ab \Rightarrow$

$aAbb$

$\Rightarrow aaabbb$

Scanned by CamScanner

Q: Find language generated by:-

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

P is given by:

$S \to SS \mid asb \mid bsa \mid \lambda$

$S \to SS$

$S \to asb$

$S \to bsa$

$S \to \lambda$

$L(G) = \{\lambda, ab, ba, abba, abab, \ldots\}$

$S \Rightarrow SS \Rightarrow asbbsa$

$\Rightarrow abba$

$S \Rightarrow asb \Rightarrow ab$

$S \Rightarrow bsa \Rightarrow ba$

~~$a^n b^n$~~

$S \Rightarrow asb \Rightarrow absab$

$\Rightarrow abab$

$L(G) = \{w : n_a(w) = n_b(w), w \in \{a, b\}^*\}$

Scanned by CamScanner

epts atleast

a

Chomsky classification of languages (21)

In the definition of grammar (V, T, S, P), V_N and T are sets of symbols and $S \in V_N$. If we want to classify grammars, we have to do it only by considering form of productions.

Chomsky classification of grammars and languages

1) Type 0 grammar (Phase structure grammar)

Is any grammar without any restrictions.

In the production of the form, $\phi A \psi \rightarrow \phi \alpha \psi$, where A is variable, ϕ is left context, ψ the right context and $\phi \alpha \psi$ the replacement string.

eg. $abAbcd \rightarrow abABbcd$.

Here $ab \rightarrow$ left context

$bcd \rightarrow$ right context

$+ \alpha \Rightarrow AB$

eg. $AC' \rightarrow A$.

Here A is left context

Λ is right context

$+ \alpha = \Lambda$

A production without any restriction is called a type 0 production.

Scanned by CamScanner

② Type 1 or context sensitive or dependent grammar

- A production of the form $\phi A \psi \to \phi \alpha \psi$ is called type 1 production if $\alpha \neq \wedge$. ie here erasing of A is not permitted.

- A grammar is called type 1 or context sensitive if all its productions are type 1 productions. (The production $S \to \wedge$ is also allowed in a type 1 grammar, but in this case S does not appear on R.H.S of any production)

eg. $aAbcD \to abcDbcD$, $AB \to AbBc$, $A \to abA$

- A language generated by a type 1 grammar is called type 1 or context sensitive language.

③ Type 2 or context free grammar

- A type 2 production is a production of the form $A \to \alpha$ where $A \in V_N$ and $\alpha \in (V \cup T)^*$ ie L.H.S has no left context or right context.

eg. $S \to Aa$, $A \to a$, $A \to \wedge$ are type 2 productions.

- A grammar is called type 2 grammar or context free grammar if it only contains type-2 productions.

- A language generated by a context free grammar is called type-2 language or context free language.

Scanned by CamScanner

(4) Type 3 or regular grammar

- A production of the form $A \to a$ or $A \to aB$ where $A, B \in V$ and $a \in T$ is called type 3 production. ($S \to \Lambda$ is allowed here but S should on appear in RHS of any other production.)
- A grammar is called type 3 or regular grammar if all its productions are type 3 productions.

- A language generated by a regular grammar is called type-3 language or regular language

Q: Find the highest type number which can be applied to the following productions:

(a) $S \to Aa$, $A \to c \mid Ba$, $B \to abc$

Here $S \to Aa$, $A \to Ba$ and $B \to abc$ are type 2 and $A \to c$ is type 3.

So, highest type is 2.

(b) $S \to ASB \mid d$, $A \to aA$

$S \to ASB$ is type 2 and $A \to aA$ and $S \to d$ is type 3, so highest type is 2.

Scanned by CamScanner

(c) $S \to aS \mid ab$

$S \to aS$ is type 3 and $S \to ab$ is type 2.

So, highest type is 2.

Scanned by CamScanner

Regular Expressions

The regular expressions are useful for representing certain sets of strings in algebraic fashion.

The formal definition of regular expression is as:

(1) Any terminal symbol, Λ and ϕ are regular expressions. When we view a in Σ as a regular expression, when denote it by **a**.

(2) The union of 2 regular expressions R_1 and R_2 is $R_1 + R_2$ which is also regular expression

(3) The concatenation of 2 regular expressions R_1 and R_2 is $R_1 R_2$ which is also regular expression.

(4) The iteration (or closure) of a regular expression R, is R^* is also a regular expression.

(5) If R is a regular expression, then (R) is also a regular expression.

(6) The regular expression over Σ are precisely those obtained recursively by application of the rules 1-5 once or several times.

