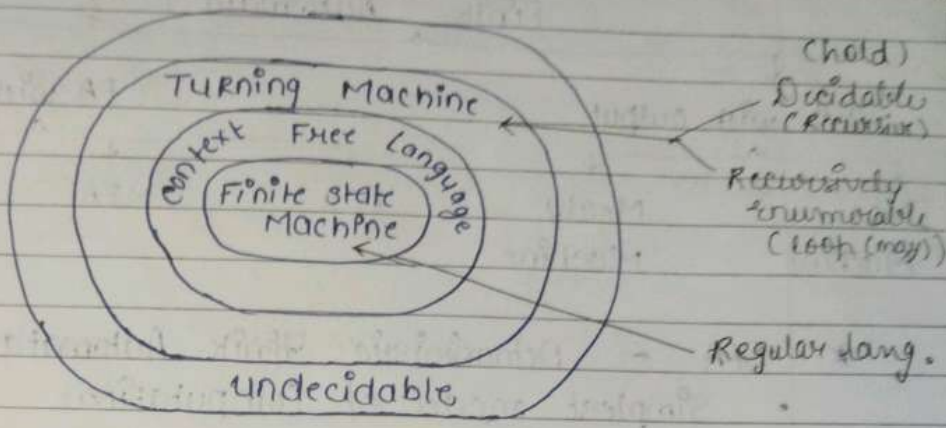
Theory Of Computation



Finite State Machine

Prerequisites :

- Symbol ← a, b, c, 0, 1, 2, 3 ....
- Alphabet (Σ) ← Collection of symbols ex. {0,b}
- String ← Sequence of symbols ex. a, b, 1, ab, bb
- Language ← set of strings < finite / infinite

ex. of Σ = {0,1}

L ← {00, 01, 10, 11} ← set of all strings of length 2

Powers of sigma (Σ) for (0,1)

Σ^0 = Set of all strings of length 0 : $\Sigma^0 = \{\epsilon\}$

Σ^1 = Set of all strings of length 1 : $\Sigma^1 = \{0,1\}$

Σ^2 = Set of all strings of length 2 : $\Sigma^2 = \{00,01,10,11\}$

Σ^n = Set of all strings of length n

Cardinality : no. of elements in a set

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots$

Finite State Machine

Finite Automata (FA)

- FA with output
  - Moore Machine
  - Mealy Machine
- FA without output
  - DFA
  - NFA
  - ϵ-NFA

★ DFA ← Deterministic Finite Automata

- Simplest model of computation
- Very limited memory
- one unique next state
- It has no choices or randomness

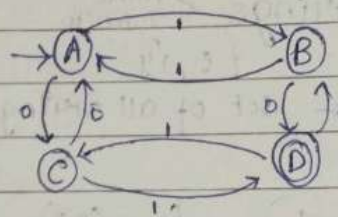
Q = Set of all states = $\{A, B, C, D\}$

Σ = Inputs = $\{0, 1\}$

q_0 = Start state / initial state = A

F = Set of final states = $\{D\}$

δ = transition function from $Q \times \Sigma \longrightarrow Q$

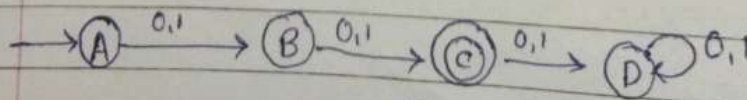


	0	1
A	C	B
B	D	A
C	A	D
D	B	C

- Dead state OR Trap state, : the state that can't reach the final state.

Ex. Construct a DFA that accepts sets of all strings over $\{0,1\}$ of length 2

→ $\Sigma = \{0,1\}$ $L = \{00, 01, 10, 11\}$

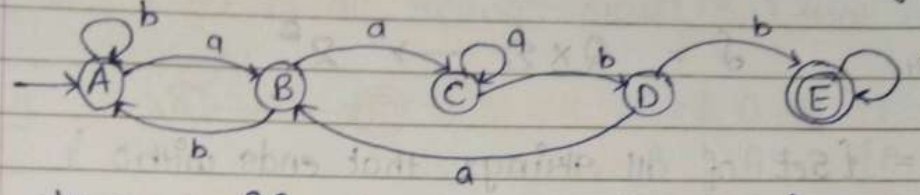


final state

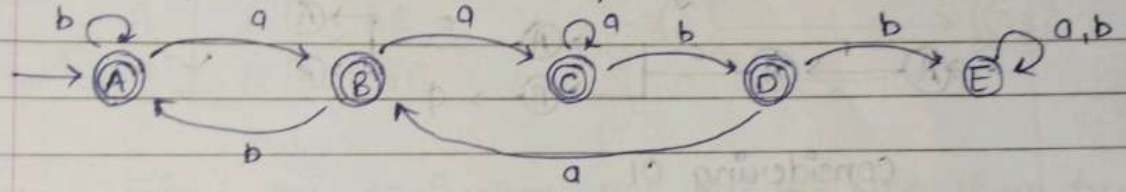
Dead state

Ex. Construct a DFA that accepts any strings over {a,b} that doesn't contain the string aabb in it.

→ Let us consider a DFA that accepts all strings over {a,b} that contains the string aabb in it.



Now, flip the states by making the final states into non-final states and make the non-final states into final states.



* A language is said to be Regular Language if and only if some finite state Machine recognizes it.

* Memory of FSM is very limited, it can't store or count strings.

Operations on Regular Languages

- UNION - $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- CONCATENATION - $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
- STAR - $A^* = \{x_1 x_2 x_3 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

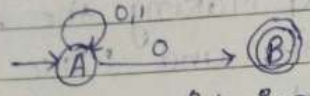
Precedence : $* > \circ > +$

Scribble
Page :
Date : / /

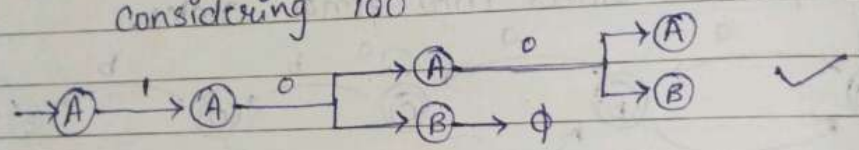
* NFA ← Non-deterministic finite Automata
  - Given the current state there could be multiple next states.
  - The next state may be choosen at random
  - All the next state may be choosen in parallel

here, $\delta = Q \times \Sigma \longrightarrow 2^Q$

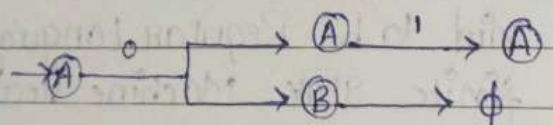
Ex. L = {Set of all strings that ends with 0 }



Considering 100



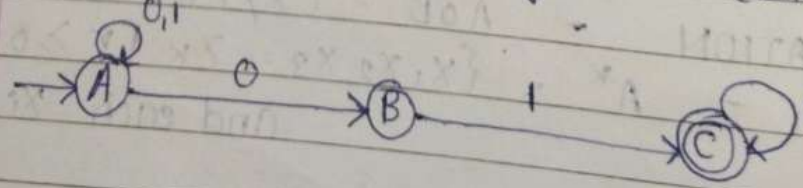
Considering 01



(dead configuration)

* if there is any way to run the machine that ends in any set of states out of which atleast one state is a final state, then the NFA accepts.

Ex. L = { Set of all strings that contain '01' }



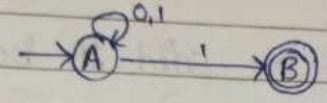
CONVERSION OF NFA to DFA

Every DFA is an NFA, but not vice versa
But there is an Equivalent DFA for every NFA.

→ NFA diagram → NFA state transition diagram
DFA diagram ← DFA state transition diagram ←

Ex. L = { Set of all strings over (0,1) that ends with 1 }

NFA

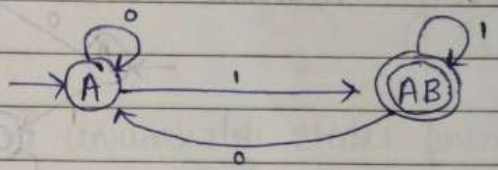


	0	1
A	A	{A, B}
B	ϕ	ϕ

Subset Construction Method

DFA

	0	1
A	A	AB
AB	A	AB

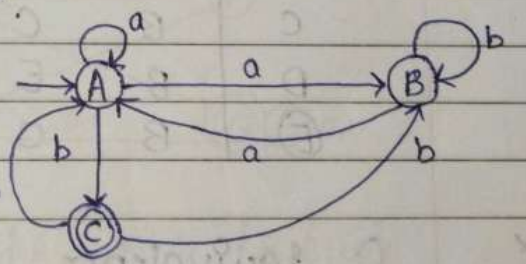


Que.. Find the equivalent DFA for the NFA. given by

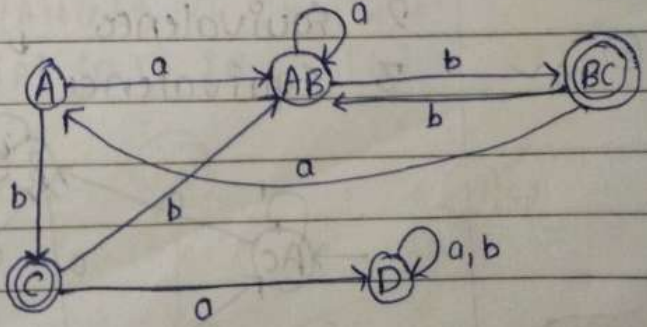
M = [{A, B, C}, (a, b), δ, A, {C}]

where δ is given by

	a	b
→ A	A, B	C
B	A	B
(C)	-	A, B



	a	b
→ A	AB	C
AB	AB	BC
(BC)	A	AB
(C)	D	AB
D	D	D



Scribble
Page :
Date : / /

Minimization of DFA (Partitioning Method)

(Minimum no. of states possible)

* Two states are said to be equivalent if

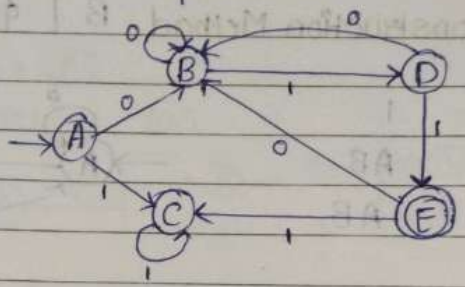
$\delta(A, x) \to F_{\text{final state}}$ OR $\delta(A, x) \not\to F$

& $\delta(B, x) \to F$ & $\delta(B, x) \not\to F$

x : any input string.

if $|x| = n$, then A and B are said to be n equivalent

Ex. Minimize



→	0	1
→A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C

① A, B ✓
A, C ✓
C, D ✗

② AB ✗
AC ✓
BC ✗

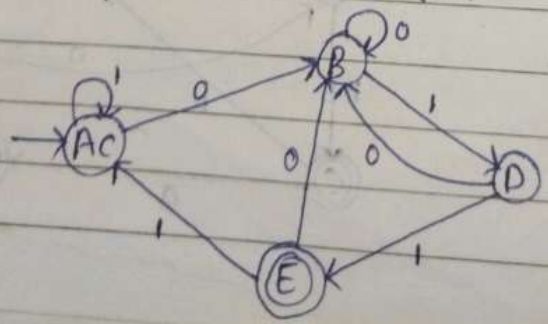
③ AC ✓

0 equivalence ↓ {A, B, C, D} {E}

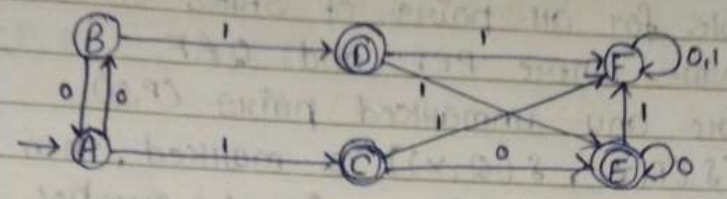
1 equivalence ↓ {A, B, C} {D} {E}

2 equivalence ↓ {A, C} {B} {D} {E}

3 equivalence {A, C} {B} {D} {E} → Same ↳ Exit



Minimizing when there are more than one final state

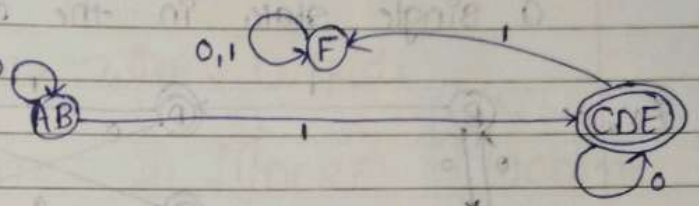


	0	1
→A	B	C
B	A	D
C	E	F
D	E	F
E	E	F
F	F	F

0 Eq. {A,B,F} {C,D,E}

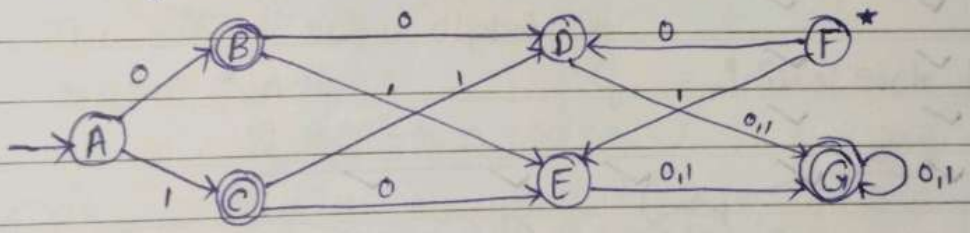
1 Eq. {A,B} {F} ~~{D} {C} {E}~~ {C,D,E}

2 Eq. {A,B} {F} {C,D,E}



Minimizing when there are unreachable states involved

★ A state is said to be unreachable if there is no way it can be reached from the initial state.