Note: In absence of parentheses, we have the hierarchy of operations as follows:
iterations concatenation and union.

Regular set: Any set represented by a regular expression is called a regular set.

If for eg. $a, b \in \Sigma$, then (i) **a** denotes the set $\{a\}$
(ii) **a+b** denotes $\{a, b\}$
(iii) **ab** denotes $\{ab\}$
(iv) a^* denotes set $\{\Lambda, a, aa, aaa\}$
(v) $(a+b)^*$ denotes $\{a, b\}^*$

Scanned by CamScanner

The set represented by R is $L(R)$
If R_1 and R_2 are any 2 regular expressions.
Then:

(1) $L(R_1 + R_2) = L(R_1) \cup L(R_2)$
(2) $L(R_1 R_2) = L(R_1)\, L(R_2)$
(3) $L(R^*) = (L(R))^* = \bigcup_{n=0}^{\infty} L(R)^n$

Q: Describe the following set by regular expressions:-

(a) $\{101\} \Rightarrow 101$
(b) $\{\wedge, ab\} \Rightarrow \wedge + ab$
(c) $\{\wedge, 0, 00, 000 - - -\} \Rightarrow 0^*$
(d) $\{1, 11, 111, - - - -\} \Rightarrow 1(1)^*$
as concatenation of 1 and any element of 1^*
(e) L_1 is set of all strings of 0's and 1's ending in 00 $\Rightarrow (0+1)^*00$
(f) L_2 is set of all strings of 0's and 1's beginning with 0 and ending with 1
$0(0+1)^*1$

Identities for Regular Expressions.

I_1	$\phi + R = R$
I_2	$\phi R = R\phi = \phi$
I_3	$\wedge R = R \wedge = R$
I_4	$\wedge^* = \wedge$ and $\phi^* = \wedge$
I_5	$R + R = R$
I_6	$R^* R^* = R^*$

Scanned by CamScanner

I_7 $RR^* = R^*R$

I_8 $(R^*)^* = R^*$

I_9 $\lambda + RR^* = \lambda + R^*R = R^*$

I_{10} $(PQ)^*P = P(QP)^*$

I_{11} $(P+Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$

I_{12} $(P+Q)R = PR + QR$ and
$R(P+Q) = RP + RQ$

Theorem (Arden's theorem): Let P and Q be 2 regular expressions over Σ. If P does not contain λ, then the following equation in R, namely $R = Q + RP$ has a unique solution given by $R = QP^*$

Q: Prove $(1+00^*1) + (1+00^*1)(0+10^*1)^*$
$(0+10^*1) = 0^*1(0+10^*1)^*$

$\Rightarrow (1+00^*1)\left(\lambda + (0+10^*1)^*(0+10^*1)\right)$ (using I_{12})

$\Rightarrow (1+00^*1)(0+10^*1)^*$ (using I_9)

$\Rightarrow 1(1+00^*)(0+10^*1)^*$ (using I_{12})

$\Rightarrow 0^*1(0+10^*1)^*$ (using I_9)

$= R.H.S.$

Scanned by CamScanner

Finite Automata and Regular Expr

Transition system containing Λ-moves

Transition system can be generalized by permitting Λ-transitions or Λ-moves, these occur whenever input is applied. But it is possible to convert a transition system with Λ-moves into an equivalent transition system without Λ-moves.

Suppose, we want to replace a Λ-move from vertex V_1 to vertex V_2; Then:

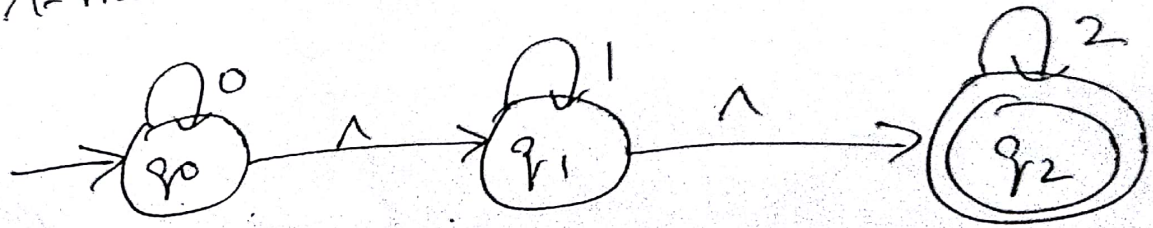
(1) Find all edges starting from V_2.

(2) Duplicate all these edges starting from V_1, without changing the edge labels.

(3) If V_1 is an intial state, make V_2 also intial state.

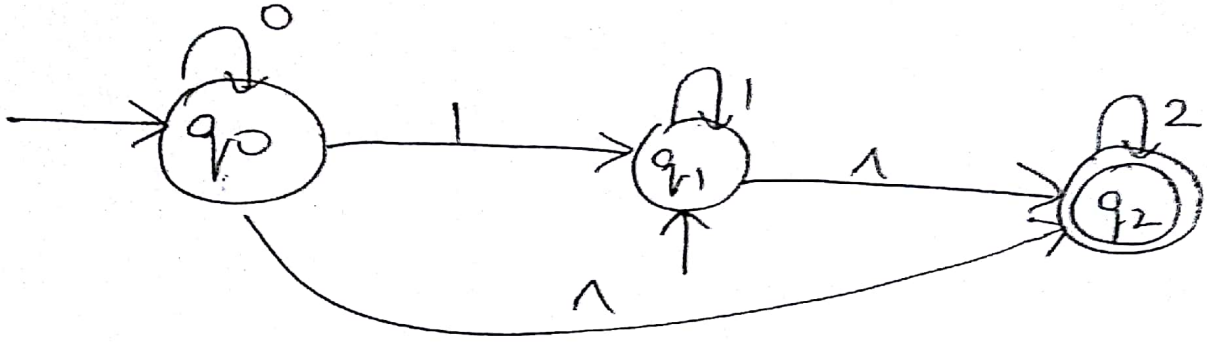
(4) If V_2 is final state, make V_1 also final state.

Q1- Consider a finite automaton, with Λ-moves Obtain an equivalent automaton without Λ-moves

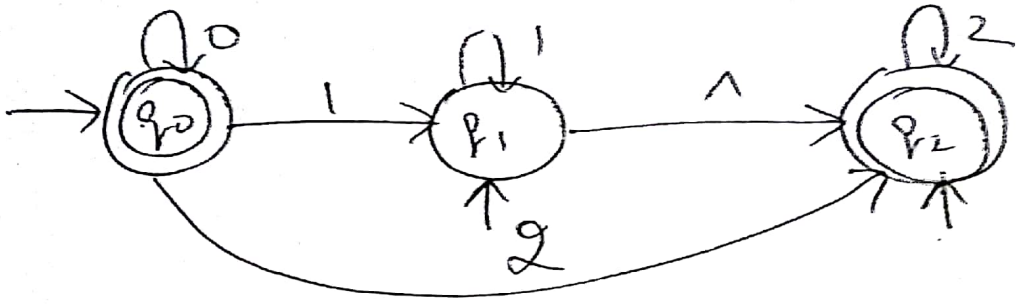


Scanned by CamScanner

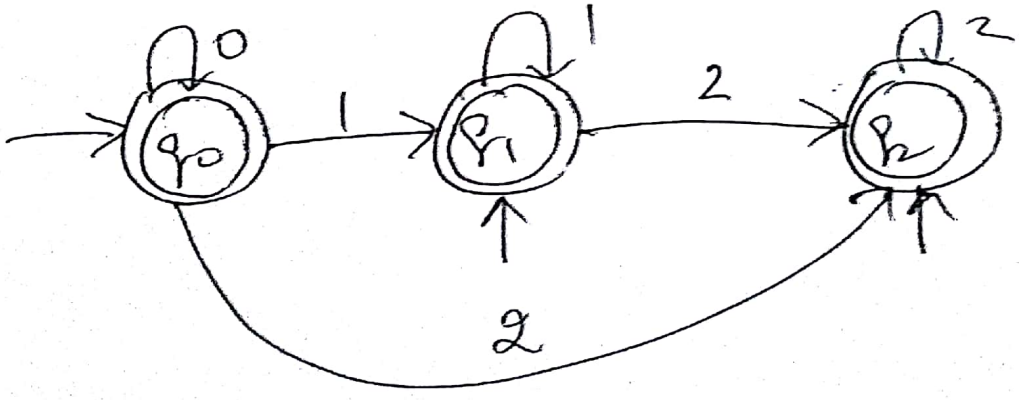
(1) We first remove $\wedge$ from q_0 to q_1