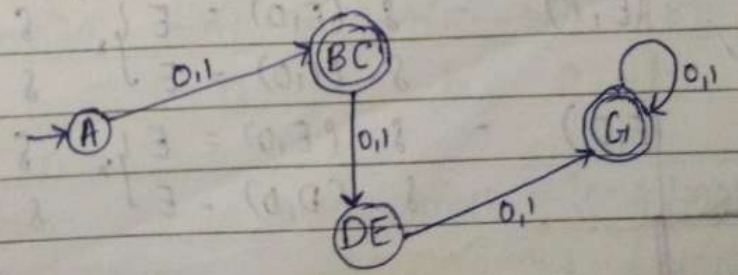
	0	1	
→A	B	C	
B	D	E	
C	E	D	
D	G	G	
E	G	G	
X F	D	E	X
G	G	G	

0 Eq. {A,D,E} {B,C,G}

1 Eq. {A,D,E} {B,C} {G}

2 Eq. {A} {D,E} {B,C} {G}

3 Eq. {A} {D,E} {B,C} {G}



(Minimization of DFA)

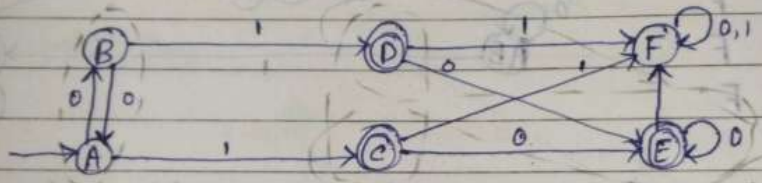
Myhill - Nerode Theorem OR Table filling Method

Scribble
Page:
Date: / /

STEPS:

1. Draw a table for all pairs of states (P,Q)
2. Mark all pairs where $P \in F$ and $Q \notin F$
3. IF there are any unmarked pairs (P,Q) such that [δ(P,x), δ(Q,x)] is marked, then mark [P,Q] where 'x' is an input symbol. Repeat this until NO More Markings can be Made.
4. Combine all the Unmarked pairs and make them a single state in the minimized DFA.

Ex.



	A	B	C	D	E	F
A						
B						
C	✓	✓				
D	✓	✓				
E	✓	✓				
F	✓	✓	✓	✓	✓	

not required
one side of diagonal

3rd Skp check

(B,A) - δ(B,0) = A, δ(A,0) = B ✗ ; δ(B,1) = D, δ(A,1) = C ✗

(D,C) - δ(D,0) = E, δ(C,0) = E ✗ ; δ(D,1) = F, δ(C,1) = F ✗

(E,C) - δ(E,0) = E, δ(C,0) = E ✗ ; δ(E,1) = F, δ(C,1) = F ✗

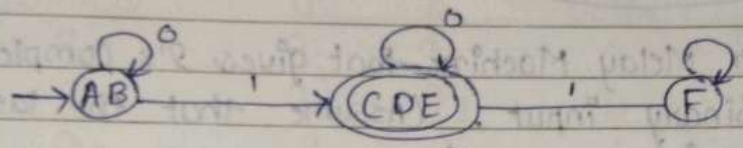
(E,D) - δ(E,0) = E, δ(D,0) = E ✗ ; δ(E,1) = F, δ(D,1) = F ✗

(F, A) - $\delta(F,0) = F$ ✓, $\delta(A,0) = B$ ✓ × $\delta(F,1) = F$ ✓, $\delta(A,1) = C$ ✓

(F, B) - $\delta(F,0) = F$ ✓, $\delta(B,0) = A$ ✓ × $\delta(F,1) = F$ ✓, $\delta(B,1) = D$ ✓

Unmarked pairs ← (A,B) (D,C) (E,C) (E,D)

=



Finite Automata with Outputs

MEALY Machine

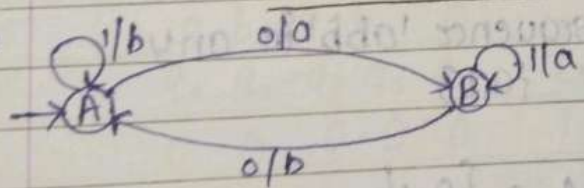
$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

Q ← finite set of states

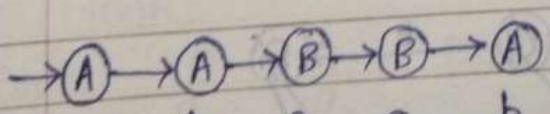
Σ ← finite non-empty set of Input alphabets

Δ ← The set of output alphab.

λ ← output function : $\Sigma \times Q \rightarrow \Delta$



Ex. 1 0 1 0



input length = output length

asynchronous / fast

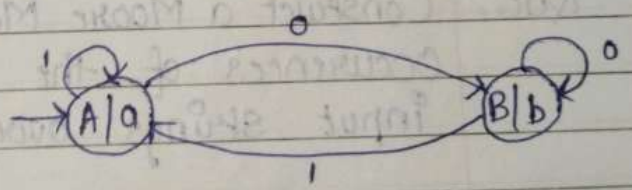
MOORE Machine

(1 kalaam sb same hai)

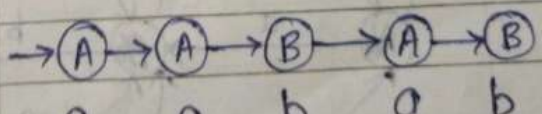
δ = Transition function : $Q \times \Sigma \rightarrow Q$

λ = output function : $Q \rightarrow \Delta$

q_0 = initial state / start state



Ex. 1 0 1 0

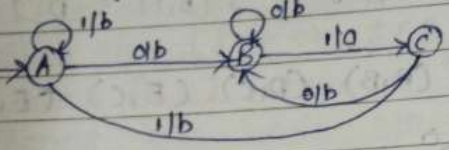


output length = input length + 1

synchronous / slow

Scribble
Page :
Date : / /

Que.. Construct a Melay Machine that prints '0' whenever the sequence '01' is encountered in any input binary string.

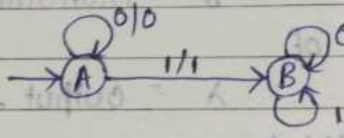


Check 0 1 1 0
b a bb

Que.. Construct a Melay Machine that gives 2's complement of any binary input. (Assume that the last carry bit is neglected)

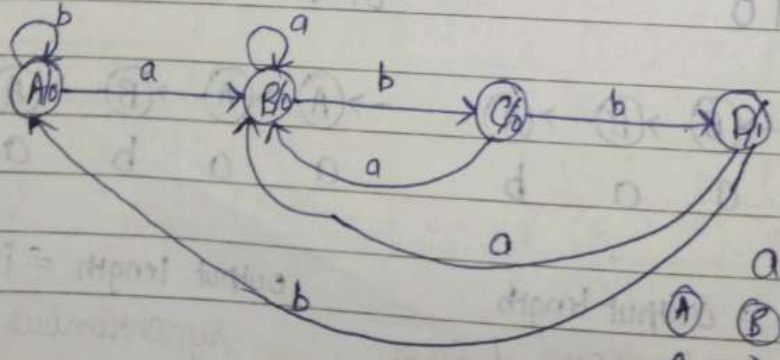
10100 ← 11100 ← 1111 ← (Pehle one k baad changed)
01011 00011 0000
+1 +1 +1
01100 00100 0001

We go Left to Right



Que.. Construct a Moore Machine that counts the occurences of the sequence 'abb' in any input strings over {a,b}

→ $\Sigma = \{a,b\}$ $\Delta = \{0,1\}$



Check
a b b
A B C D
0 0 0 1

Moore → Mealy : No of states remains same
Mealy → Moore : No. of states increased

Scribble

Page:
Date: / /

Ques. For the following Moore Machine the input alphabet is $\Sigma = \{a, b\}$ and the output alphabet is $\Delta = \{0, 1\}$. Run the following input sequences and find the respective outputs

(i) aabab (ii) abbb (iii) ababb

States	a	b	Outputs
→ q_0	q_1	q_2	0
q_1	q_2	q_3	0
q_2	q_3	q_4	1
q_3	q_4	q_4	0
q_4	q_0	q_0	0

(i) ~~aab~~ a a b a b

q_0 q_1 q_2 q_4 q_0 q_2

0 0 1 0 0 1

(ii) a b b b

q_0 q_1 q_3 q_4 q_0

0 0 0 0 0

(iii) a b a b b

q_0 q_1 q_3 q_4 q_0 q_2

0 0 0 0 0 1

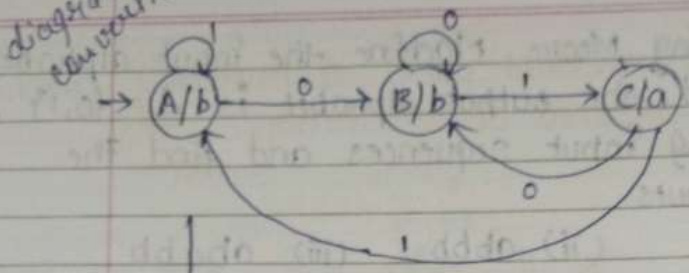
Converting Moore Machine into Mealy Machine

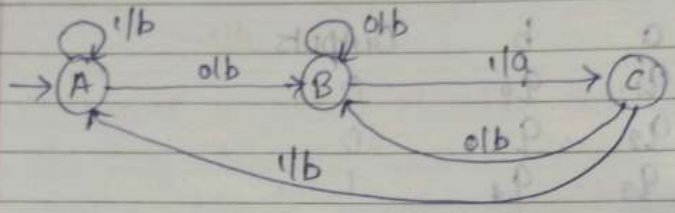
Table

MOORE			
State	0	1	Output
→ q_0	q_1	q_2	1
q_1	q_3	q_2	0
q_2	q_2	q_1	1
q_3	q_0	q_3	1

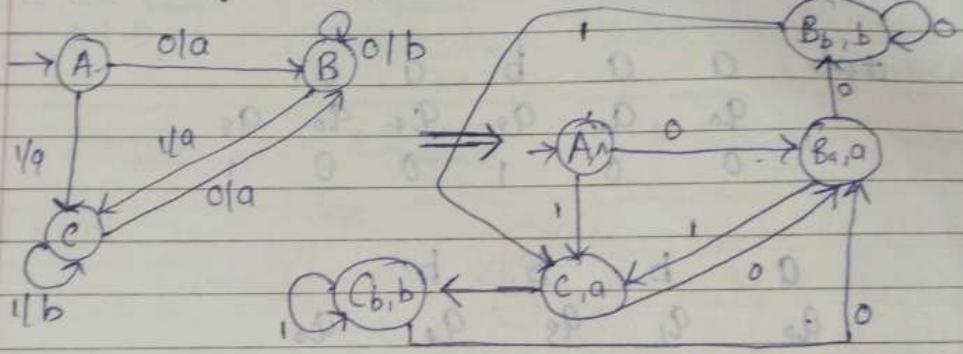
MELAY		
State	0	1
→ q_0	$q_1, 0$	$q_2, 1$
q_1	$q_3, 1$	$q_2, 1$
q_2	$q_2, 1$	$q_1, 0$
q_3	$q_0, 1$	$q_3, 1$

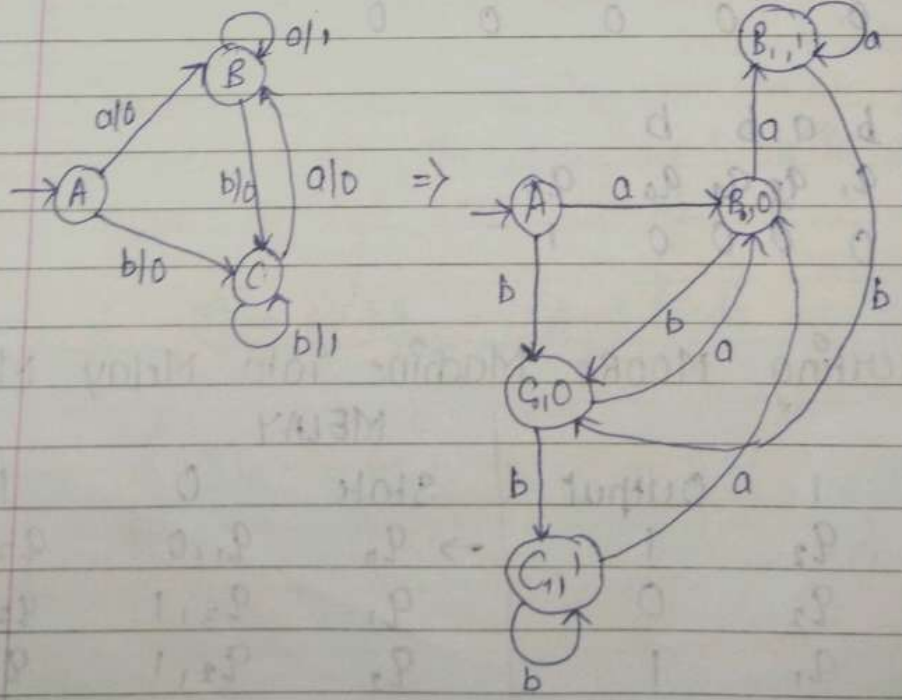
diagram explanation also



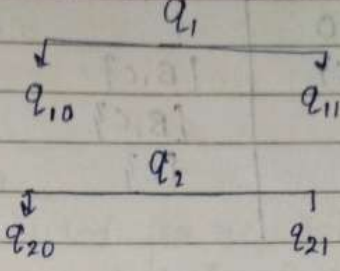


Converting Melay Machine into Moore Machine





State	a	b
→ q_0	$q_3, 0$	$q_1, 1$
q_1	$q_0, 1$	$q_3, 0$
q_2	$q_2, 1$	$q_2, 0$
q_3	$q_1, 0$	$q_0, 1$



State	a	b	Output
→ q_0	q_3 0	q_{11} 1	1
q_{10}	q_0 1	q_3 0	0
q_{11}	q_0 1	q_3 0	1
q_{20}	q_{21} 1	q_{20} 0	0
q_{21}	q_{21} 1	q_{20} 0	1
q_3	q_{10} 0	q_0 1	0

Epsilon NFA $\{Q, \Sigma, q_0, \delta, F\}$

(Empty symbol)

$\delta : Q \times \Sigma \cup \epsilon \rightarrow 2^Q$

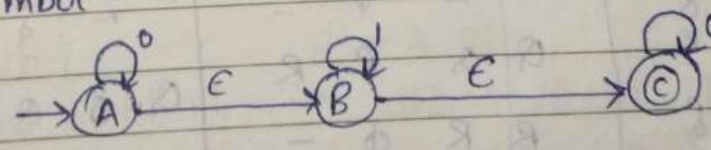
Every state on ϵ goes to itself.

Ques. Convert the following ϵ-NFA to its equivalent NFA

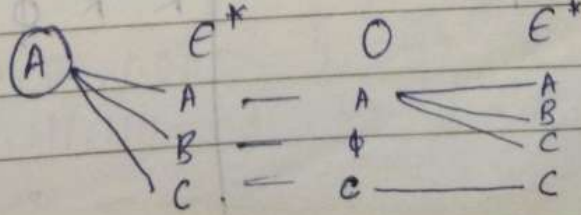
→ State ϵ^* input ϵ^*

ϵ - closure (ϵ^*) - All the states that can be reached from a particular state only by seeing the ϵ symbol

Q.



→



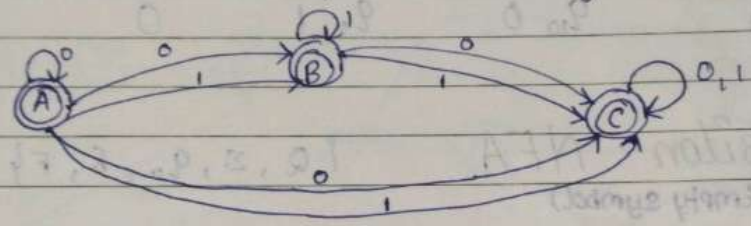
	0	1
→ (A)	{A,B,C}	{B,C}
(B)	{C}	{B,C}
(C)	{C}	{C}

(A)	ϵ^*	1	ϵ^*
	A	ϕ	
	B	B	B,C
	C	C	C

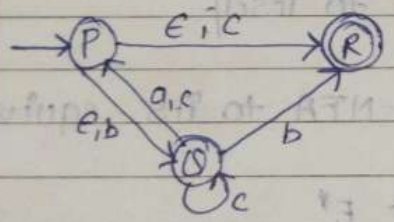
(B)	ϵ^*	1	ϵ^*
	B	B	B,C
	C	C	C

(B)	ϵ^*	0	ϵ^*
	B	ϕ	
	C	C	C

* Final state will be any state that can ~~only~~ reach the final state only by seeing ϵ.



Que.. Convert the following ϵ-NFA to its equivalent NFA

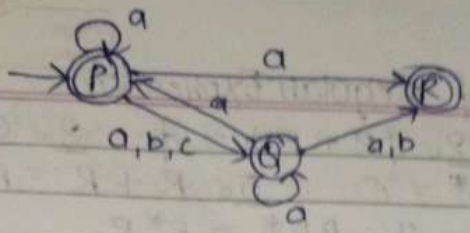


	a	b	c
→ P	{P,Q,R}	{Q,R}	{Q,R}
Q	{P,Q,R}	{R}	{Q}
R	ϕ	ϕ	ϕ

	E^*	a	E^*
P	P	ϕ	ϕ
	Q	P	P
	R	Q	Q
			R
Q	Q	P	P
			Q
			R
R	R	ϕ	–

	E^*	b	E^*
P	P	Q	Q
	Q	R	R
	R	ϕ	–
Q	Q	R	R
R	R	ϕ	–

	E^*	c	E^*
P	P	R	R
	Q	Q	Q
	R	–	
Q	Q	Q	Q
R	R	ϕ	–



Regular Expression

Regular expressions are used for representing certain sets of strings in an algebric fashion.

- Any terminal symbol i.e. symbols $\in \Sigma$ including $\wedge$ and ϕ are regular expressions.
- The Union of two regular expressions is also a regular expression.
- The concatenation of two regular expression is also a regular expression.
- The iteration (or closure) of a regular expression is also a regular expression.
- The regular expression over Σ are precisely those obtained recursively by the application of the above rules once or several times.

Que.. Describe the following sets as Regular Expression

1) $\{0,1,2\}$ → 0 or 1 or 2 , $R = 0+1+2$
2) $\{\wedge, ab\}$ → $R = \wedge$ ab
3) $\{abb, a, b, bba\}$ → abb or a or b or bba , $R = abb + a + b + bba$
4) $\{\wedge, 0, 00, 000 \ldots\}$ → $R = 0^*$, closure of 0
5) $\{1, 11, 111, 1111 \ldots\}$ → $R = 1^+$, closure of 1

Excluding empty symbol

Scribble
Page :
Date : / /

Identities of Regular Expression

1. $\phi + R = R$
2. $\phi R + R\phi = \phi$
3. $\epsilon R = R\epsilon = R$
4. $\epsilon^* = \epsilon$ and $\phi^* = \epsilon$
5. $R + R = R$
6. $R^* R^* = R^*$
7. $RR^* = R^* R$
8. $(R^*)^* = R^*$
9. $\epsilon + RR^* = \epsilon + R^* R = R^*$
10. $(PQ)^* P = P(QP)^*$
11. $(P+Q)^* = (P^* Q^*)^*$
    $= (P^* + Q^*)^*$
12. $(P+Q)R = PR + QR$
    and $R(P+Q) = RP + PQ$

ARDEN'S THEOREM

If P and Q are two Regular Expressions over Σ, and if P does not contain ϵ, then the following equation in R given by

$R = Q + RP$ has a unique solution

i.e. $R = QP^*$

→ $R = Q + RP \quad - (1)$

$= Q + QP^* P \quad \{R = QP^*\}$

$= Q(\epsilon + P^* P)$

$= QP^*$, proved

→ $R = Q + RP$

$= Q + (Q+RP)P = Q + QP + RP^2$

$= Q + QP + (Q+RP)P^2$

$= Q + QP + QP^2 + RP^3$

$= Q + QP + QP^2 + \dots QP^n + RP^{n+1}$

$\{R = QP^*\} = Q + QP + QP^2 + \dots QP^n + [QP^*]P^{n+1}$

$= Q[\epsilon + P + P^2 + \dots P^n + P^* P^{n+1}]$

$R = QP^*$ { unique solution proved }

Que:- Prove that $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$ is equal to $0^*1\,(0+10^*1)^*$

→ $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$

$= (1+00^*1)\,[\epsilon + (0+10^*1)^*(0+10^*1)]$ · $\{\epsilon + R^*R = R^*\}$

$= (1+00^*1)\,(0+10^*1)^*$ $\{\epsilon \cdot R = R\}$

$= (\epsilon \cdot 1+00^*1)\,(0+10^*1)^*$

$= 1\,(\epsilon + 00^*)\,(0+10^*1)^*$

$= 1\,0^*\,(0+10^*1)^* = 0^*1\,(0+10^*1)^*$ =

Designing Regular Expressions ...

Design Regular Expression for the following languages over {a, b}

1. Language accepting strings of length exactly 2
2. Language accepting strings of length atleast 2
3. Language accepting strings of length atmost 2

1. $L_1 = \{aa, ab, ba, bb\}$

   $R = aa + ab + ba + bb = a(a+b) + b(a+b)$

   $= (a+b)(a+b)$

2. $L_2 = \{aa, ab, ba, bb, aaa - -\}$

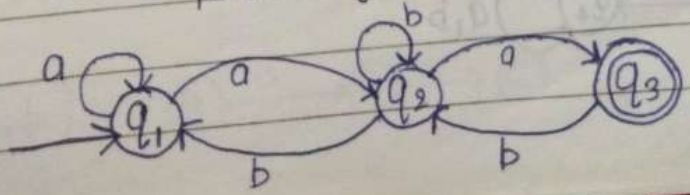
   $R = (a+b)(a+b)(a+b)^*$

3. $L_3 = \{\epsilon, a, b, aa, ab, ba, bb\}$

   $R = \epsilon + a + b + aa + ab + ba + bb$

   $= (\epsilon + a + b)(\epsilon + a + b)$

Que:- Find the Regular Expression for the following NFA.



→ Write the equations for each of the states of NFA and then put all eq.ⁿ to the eq.ⁿ of final state.

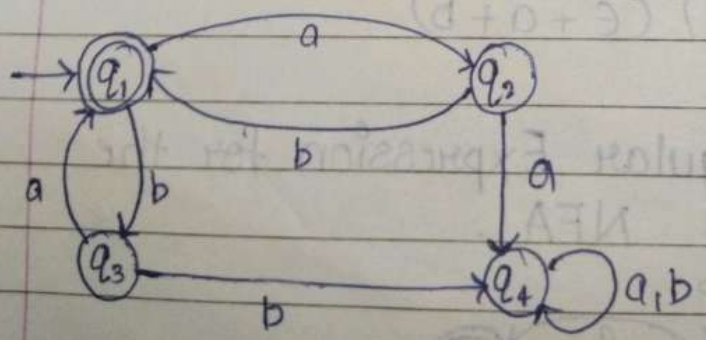
$q_3 = q_2 a$
$q_2 = q_1 a + q_2 b + q_3 b$
$q_1 = q_1 a + \epsilon + q_2 b$

$q_3 = q_1 aa + q_2 ba + q_3 ba$
$q_2 = q_1 a + q_2 b + q_2 ab$
$q_2 = q_1 a + q_2 (b + ab)$ {Arden's theorem $R = Q + RP$}
$q_2 = q_1 a (b+ab)^*$ {$R = QP^*$}

$q_1 = \epsilon + q_1 a + (q_1 a (b+ab)^*) b$
$q_1 = \epsilon + q_1 (a + a(b+ab)^*) b$ {$R = Q + RP$}
$q_1 = \epsilon ((a + a(b+ab)^*) b)^*$ {$\epsilon \cdot R = R$}
$q_1 = (a + a(b+ab)^* b)^*$

$q_3 = q_2 a$
$= q_1 a (b+ab)^* a$
$R = ((a + a(b+ab)^* b)^* (a(b+ab)^* a))$

Ques Find the Regular Expression for the following DFA.