(2) We remove $\wedge$ from q_0 to q_2



(3) We remove $\wedge$ from q_1 to q_2



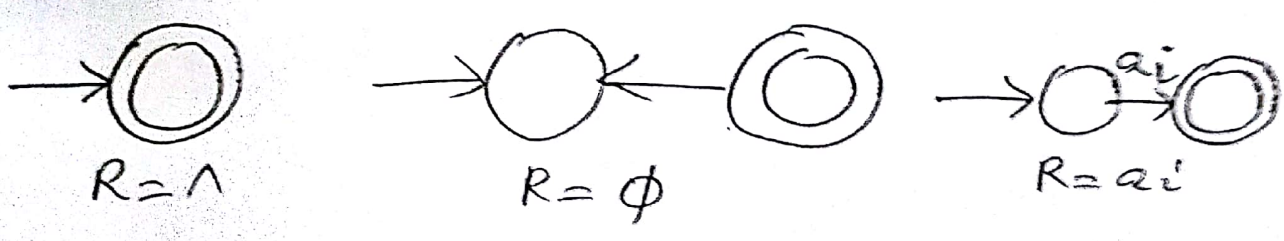
Scanned by CamScanner

NDFA with Λ-moves and Regular Expressions

Kleene's theorem: If R is a regular expression over Σ representing $L \subseteq \Sigma^*$, then there exists an NDFA M with Λ-moves such that $L = T(M)$.

Proof. Proof by principle of induction on total no. of characters in R. (eg. $\Sigma, \Lambda, \phi, *, +$). (eg. $R = \Lambda + 10^* 11^* 0$, no. of char = 9)
Let $L(R)$ denote the regular set represented by R.

Basic step:- Let the no. of characters in R be 1.
Then $R = \Lambda$ or $R = \phi$ or $R = a_i$, $a_i \in \Sigma$.



Inductive step:- Assume that the theorem is true for regular expressions having n characters. Let R be a regular expression having $n+1$ characters. Then,

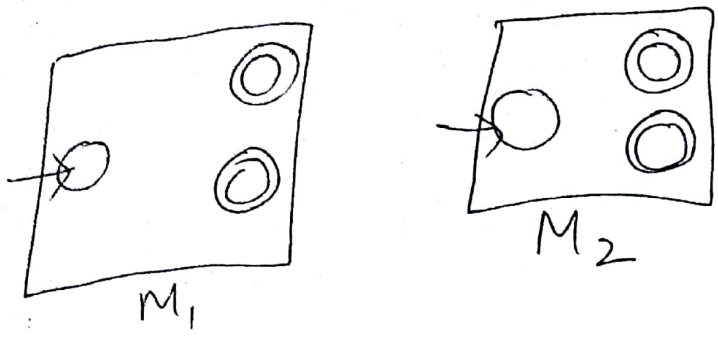
$R = P + Q$ or $R = PQ$ or $R = P^*$

according as last operator in R is $+$, product or closure. P and Q are regular expressions

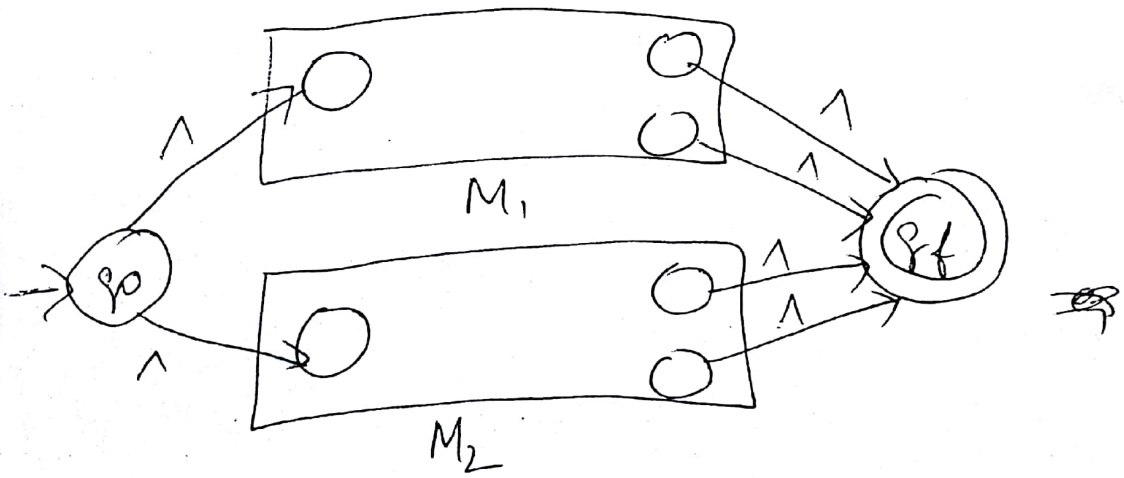
Scanned by CamScanner

having n characters or less.