$q_1 = \epsilon + q_2 b + q_3 a$

$q_2 = q_1 a$

$q_3 = q_1 b$

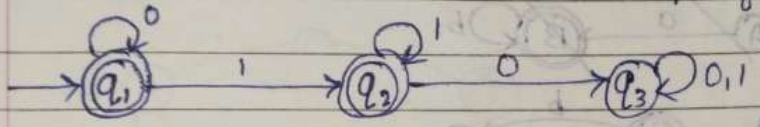
$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$

$q_1 = \epsilon + q_1 ab + q_1 ba$

$= \epsilon + q_1 (ab + ba) = \epsilon (ab + ba)^*$

$R = (ab + ba)^*$

Que.. Find the Regular Expression for the following DFA (Multiple final states)



final states ka union just

$q_1 = \epsilon + q_1 0$

$q_2 = q_1 1 + q_2 1$

$q_3 = q_2 0 + q_3 0 + q_3 1$

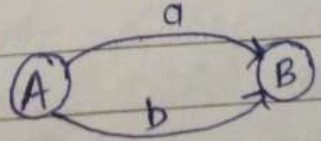
$q_1 = \epsilon 0^* = 0^*$, $q_2 = 0^* 1 + q_2 1$

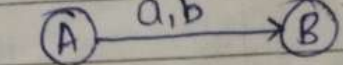
$q_2 = 0^* 1 1^*$

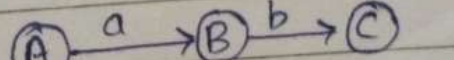
$R = q_1 \cup q_2 = 0^* + 0^* 1 1^*$.

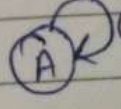
$= 0^* (\epsilon + 11^*) = 0^* 1^*$

Conversion of Regular Expression to Finite Automata.

$(a+b)$ 

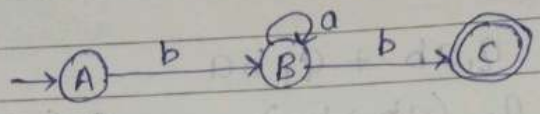
or 

$(a.b)$ 

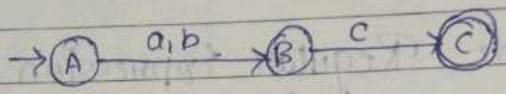
, 0^* 

Que.. Convert the following Regular Expression to their equivalent Finite Automata.

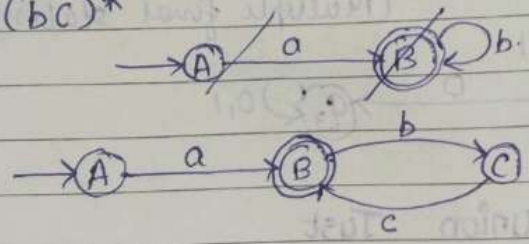
1. $b\,a^*\,b$ ⇒ bb, bab, baab, baaab ....



2. $(a+b)c$



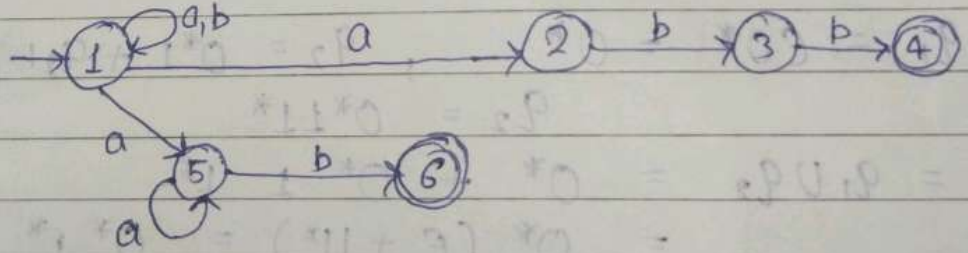
3. $a(bc)^*$



Que.. Regular Ex. to Finite Automata

*

1. $(a|b)^*(abb|a^+b)$ {start from start, + = | = OR}

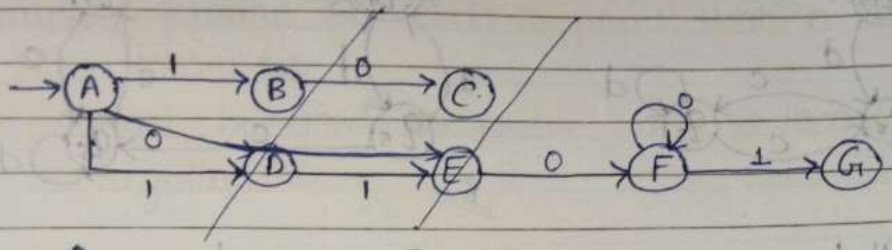


$a^+ = \{a, aa, aaa \ldots\}$

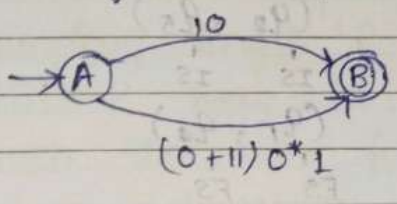
$a^* = \{\epsilon, a, aa, aaa \ldots\}$

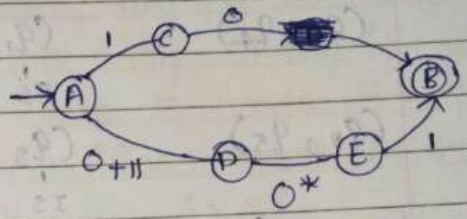
Scribble
Page :
Date : / /

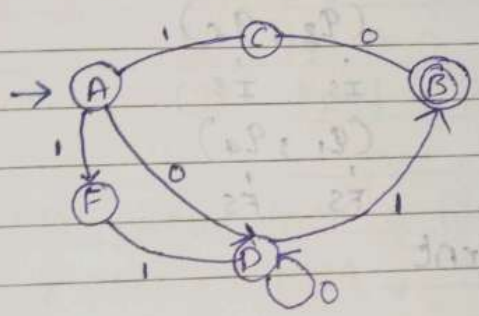
Q. $10 + (0 + 11)\,0^*1$



Creating step wise







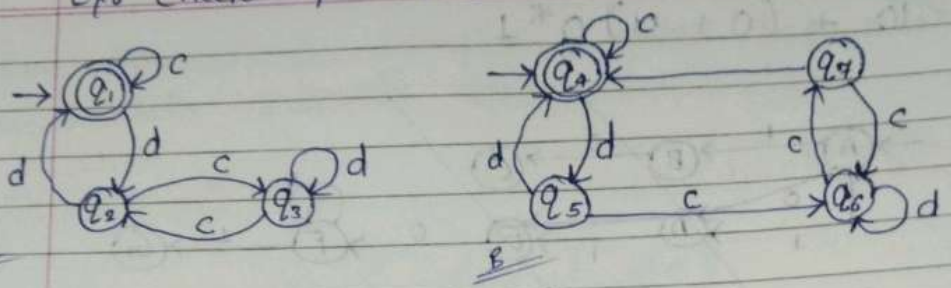
Equivalence of two finite Automata

STEPS TO IDENTIFY EQUIVALENCE

1. For any pair of states $\{q_i, q_j\}$ the transition for input $a \in \Sigma$ is identified by $\{q_a, q_b\}$ where $\delta\{q_i, a\} = q_a$ and $\delta\{q_j, a\} = q_b$ The two automata are not equivalent if for a pair $\{q_a, q_b\}$ one is INTERMEDIATE state and the other is FINAL state.

2. If Initial state of one automation, then in second automation also Initial state must be Final state for them to be equivalent.

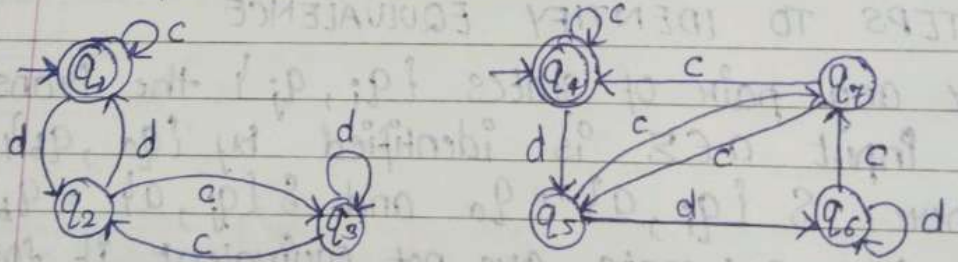
Ex. check for 2 automatas.



Check: 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_2, q_7)	(q_3, q_6) IS IS	(q_1, q_4) FS FS

→ A and B are equivalent

Que.. Find out whether the following automata are equivalent or not.



Check: States	c	d
(q_1, q_4)	(q_1, q_4) F F	(q_2, q_5) I I
(q_2, q_5)	(q_3, q_7) I I	(q_1, q_6) F I ✗

→ not equivalent.

Pumping Lemma for regular languages

» Pumping lemma is used to prove that a language is NOT REGULAR

» It cannot be used to prove that a language is 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:

(1) $xy^iz \in A$ for every $i \geq 0$

(2) $|y| > 0$

(3) $|xy| \leq P$

To prove that a language is not Regular using PUMPING LEMMA, STEPS:-

(We Prove using contradiction)

- Assume that A is Regular
- It has to have a pumping length (say P)
- All strings longer then P can be Pumped $|S| \geq P$
- Now find a string 'S' in A such that $|S| \geq P$
- Divide S into xyz
- Show that $xy^iz \notin A$ for some i
- Then consider all ways that S can be divided into xyz.
- Show that none of these can satisfy all the 3 Pumping conditions at the same time
- S can't be pumped == CONTRADICTION.

Scribble
Page :
Date : / /

Ex. Using Pumping lemma prove that the language $A = \{a^n b^n \mid n \geq 0\}$ is not Regular

→ Assuming that A is Regular
Pumping length = P
$S = a^P b^P$
x y z
assuming P = 7
then S = aaaaaaabbbbbbb

Case 1: y is in 'a' part

aaaaaaabbbbbbb
x y z

Case 2: y is in 'b' part

aaaaaaabbbbbbb
x y z

Case 3: y is in 'a' as well as 'b' part

aaaaaaabbbbbbb
x y z

× for case 1 :

$xy^i z \Rightarrow xy^2 z$
= aa aaaaaaaaabbbbbbb
$\#_a \neq 7_b$

× for case 2 :
$xy^i z \Rightarrow xy^2 z$
aaaaaaabb bbbbbbbb
$7_a \neq 11_b$

× For case 3 :

$xy^i x \Rightarrow xy^2z$

= aaaaaa aabb aabb bbbbbb

~~not~~ not follows $a^n b^n$ pattern though $a_n = b_n$.

	$\lvert xy \rvert \leq P$	Case 2 :	case 3 :
Case 1 :	$6 \leq 7$	$13 \nleq 7$	$9 \nleq 7$.

Hence conditions doesn't satisfies

hence A is not Regular.

Que.. Using Pumping lemma prove that the language $A = \{yy \mid y \in \{0,1\}^*\}$ is not Regular.

→ Assuming 'A to be Regular

Pumping length = P

$S = 0^P 1\, 0^P 1$ (P no of 0 followed by 1 yy satisfy)

x y z

let P = 7

String = 00000001 00000001

Case 1 : 00 0000001 0000001

x y z

$xy^i z \Rightarrow xy^2z$ = 00 000000001 00000001

$10_0 \neq 7_0$, $\notin A$

~~also~~ $\lvert xy \rvert = 6 \leq 7$

∴ A is not Regular.

Automaton is a machine which performs a range of functions acc. to a predetermined set of coded instructions.

Scribble
Page:
Date: / /

Regular Grammer

Noam chomsky gave a Mathematical model of Grammer which is effective for writing Computer Languages.

The 4 Types of Grammer acc. to chomsky :

GRAMMER TYPE	GRAMMER ACCEPTED	Language Accepted	Automaton
TYPE-0	Unrestricted	Recursively Enumerable	Turing Machine
TYPE-1	context sensitive	Context sensitive	linear bounded
TYPE-2	Context Free	Context Free	Push down
TYPE-3	Regular Grammer	Regular	Finite state

Grammer

A Grammer 'G' can be formally described using 4 tuples as $G = (V, T, S, P)$ where,

V = Set of Variables or Non-terminal symbols
T = Set of Terminal symbols
S = Start Symbol
P = Production rules for Terminals and Non-terminals

A production rules has the form $\alpha \rightarrow \beta$ where, α and β are strings on VUT and atleast on symbol of α belongs to V.

Ex. $G = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

Here,

$V = \{S, A, B\}$ $\quad T = \{a, b\}$ $\quad S = S$

$P = S \to AB, A \to a, B \to b$

$S \to AB \to aB \to ab$ //

Regular Grammer (2 types)

Right Linear Grammer

if all productions are of the form

$A \to xB$

$A \to x$

where $A, B \in V$ and $x \in T$

Ex. $S \to abS \mid b$

Left Linear Grammer

if all productions are of the form

$A \to Bx$

$A \to x$

$A, B \in V$ and $x \in T$

$S \to Sbb \mid b$

Derivations from a Grammer

The set of all strings that can be derived from a Grammer is said to be the Languge generated from that grammer.

Ex. Consider the Grammer $G1 = (\{S, A\}, \{a, b\}, S, \{S \to aAb, aA \to aaAb, A \to \epsilon\}$

$\to S \to aAb$ [by $S \to aAb$]

$\to aaAbb$ [by $aA \to aaAb$]

$\to aaaAbbb$ [by $aA \to aaAb$]

$L(G1) \to aaabbb$ [by $A \to \epsilon$]

$L(G1)$ → Grammer generated by G1

Ex. G_3 $(\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to aA \mid a, B \to bB \mid b\})$

$S \to AB$, $S \to AB$

$\to ab$ $\to aAbB \to aabb$

$L(G_3) = \{ab, a^2b^2, a^2b, ab^2 \ldots\}$

$= \{a^m b^n \mid m \geq 0 \text{ and } n \geq 0\}$

Context Free Languages

In formal language theory, a context free language is a language generated by some context free grammar.

The set of all CFL is identical to the set of languages accepted by Pushdown Automata.

Context free Grammar is defined by 4 tuples as $G = \{V, \Sigma, S, P\}$ where,

V = Set of variables or Non-terminal symbols

Σ = Set of Terminal Symbols

S = Start Symbol

P = Production Rule

CFG has production Rule of the form :-

$A \to \alpha$

where, $\alpha = \{V \cup E\}^*$ and $A \in V$

Ex. For generating a language that generates equal no. of a's and b's in the form $a^n b^n$, the context free grammar will be defined as

$G = \{(S, A), (a, b), (S \to aAb, A \to aAb | \epsilon)\}$

→ $S \to aAb$

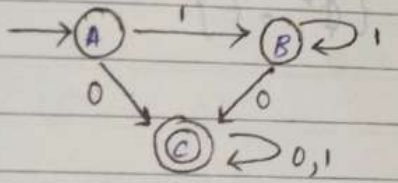
$\to aaAbb$ (by $A \to aAb$)

$\to aaaAbbb$ (")

$\to aaabbb$ (by $A \to \epsilon$)

$\to a^3 b^3 \Rightarrow \ldots \Rightarrow a^n b^n$

Que.. 2013 Consider the DFA 'A' given below:



Which of the following are false?

1. Complement of L(A) is context free True
2. $L(A) = L((11^*0 + 0)(0+1)^* 0^* 1^*)$ True
3. For the language accepted by A, A is the minimal DFA False
4. A accepts all strings over {0,1} of length at least 2. False (0 accept krta hai)

(A) 1 and 3 only (B) 2 and 4 only (C) 2 and 3 only

✓ (D) 3 and 4 only.

2. $C = A0 + B0 + C0 + C1$

$B = A1 + B1 = \epsilon 1 + B1 = 1 + B1$ $\{R = QP^*\}$

$A = \epsilon$, $B = 11^*$

anything multiply with ε remains same

$\underbrace{C}_{R} = \underbrace{0 + 11^*0}_{Q} + \underbrace{C}_{R}\underbrace{(0+1)}_{P}$, $C = 0 + 11^*0(0+1)^*$

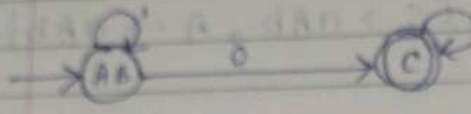
$= 11^*0 + 0(0+1)^*$

Minimizing DFA

8.

0 Eq.ⁿ = {A,B} {C} → same

1 Eq.ⁿ = {A,B} {C}



Que.. 2013 Consider the language $L_1 = \phi$ and $L_2 = \{a\}$. Which one of the following represents $L_1 L_2^* \cup L_1^*$?

(A) $\{\epsilon\}$ (B) ϕ (C) a^* (D) $\{\epsilon, a\}$

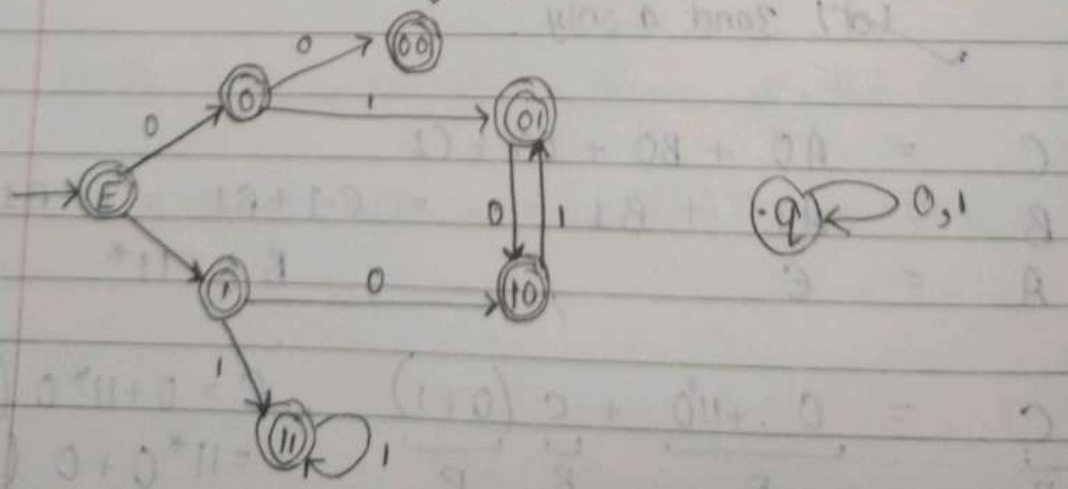
$\phi a^* \cup L_1^*$ $\quad \{\phi R = R\phi = \phi\}$

$\phi \cup \phi^*$ $\quad \{\phi^* = \epsilon\}$

$\phi \cup \epsilon$

$= \epsilon$

Que.. 2019 Consider the set of strings on $\{0,1\}$ in which, every substring of 3 symbols has at most two zeros. For example, 001110 and 011001 are in the language, but 100010 is not. All strings of length less than 3 are also in the language. A partially completed DFA that accepts this language is shown below.



Non-terminal ← upper case

The missing arcs in the DFA are:

(A) A (B) B (C) C ~~(D)~~ D ✓

(A)

	00	01	10	11	q
00	1	0			
01				1	
10	0				
11			0		

(B)

	00	01	10	11	q
00		0			1
01		1			
10				0	
11		0			

(C)

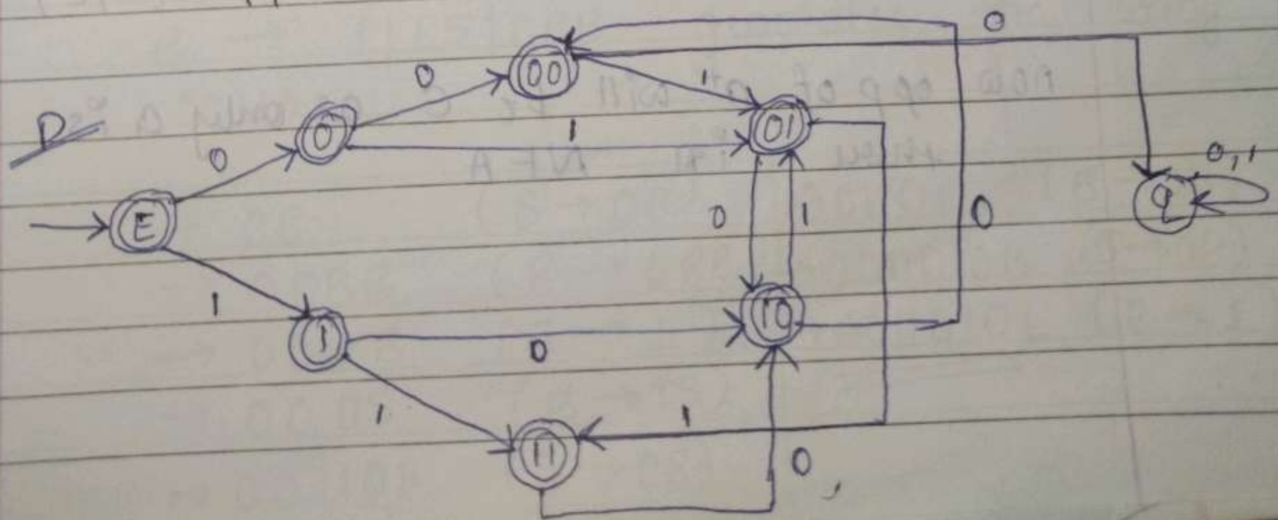
	00	01	10	11	q
00		1			0
01		1			
10			0		
11		0			

~~(D)~~ ✓

	00	01	10	11	q
00		1			0
01				1	
10	0				
11			0		

in sb tables se check karne ka kisi se '000' to ni ho rha

D



Que.. 2009 Which one of the following languages over the alphabet {0,1} is described by the regular Expression : $(0+1)^*\,0\,(0+1)^*\,0\,(0+1)^*$?

✗ (A) The set of all strings containing the substring 00.

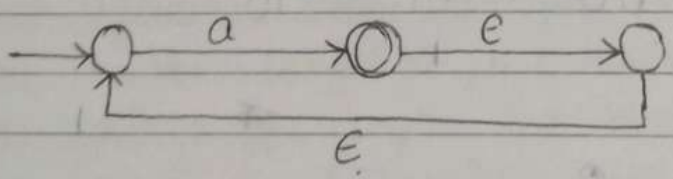
✗ (B) The set of all strings containing at most two 0's.

✓ (C) The set of all strings containing at least two 0's.

✗ (D) The set of all strings that begin and end with either 0 or 1.

$\underbrace{(0+1)^*}_{0 \text{ or } 1 \text{ or none}}\ \underbrace{0}_{\text{sure}}\ \underbrace{(0+1)^*}_{0 \text{ or } 1 \text{ or more}}\ \underbrace{0}_{\text{sure}}\ \underbrace{(0+1)^*}_{0 \text{ or } 1 \text{ or more}}$

Que.. 2012 What is the complement of the language accepted by the NFA shown below ?



a^+

a ✓
aa ✓
a... ✓
at least one a → a^+

(A) ϕ ✓(B) $\{\epsilon\}$ (C) a^* (D) $\{a, \epsilon\}$

now opp of a^+ will be ϵ as only a is there in NFA

Que. Given the language L = {ab, aa, baa}, which of 2012 the following strings are in L*?

1) abaabaaabaa
2) aaaabaaaa
3) baaaaabaaaab
4) baaaaabaa

(A) 1, 2 and 3
(B) 2, 3 and 4
(C) 1, 2 and 4
(D) 1, 3 and 4

Method to find whether a string belongs to a Grammer or not

- Start with the start symbol and choose the closest production that matches to the given string.
- Replace the variable with its most appropriate production. Repeat the process until the string is generated or until no other productions are left.

Eg. Verify whether the Grammer $S \to 0B|1A$, $A \to 0|0S|1AA$, $B \to 1|1S|0BB$ generates the string 00110101.

$\Rightarrow$ $S \to 0B$ $(S \to 0B)$
$\to 00BB$ $(B \to 0BB)$
$\to 001B$ $(B \to 1)$
$\to 0011S$ $(B \to 1S)$
$\to 00110B$ $(S \to 0B)$
$\to 001101S$ $(B \to 1S)$
$\to 0011010B$ $(S \to 0B)$
$\to 00110101$ $(B \to 1)$

Eg. Verify whether the Grammer $S \to aAb$, $A \to aAb \mid \wedge$ generates the string aabbb

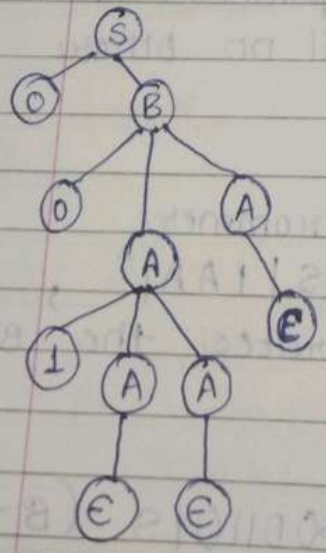
$S \to aAb$ $(S \to aAb)$
$\to aaAbb$ $(A \to aAb)$
$\to aabb$ $(A \to \wedge)$
$\to aaaAbbb$ $(A \to aAb)$
$\to aaabbb$ $(A \to \wedge)$

∴ → the string aabbb doesn't belongs.

Derivation Tree

A derivation Tree or Parse Tree is an ordered rooted tree that graphically represents the semantic information of strings derived from a context free grammer.