$L(P)$ and $L(Q)$ are recognized by M_1 and M_2 where M_1 and M_2 are NDFAs with Λ-moves such that $L(P) = T(M_1)$ and $L(Q) = T(M_2)$



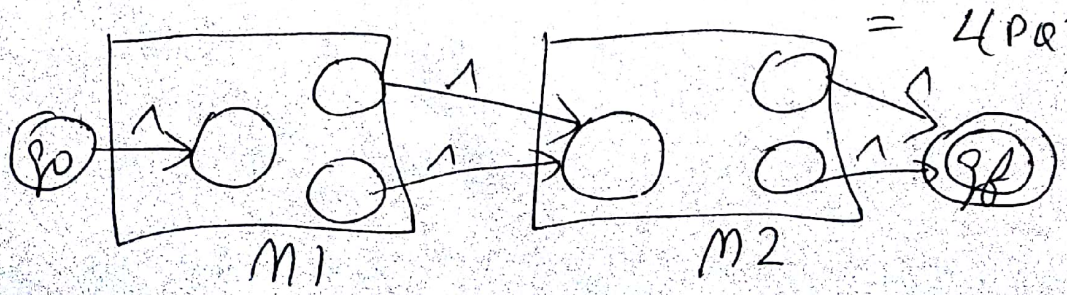
Case ①: $R = P + Q$:- In this case we construct an NDFA M with Λ-moves that accepts $L(P+Q)$ as:



$$T(M) = T(M_1) \cup T(M_2)$$
$$= L(P+Q)$$

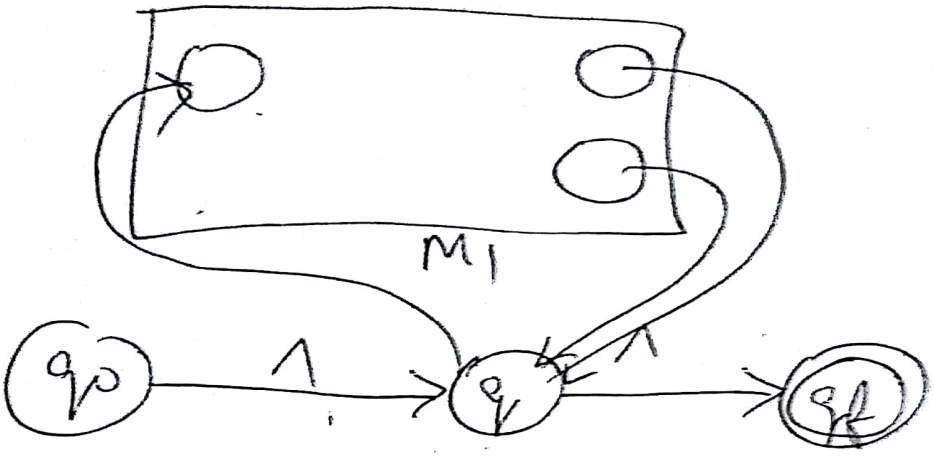
Case ② $R = PQ$:-

$$T(M) = T(M_1) \cdot T(M_2)$$
$$= L(PQ)$$



Scanned by CamScanner

Case (3): $R = P*$



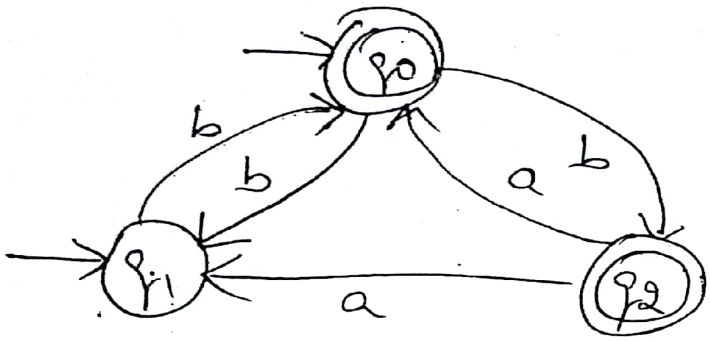
Thus in all cases, there exists an NDFA M with Λ-moves accepting regular expression R with n+1 characters.

Scanned by CamScanner

Conversion of NDFA to DFA (Regular Expression)

The construction is similar to the one done earlier, only in the initial step, take set of all initial states.