Eg. For the Grammer $G_1 = (V, T, P, S)$ where
$S \to 0B$, $A \to 1AA \mid \epsilon$, $B \to 0AA$



Root Vertex : Must be labelled by the start symbol

Vertex : Labelled by Non-terminal symbols

Leaves : Labelled by Terminal symbols or ϵ

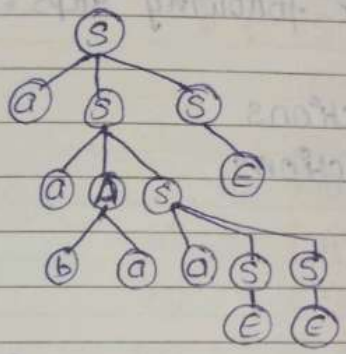
Left Derivation TREE

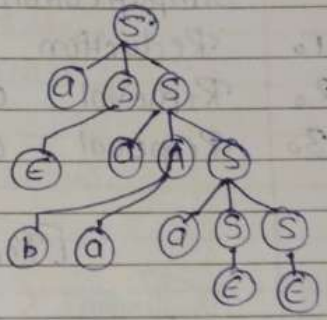
It is obtained by applying production to the left most variable in each step.

Right Derivation Tree

It is obtained by applying production to the rightmost variable in each step.

Ex. For generating the string aabaa from the Grammer: $S \to aAS \mid aSS \mid \epsilon$,
$A \to SbA \mid ba$





Ambiguous Grammer

A grammer is said to be ambiguous if there exists two or more derivation tree for a string w (that means two or more left derivation trees)

Ex. $G = \{\{S\}, \{a+b, +, *\}, P, S\}$ where P consists of $S \to S+S \mid S*S \mid a \mid b$.
The string $a+a*b$ can be generated as:

1 way $S \to S+S$
$\to a+S*S \to a+a*S \to a+a*b$

2 way $S \to S*S$
$\to S+S*S \to a+S*S \to a+a*S$
$\to a+a*b$

Simplification Of Context free Grammer
~~[Reduction of CFG]~~

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 productions and unit productions. Elimination of these productions and symbols is called simplification of CFG.

Simplification consists of the following steps:

1. Reduction of CFG
2. Removal of unit Productions
3. Removal of null productions.

[Reduction of CFG]

Phase 1 : Derivation of an equivalent grammer 'G', from the CFG, G, such that each variable derives some terminal string

DERIVATION PROCEDURE:

Step 1 : Include all symbols W_1, that derives some terminal and initialize $i = 1$

Step 2 : Include Symbols W_{i+1}, that derives W_i

Step 3. Increment i and repeat Step 2, until $W_{i+1} = W_i$

Step 4. Include all production rules that have W_i in it

Phase 2 : Derivation of an equivalent grammer 'G'', from the CFG, G', such that each symbol appears in a sentential form

Page: Scribble
Date: / /

DERIVATION PROCEDURE :

Step 1 : Include the start symbol in y_1 and initialize $i = 1$

Step 2 : Include all symbols y_{i+1}, that can be derived from y_i and include all production rules that have been applied.

Step 3 : Increment i and repeat Step 2, until $y_{i+1} = y_i$

Example :

Find a reduced grammer equivalent to the grammer G, having production rules.

P : $S \to AC|B$, $A \to a$,
$C \to c|BC$, $E \to aA|e$

Phase 1 $T_0 = \{a, c, e\}$ → derive this

$W_1 = \{A, C, E\}$

$W_2 = \{S, e, A, C, E, S\}$ → same

$W_3 = \{A, C, E, S\}$

$G' = \{(A, C, E, S), \{a, c, e\}, P, (S)\}$

(NT: A, C, E, S; T: a, c, e)

P : $S \to AC$, $A \to a$, $C \to c$, $E \to aA|e$

Phase 2 $Y_1 = \{S\}$

$Y_2 = \{S, A, C\}$

$Y_3 = \{S, A, C, a, c\}$

$Y_4 = \{S, A, C, a, c\}$

$G'' = \{(A, C, S), \{a, c\}, P, \{S\}\}$

P : $S \to AC$, $A \to a$, $C \to c$

reduced form

[Removal of Unit Productions]

Any Production Rule of the form $A \to B$ where $A, B \in$ Non-terminals is called unit production

Procedure of Removal

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

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

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

Ex.. Remove Unit productions from the grammar whose production rule is given by

$P: S \to XY$, $X \to a$, $Y \to Z|b$,
$Z \to M$, $M \to N$, $N \to a$

⇒ Unit Productions, $Y \to Z$, $Z \to M$, $M \to N$

1) Since $N \to a$, we add $M \to a$

$P: S \to XY$, $X \to a$, $Y \to Z|b$, $Z \to M$, $M \to a$, $N \to a$

2) Since $M \to a$, we add $Z \to a$

$P: S \to XY$, $X \to a$, $Y \to Z|b$, $Z \to a$, $M \to a$, $N \to a$

3) Since $Z \to a$, we add $Y \to a$

$P: S \to XY$, $X \to a$, $Y \to a|b$, $Z \to a$, $M \to a$, $N \to a$

(Z → a, M → a, N → a : unreachable symbols.)

Remove unreachable symbols.

$P: S \to XY$, $X \to a$, $Y \to a|b$

[Removal of Null Productions]

In a CFG, a Non-terminal symbol 'A' is a nullable variable if there is a production $A \to \epsilon$ or there is a derivation that starts at 'A' and leads to ϵ. (Like $A \to \ldots \to \epsilon$)

PROCEDURE FOR REMOVAL

Step1: To remove $A \to \epsilon$, look for all productions whose right side contains A.

Step2: Replace each occurrences of 'A' in each of these productions with ϵ.

Step3: Add the resultant productions to the grammer.

Ex. Remove Null Productions from the following Grammer.

$S \to ABAC$, $A \to aA \mid \epsilon$, $B \to bB \mid \epsilon$, $C \to c$

Null Productions : $A \to \epsilon$, $B \to \epsilon$

1) To eliminate $A \to \epsilon$

$S \to ABAC \mid \to ABC \mid BAC \mid BC$

$A \to aA \quad \to A \to a$,

P: $S \to ABAC \mid ABC \mid BAC \mid BC$

$A \to aA \mid a$, $B \to bB \mid \epsilon$, $C \to \epsilon$

2) To eliminate $B \to \epsilon$

~~$S \to ABAC \to AAC$~~

$S \to AAC \mid AC \mid C$, $B \to b$

P: $S \to ABAC \mid ABC \mid BAC \mid BC \mid AAC \mid AC \mid C$,

$A \to aA \mid a$, $B \to bB \mid b$,

$C \to c$

CHOMSKY NORMAL FORM

In chomsky Normal form (CNF), we have a restriction on the length of RHS; which is; elements in RHS should either be two variables or a Terminal.

A CFG is in CHomsky Normal form if the productions are in the following forms:

$A \rightarrow a$

$A \rightarrow BC$

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

STEPS TO CONVERT A GIVEN CFG TO CHOMSKY NORMAL FORM:

Step1: If the start symbol S occurs on some right side, create a new start symbol S' and a new production $S' \rightarrow S$.

Step2: Remove Null Productions.

Step3: Remove Unit Productions.

Step4: Replace each production $A \rightarrow B_1 \ldots B_n$ where $n > 2$, with $A \rightarrow B_1 C$ where $C \rightarrow B_2 \ldots B_n$. Repeat this step for all productions having two or more symbols on the right side.

Step5: If the right side of any Production is in the form $A \rightarrow aB$ where 'a' is a terminal and A and B are non-terminals, then the production is replaced by $A \rightarrow XB$ and $X \rightarrow a$. Repeat this step for every Production which is of the form $A \rightarrow aB$.

Que. Convert the following CFG to CNF :
P : S → ASA | aB , A → B|S , B → b|ϵ

1) Since S appear in RHS , we add a new state S' and S' → S is added to the production
P : S' → S , S → ASA|aB , A → B|S , B → b|ϵ

2) Remove the Null Productions : B → ϵ and A → ϵ :
After Removing B → ϵ : P : S' → S , S → ASA|aB|a, A → B|S|ϵ , B → b
After Removing A → ϵ : P : S' → S , S → ASA|aB|a|AS|SA|S, A → B|S , B → b

3) Remove the Unit Productions : S → S , S' → S , A → B and A → S :
After Removing S → S : P : S' → S , A → B|S , S → ASA|aB|a|AS|SA , B → b
After Removing S' → S : P : S' → ASA|aB|a|AS|SA, S → ASA|aB|a|AS|SA, A → B|S, B → b
After Removing A → B : P : S' → ASA|aB|a|AS|SA , S → ASA|aB|a|AS|SA , A → b|S , B → b
After Removing A → S : P : S' → ASA|aB|a|AS|SA, S → ASA|aB|a|AS|SA , A → b|ASA|aB|a|AS|SA, B → b

4) Now find out the production that has more than TWO variables in RHS : S' → ASA , S → ASA and A → ASA

after removing these, we get:

P : S' → AX | aB | a | AS | SA
S → AX | aB | a | AS | SA
A → b | AX | aB | a | AS | SA
B → b
X → SA

5) Now change the productions S' → aB, S → aB and A → aB

finally we get, P : S' → AX | YB | a | AS | SA ,
S → AX | YB | a | AS | SA ,
A → b | AX | YB | a | AS | SA ,
B → b , X → SA , Y → a

Greibach Normal form

A CFG is in greibach normal form if the productions are in the following forms:

A → b
A → bC1C2 .... Cn

where, A, C1, ..., Cn are Non-terminals and b is a Terminal.

STEPS to Convert a given CFG to GNF :

Step 1: Check if the given CFG has any Unit Productions or Null Productions and remove if there are any.

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

Step 3: Change the names of the Non-terminal symbols into some Ai in ascending order of i.

Ex. $S \to CA \mid BB$
$B \to b \mid SB$
$C \to b$
$A \to a$

Replace S with A_1
C with A_2
A with A_3
B with A_4

we get, $A_1 \to A_2 A_3 \mid A_4 A_4$
$A_4 \to b \mid A_1 A_4$
$A_2 \to b$
$A_3 \to a$

Step4: Alter the rules so that the Non-Terminals are in ascending order, such that, if the production is of the form $A_i \to A_j x$, then, $i < j$ and should never be $i \geq j$

$A_4 \to b \mid A_1 A_4$
$A_4 \to b \mid A_2 A_3 A_4 \mid A_4 A_4 A_4$
$A_4 \to b \mid b A_3 A_4 \mid A_4 A_4 A_4$

$b A_3 A_4$ is GNF (A → bC₁C₂...)
$A_4 A_4 A_4$ left recursion ($i = j$)

Step5: Remove Left Recursion
→ Introduce a new variable ~~that~~ [one with new and one without]

$Z \to A_4 A_4 Z \mid A_4 A_4$
$A_4 \to b \mid b A_3 A_4 \mid bZ \mid b A_3 A_4 Z$
GNF

now,
$A_1 \to A_2 A_3 \mid A_4 A_4$
$A_4 \to b \mid b A_3 A_4 \mid bZ \mid b A_3 A_4 Z$
$Z \to A_4 A_4 \mid A_4 A_4 Z$
$A_2 \to b$ $A_3 \to a$

then, $A_1 \to bA_3 \mid bA_4 \mid bA_3 A_4 A_4 \mid bZA_4 \mid bA_3 A_4 Z A_4$

$A_4 \to b \mid bA_3A_4 \mid bX \mid bA_3A_4Z$

~~$X \to b \mid A_4 t$~~

$X \to bA_4X \mid bA_3A_4X \mid bXA_4X \mid bA_3A_4XA_4Z$

also

$X \to bA_4 \mid bA_3A_4A_4 \mid bXA_4 \mid bA_3A_4ZA_4$

Pumping Lemma for context free languages

Pumping lemma for context free languages is used to prove that a language is NOT context free.

If A is a Context Free language, then, A has a Pumping length 'p' such that any string 's', where $|s| \geq p$ may be divided into 5 pieces $s = uvxyz$ such that the following conditions must be true:

(1) uv^ixy^iz is in A for every $i \geq 0$

(2) $|vy| > 0$

(3) $|vxy| \leq p$

⇒ To prove that a language is not Context free using Pumping lemma (for CFL), we use Contradiction

- Assume that A is Context Free
- It has to have a Pumping length (say p)
- All strings longer than p can be pumped $|s| \geq p$
- Now find a string 's' in A such that $|s| \geq p$
- Divide s into uvxyz
- Show that $uv^ixy^iz \notin A$ for some i
- Then consider the ways that s can be divided into uvxyz
- Show that none of these can satisfy all the 3 pumping conditions at the same time

can't be pumped == Contradiction

Que. Show that $L = \{0^N b^N c^N \mid N > 0\}$ is not Context Free

$S = a^p b^p c^p$

Ex. $P = 4$ so $S = a^4 b^4 c^4$

case 1 : V and Y each contain only one type of symbol

aaaa bbbb cccc

u v x y z

$\Rightarrow uv^i x y^i z$ must belong to L

let $i = 2$, uv^2xy^2z

aaaaaabbbbcccccc $\Rightarrow a^6 b^4 c^5 \notin L$

case 2 : Either V or Y has more than one kind of symbols

aaaabbbbcccc

u v x y z

$i = 2$ for uv^ixy^iz:

aaaabbaabbbbbbcccc $\notin L$

As case 1 & 2 's output doesn't belong to L hence L is not a Context free language.