Q: Find the equivalent DFA:-



δ

State / Σ	a	b
→ q0		q1, q2
→ q1		q0
(q2)	q0, q1	

Equivalent DFA

δ

State / Σ	a	b
[q0, q1]	ϕ	[q0, q1, q2]
ϕ	ϕ	ϕ
[q0, q1, q2]	[q0, q1]	[q0, q1, q2]

Scanned by CamScanner

Conversion of a finite automaton to regular expression

(Using Algebraic method using Arden's theorem)

The following assumptions are made regarding the transition system:-

1. The transition graph does not have λ-moves.
2. It has only one initial state, say V_1.
3. Its vertices are $V_1, \ldots V_n$.
4. V_i the r.e. represents set of strings accepted by the system even though V_i is final state.
5. α_{ij} denotes the r.e. representing set of labels of edges from V_i to V_j. When there is no edge $\alpha_{ij} = \phi$.

$$V_1 = V_1\alpha_{11} + V_2\alpha_{21} + \ldots + V_n\alpha_{n1} + \lambda$$

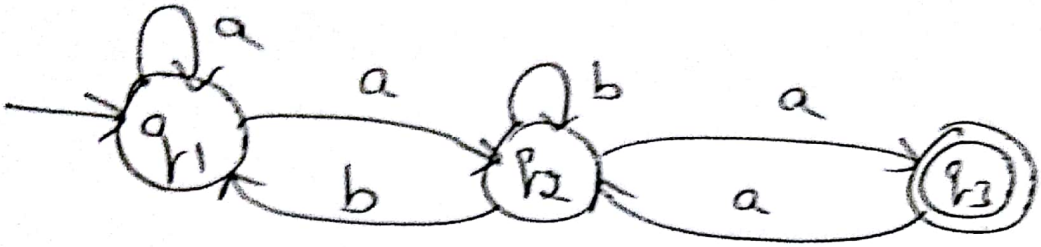
$$V_2 = V_1\alpha_{12} + V_2\alpha_{22} + \ldots + V_n\alpha_{n2}$$

$$V_n = V_1\alpha_{1n} + V_2\alpha_{2n} + \ldots + V_n\alpha_{nn}$$

For getting set of strings recognized by the transition system we have to take union of all V_i's corresponding to final states.

Scanned by CamScanner

Q: Consider the transition system. Find the regular expression:- (2)



We can apply arden's theorem as graph does not contain any $\wedge$-move and there is only one initial state.

$$q_1 = q_1 a + q_2 b + \wedge \quad \text{— (1)}$$

$$q_2 = q_1 a + q_2 b + q_3 a \quad \text{— (2)}$$

$$q_3 = q_2 a \quad \text{— (3)}$$

Putting (3) in (2)

$$\Rightarrow q_2 = q_1 a + q_2 b + q_2 aa$$

$$\Rightarrow q_2 = q_1 a + q_2 (b + aa)$$

(By applying arden's theorem)

$$\Rightarrow q_2 = q_1 a (b + aa)^* \quad \text{— (4)}$$

Putting (4) in (1)

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

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

(using arden's theorem)

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

Scanned by CamScanner

$q_2 = q_1 a(b+aa)^* b)^* a(b+aa$

$q_2 = (a+a(b+aa)^* b)^* a(b+aa$

$q_3 = q_2 a$

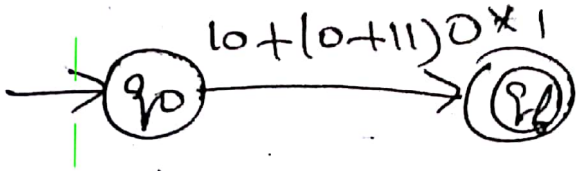
$$q_3 = (a+a(b+aa)^* b)^* a(b+aa)^* a$$

Construction of finite automata equivalent to a regular expression

(1) Construct a transition graph equivalent to the given regular expression using Λ-moves.

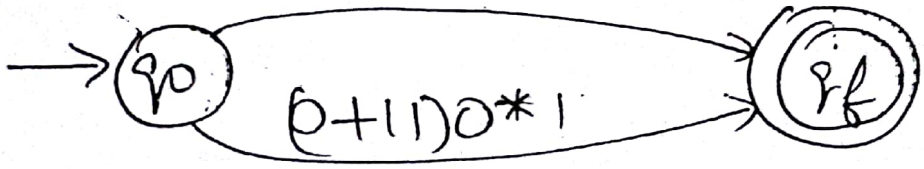
(2) Construct the transition table without Λ-moves and find the equivalent DFA. Reduce the no. of states if possible.