Que. Show that $L = \{WW \mid W \in \{0,1\}^*\}$ isn't context free

$S = 0^p 1^p 0^p 1^p$, $P = 5$

case 1 : 00000 11111 00000 11111

u vxy z

vxy doesn't straddle a boundary

$i = 2$, uv^2xy^2z

$0000011111100000 11111 \Rightarrow 0^5 1^7 0^5 1^5 \notin L$

case 2 : vxy straddles the first boundary

00000 11111 00000 11111

u vxy z

$uv^2xy^2z \Rightarrow 00000001111111000001111$

$0^7 1^7 0^5 1^5 \notin L$

Case 3 : vxy straddles the third boundary

00000 11111 000 00 111 11

u vxy z

$uv^2xy^2z \Rightarrow 00000\ 11111\ 000\ 00001111111$

$0^5 1^5 0^7 1^7 \notin L$

The output of δ is finite set of pairs (P, Y) where:
P is a new state
Y is a string of stack symbols that replaces x

Ex. Y = G, stack is popped
Y = X, stack is unchanged
Y = YZ, x replaced by z and Y is pushed onto the stack

Case 4: UXY straddles the midpoint

0000011 | 1110 | 000 | 1111
u | vxy | z

$U V^2 x y^2 z \Rightarrow 0000011\,1111\,0000\,0001111$

$0^{5,7} 0^{7,3} \notin L$

Condition 1 doesn't satisfy by any case hence L isn't context free.

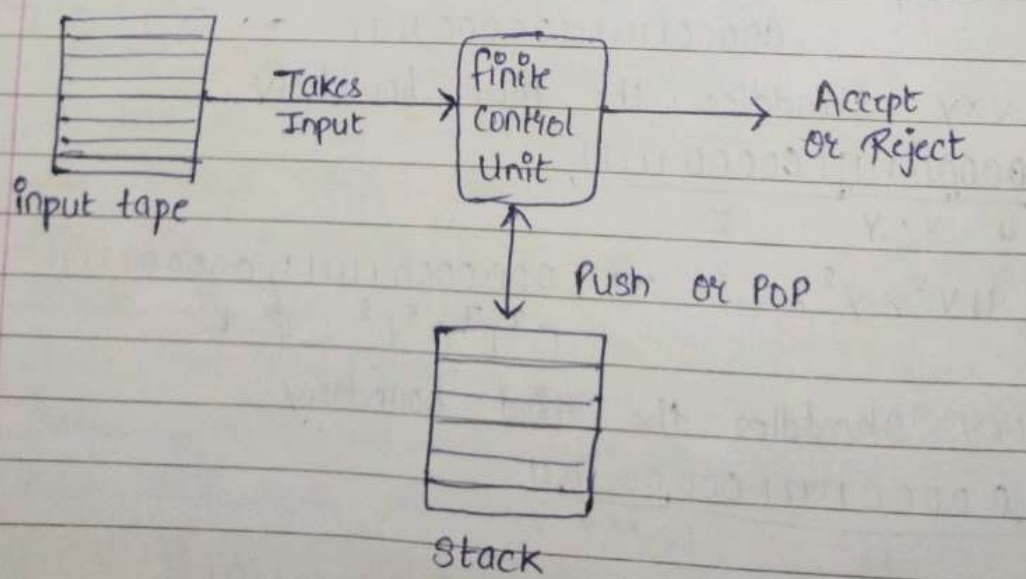
Pushdown Automata (PDA)

It is a way to implement a context free Grammar in a similar way we design finite Automata for regular Grammer

- It is more powerful than Finite state Machine (FSM)
- FSM has a very limited memory but PDA has more memory
- PDA = FSM + stack

A PDA has 3 components:

1. An input tape
2. A finite Control Unit
3. A stack with infinite size



$\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \rightarrow Q \times \Gamma^*$

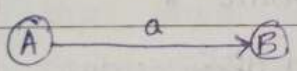
A pushdown Automata is defined using 7 Tuples

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

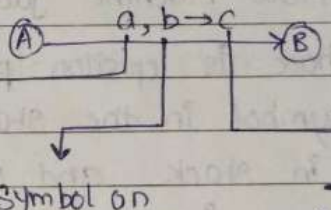
- Q ← A finite set of states
- Σ ← A finite set of Input Symbols
- Γ ← A finite stack Alphabets
- δ ← The transition function
- q_0 ← The start state
- z_0 ← The start stack Symbol
- F ← The set of final / Accepting States.

* The transition function takes as args a triple $\delta(q, a, X)$
  - q is a state in Q.
  - a is either an Input Symbol in Σ or $a = \epsilon$ (empty)
  - X is a stack Symbol, that is a member of Γ.

FSM representation



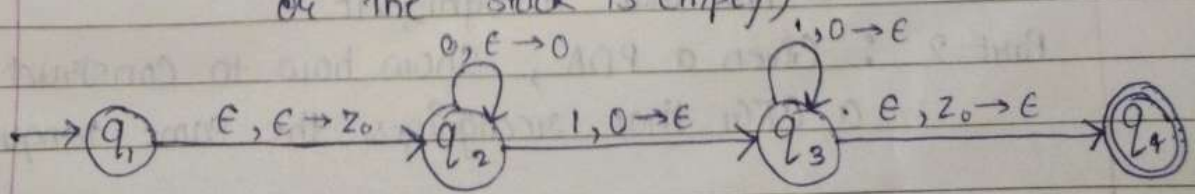
PDA representation



ϵ means the stack is neither read nor popped

Ex. Construct a PDA that accepts $L = \{0^n 1^n \mid n \geq 0\}$

(either it has to reach the final state or the stack is empty)



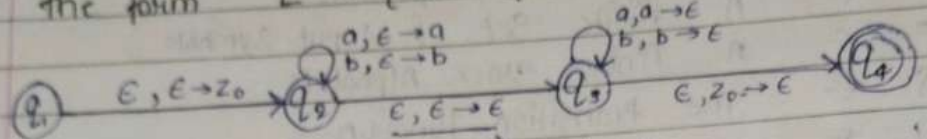
(it is non-deterministic PDA)

+ ← can be empty
* ← epsilon allowed
R ← reverse

From $\epsilon \rightarrow$ (Push)
to $\rightarrow \epsilon$ (Pop)

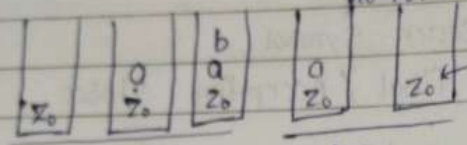
Scribble
Page:
Date: / /

Ques. Construct a PDA that accepts Even Palindromes of the form $L = \{ WW^R \mid W = (a+b)^+ \}$



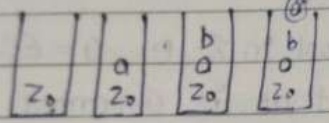
(nor push) ← midpoint
no pop

Ex. abba



Push
POP
STACK

Ex. abab



no a so will remain in q_3
so not accepted

How Machine judges the middle point ?

there is epsilon present before and after every symbol in the stack is our assuption of Proceeding in stack and whenever the popping operation starts in epsilon path without failing the conditions than it is considered the middle point And eventually reaches the final state unlike other paths & the stack is empty.

★ Equivalence of CFGs and PDA ★

Theorem : A language is Context Free iff some Pushdown Automata recognizes it.

Proof : Part 1 : Given a ~~GF~~ CFG, show how to Construct a PDA that recognizes it

Part 2 : Given a PDA, show how to construct a CFG that recognizes the same language.

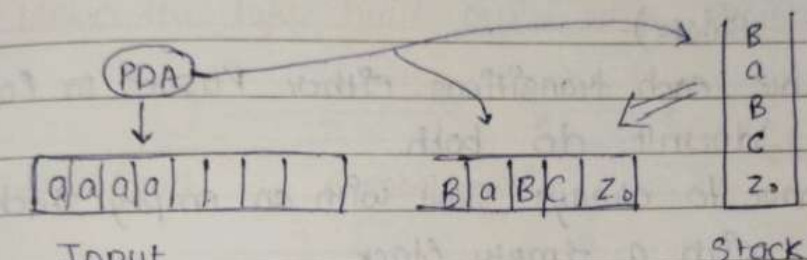
you can only push one element to a stack at a time

Page: Scribble
Date: / /

Given a grammar $S \to BS \mid A$
$A \to 0A \mid \epsilon$
$B \to BB1 \mid 2$ find or build a PDA

left most derivation $\to S \to BS \to BBIS \to 2B1S$
$\to 221S \to 221A \to 221\epsilon \to 221$

General form : $\underbrace{aaaa}_{\text{Terminals}}\underbrace{BaBC}_{\text{Rest}}$

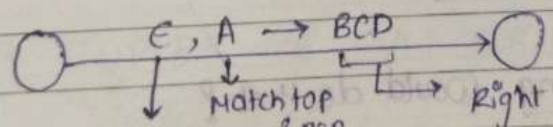


PART 1

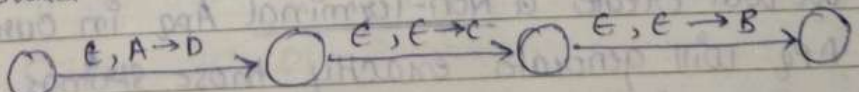
Steps :

1. Match Stack Top to a Rule
2. Pop Stack
3. Push Right Hand side of Rule onto stack

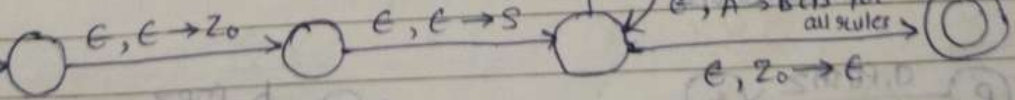
Rule : $A \to BCD$



for A→BCD:



final PDA



* There will be a NON-Terminal for every pair of states.

PART 2

There will be a Non-Terminal for every pair of states.

For Simplifying the PDA :

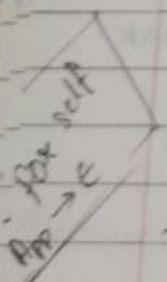
1. The PDA should have only one final / accept state.
2. The PDA should empty its stack before accepting (Create a new start state q_0 which pushes Z_0 to the stack)
3. Make sure each transitions either Pushes or Pops but doesn't do both
4. We have to always start with an empty stack & finish with a empty stack

(no underflow, or overflow condition must occur)

For building CFG

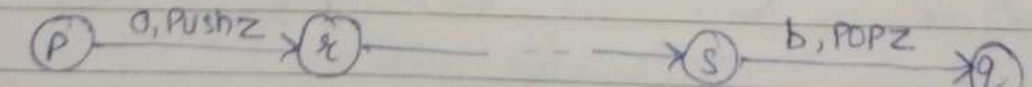
* Consider two states P and q in the PDA
Could we go from P to q without stack Underflow & maintaining an empty stack at the beginning and end? OR if something were already on the stack they would never be touched.

What string would do that?



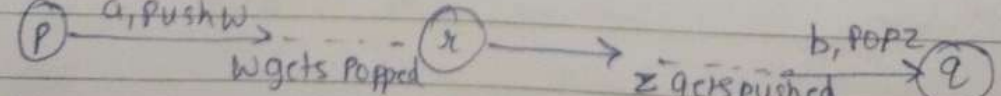
We will create a Non-Terminal A_{pq} in our Grammar. A_{pq} will generate exactly those strings that will take us from p to q maintaining all the above stack conditions.

Case1



$A_{pq} \rightarrow a A_{rs} b$

Case2



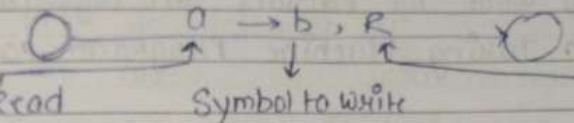
$A_{pq} \rightarrow A_{pr} A_{rq}$

(Recursively enumerable languages)

Turing Machine

Tape Alphabets : $\Sigma = \{0, 1, a, b, X, Z\}$ (input strings)
The blank ($\sqcup$) is a special symbol used to fill the tape

Operations on the Tape :
- Read / scan symbol below the tape head
- update / write a symbol below the tape head
- Move the tape head one step LEFT
- Move the tape head one step RIGHT



If you don't want to update the cell, Just write the SAME SYMBOL

- There are 2 final states — The Reject state, The accept state.
- computation can either
  - → HALT and Accept → HALT and Reject
  - → LOOP (the machine fails to HALT)

★ 7 TUPLES of A TURING MACHINE
$(Q, \Sigma, \Gamma, \delta, q_0, b, F)$

Q ← Non-empty set of states
Σ ← Non-empty set of symbols
Γ ← Non-empty set of tape symbols
δ ← Transition function defined as
$Q \times \Sigma \rightarrow \Gamma \times (R/L) \times Q$
q_0 ← Initial State
b ← Blank State
F ← Set of Final States

Turing Machine's behaviour is determined by a finite state Machine.