Q: Construct a DFA with reduced states equivalent to r.e. $10+(0+11)0^*1$



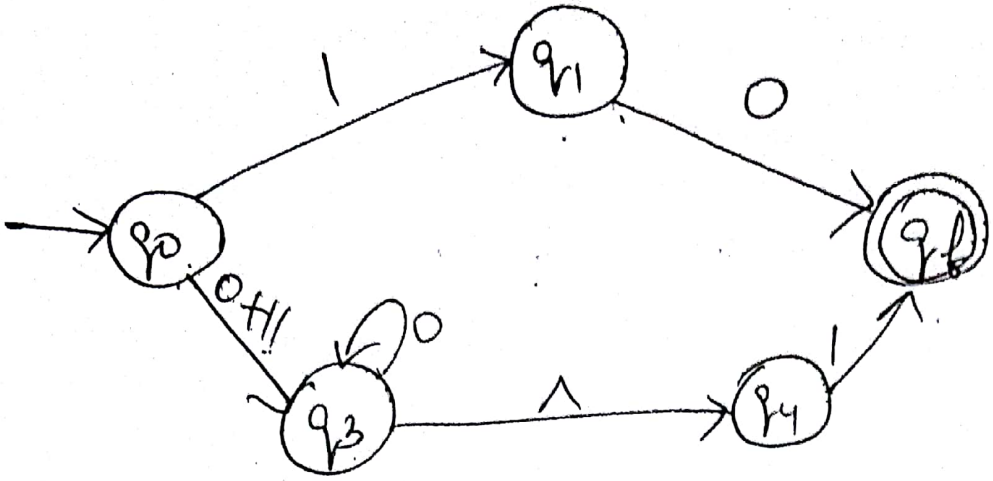
We eliminate +

10

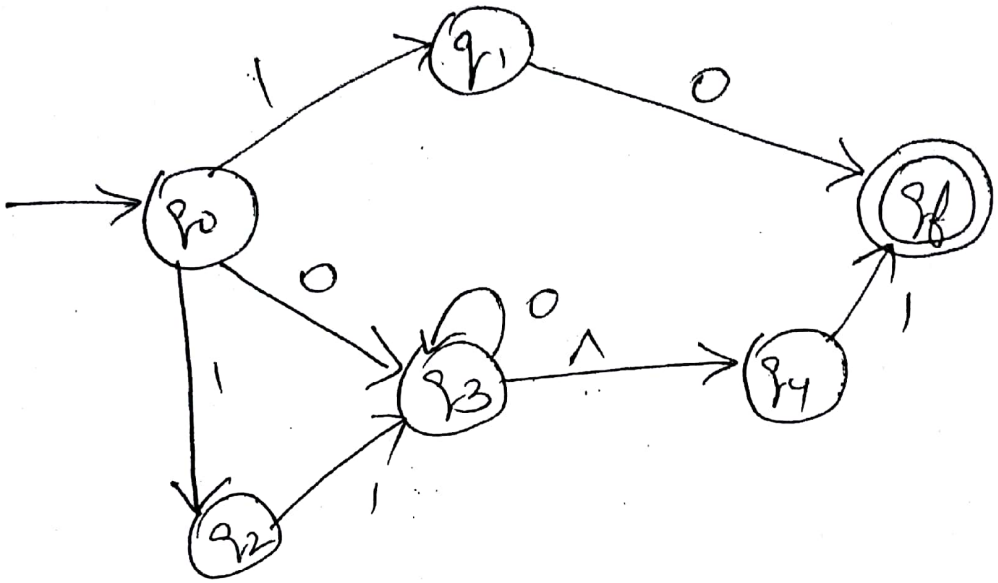


We eliminate concatenation and *

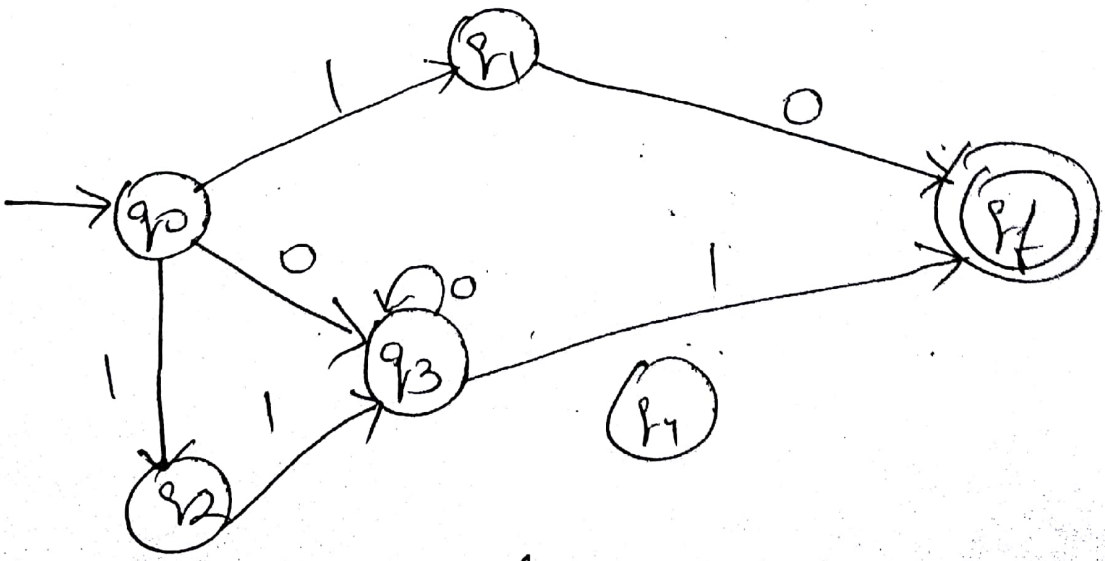
Scanned by CamScanner



Elimination of +



Elimination of Λ-moves



We can neglect q_7

Scanned by CamScanner

0...0

(2)

State/Σ	0	1
→ q_0	q_3	q_1, q_2
q_1	q_f	ϕ
q_2	ϕ	ϕ q_3
q_3	q_3	q_f
q_f	ϕ	ϕ

(3) The equivalent DFA is:

State/Σ	0	1
→ $[q_0]$	$[q_3]$	$[q_1, q_2]$
$[q_3]$	q_3	q_f
$[q_1, q_2]$	q_f	q_3
$[q_f]$	ϕ	ϕ
ϕ	ϕ	ϕ

Scanned by CamScanner

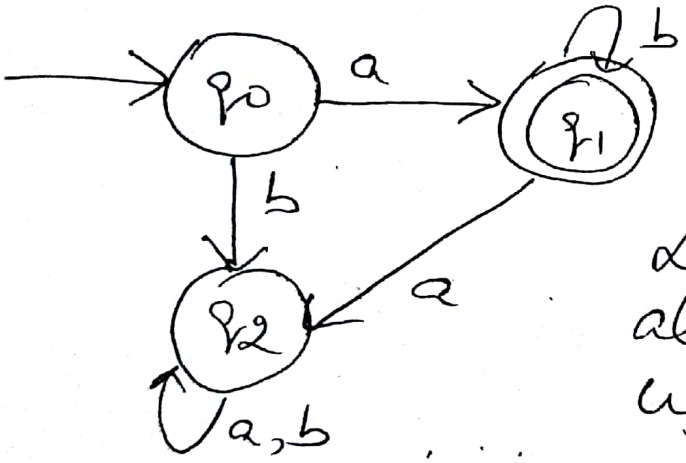
Pumping Lemma

Theorem:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton with n states. Let L be the regular set (lang.) accepted by M. Then, there exists some positive integer m such that for any string $w \in L$ with $|w| \geq m$ and $m \geq n$, can be decomposed as $w = xyz$ with $|xy| \leq n$, $y \neq \Lambda$ and $xy^iz \in L$ for each $i \geq 0$.

Proof:

Eg. $L = \{ab^n : n \geq 0\}$



Let us consider all strings w with $|w| \geq 3$

Now w can be decomposed as

$w = xyz$

where $x = a$
$y = b$
$z = b$

$\left[|xy| = |ab| = 2 \leq 3 \right.$
$\left. |y| = |b| = 1 \geq 1 \right]$

Now, since every $w_i = xy^iz$
$= ab^ib \in L$ for $i > 0$, $\therefore L$ is regular

Scanned by CamScanner

Proof: Let $w = a_1 a_2 \dots a_m$, $m \geq n$

$\delta(q_0, a_1 a_2 \dots a_i) = q_i$ for $i = 1, 2 \dots m$;

$Q_1 = \{q_0, q_1 \dots q_m\}$

Here Q_1 is sequence of states in the path with path value $w = a_1 a_2 \dots a_m$. As there are only n distinct states, atleast 2 states in Q_1 must coincide. Let us take them as q_j and q_k $(q_j = q_k)$. Then, $0 \leq j < k \leq n$.

So, w can be decomposed as:-

Let $\underbrace{a_1 a_2 \dots a_j}_{x}$, $\underbrace{a_{j+1} \dots a_k}_{y}$ and $\underbrace{a_{k+1} \dots a_m}_{z}$

As $k \leq n$, $|xy| \leq n$ and $w = xyz$

Path will be:-

It starts with q_0, on applying string x it reaches $q_j (= q_k)$. On applying string y, it comes back to $q_j (= q_k)$. So after application of y^i for $i \geq 0$, automaton remains in same state q_j. On applying z, it reaches q_m.

Hence $xy^i z \in L$.

Scanned by CamScanner