Production Rule $\delta(q_0, a) \rightarrow (q_1, Y, R)$

Scribble
Page:
Date: / /

Turing's Thesis

- states that any computation that can be carried out by mechanical means can be performed by some Turing Machine.

ARGUMENTS FOR ACCEPTING THIS THESIS ARE:

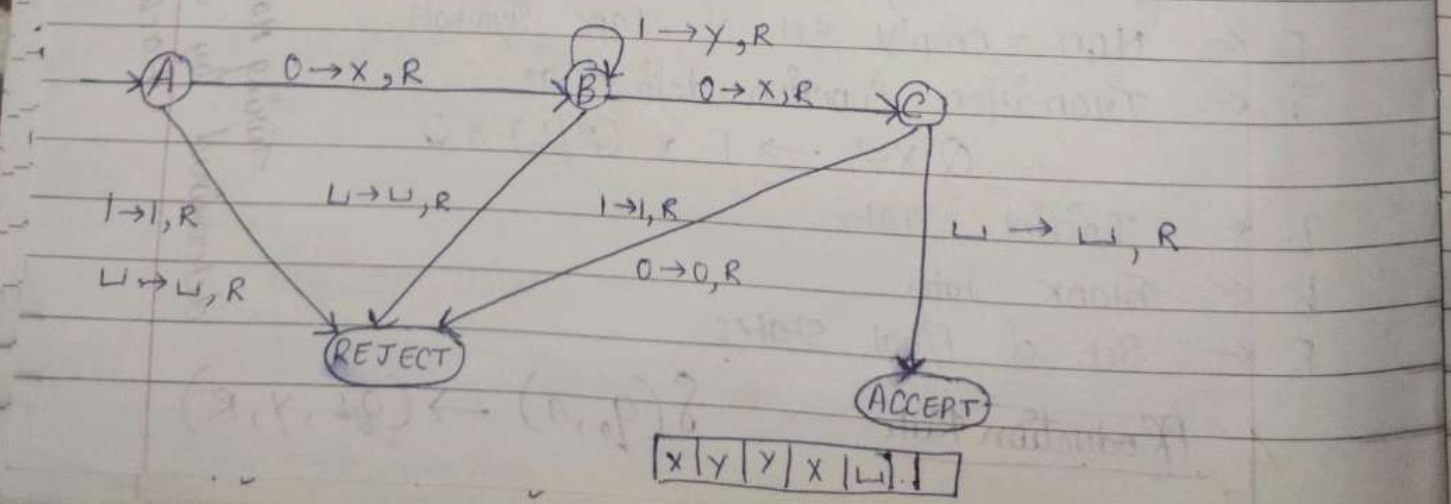
i. Anything that can be done on existing digital comp. can also be done by Turing Machine
ii. No one has yet been able to suggest a problem solvable by what we consider an algorithm, for which a Turing Machine program cannot be written

*** A language L and Σ is said to be Recursively Enumerable if there exists a Turing Machine that excepts it.

(A turing Machine is Mathematical Model of computation describing an abstract Machine or hypothetical device that manipulates symbols on an infinite strip of tape acc. to a table of rules).

(These are used to study the properties of Algorithms & to determine what problems can & can't be solved by computers (time + memory))

Que. Design a Turing Machine which recognizes the language $L = 01^*0$



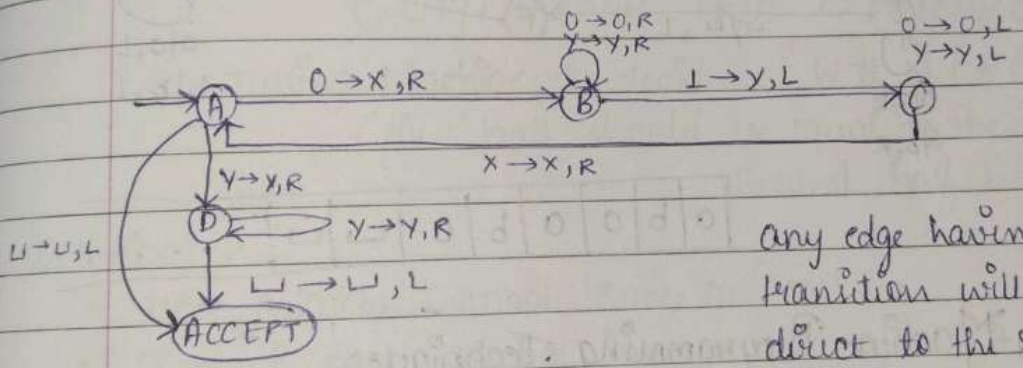
we have to keep account of 0 and 1 no. ∴ not done by FSM

Page: Date: / / Scribble

Que.. Design a Turing Machine which recognizes the language $L = 0^N 1^N$

ALGORITHM

- Change "0" to "x"
- Move RIGHT to FIRST "1"
  If NONE : REJECT
- 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



any edge having missing transition will automatically direct to the reject state.

The CHURCH-TURING Thesis

Alnzo Church - LAMBDA CALCULUS {Computable}
Allen Turing - TURING Machine

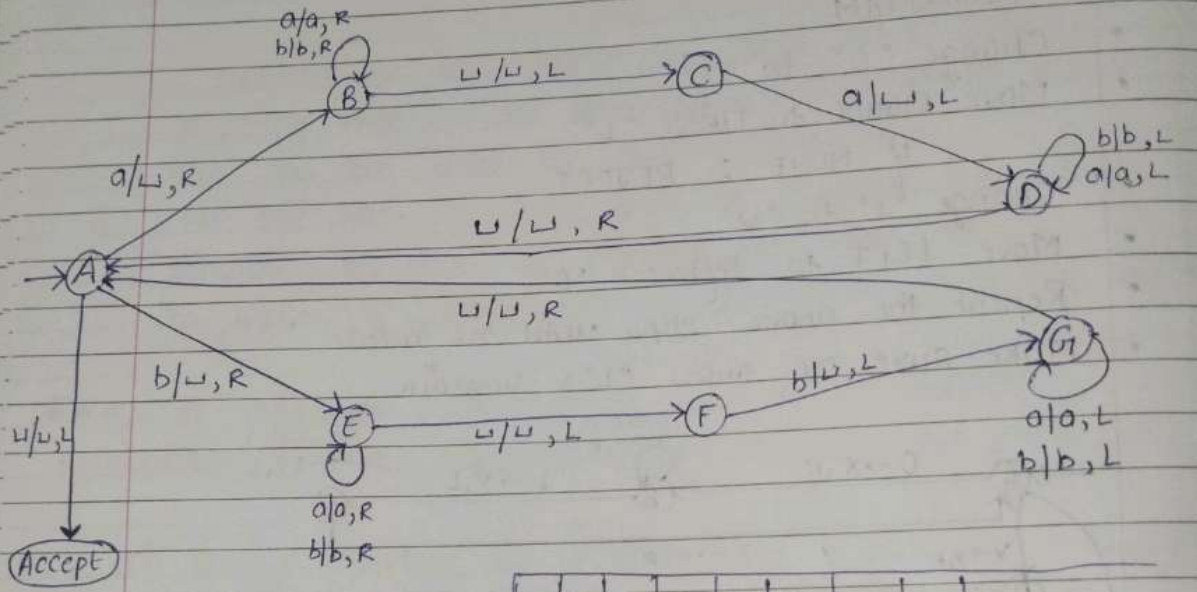
Several variations of Turing Machine: All variations are equivalent in computing capability

- One Tape or many
- Infinite on both ends
- Alphabets only {0,1} or more?
- Can the head also stay in the same place?
- Allow Non-Determinism

Scribble
Page :
Date : / /

Ex.. Design a Turing Machine that accepts Even Palindrome over the alphabet

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

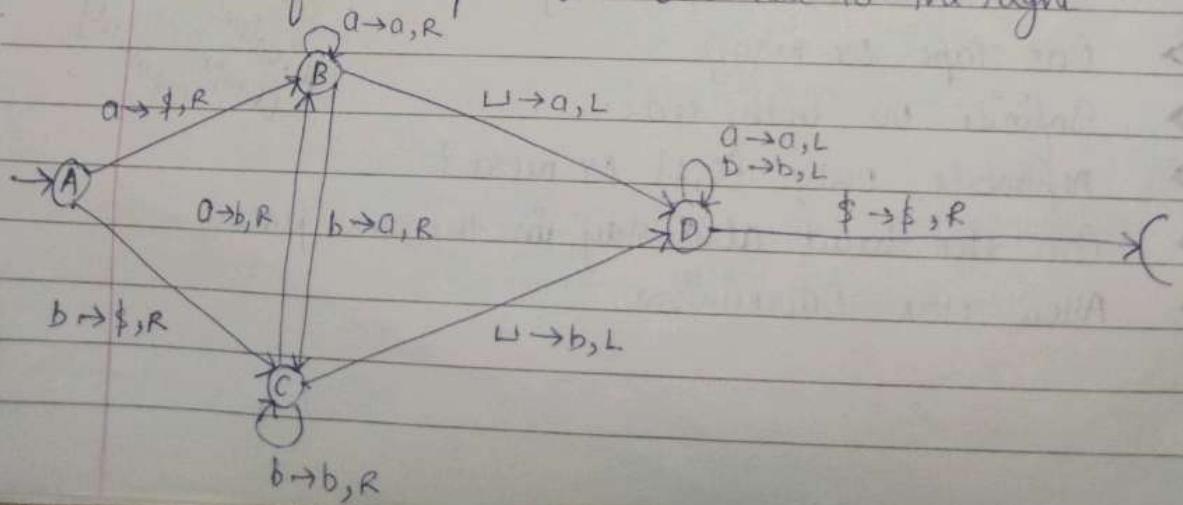


a	b	a	a	b	a	␣	␣	. . . . .

TURING Machine Programming Techniques ...

Problem : How can we recognize the left end of the Tape of a Turing Machine ?

Solution : Put a special Symbol '$' on the left end of the Tape and shift the input over one cell to the right



Ques. Build a Turing Machine to recognize the language $0^N 1^N 0^N$

COMPARING TWO STRINGS

A Turing Machine to decide $\{w\#w \mid w \in \{a,b,c\}^*\}$
(first half should be equal to the second half)

Solution :

- Use a new symbol such as 'X'
- Replace each symbol into an 'X' after it has been examined (here we looses the original string)

So, Replace each unique symbol with another unique symbol instead of replacing all with the same symbol.

Multitape TURING Machine

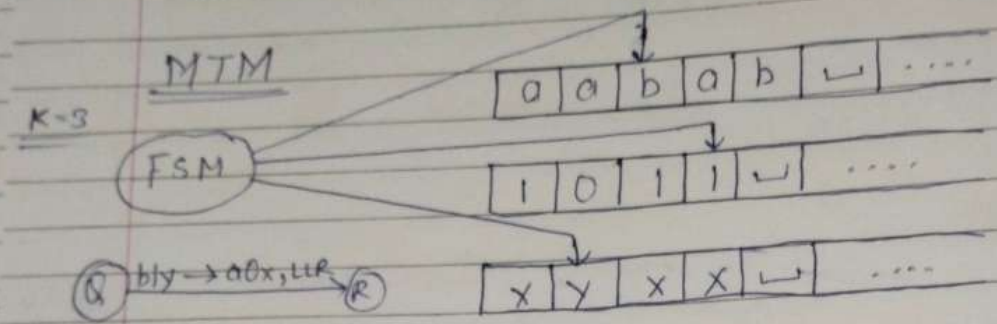
* Every Multitape turing Machine has a equivalent single tape Turing Machine

Proof : Given a MTM show how to build a STM :

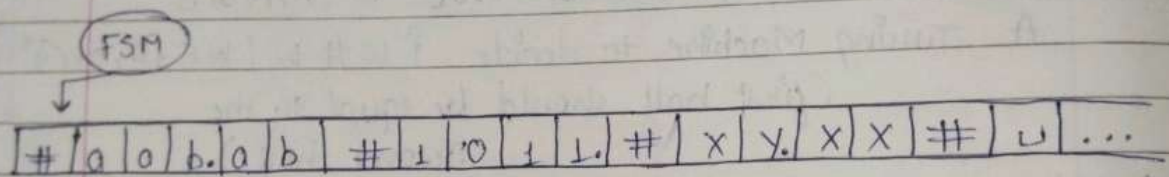
- Need to store all tapes on a single tape
  → show data representation
- Each tape has a tape head
  → Show how to store that info.

Scribble
Page :
Date : / /

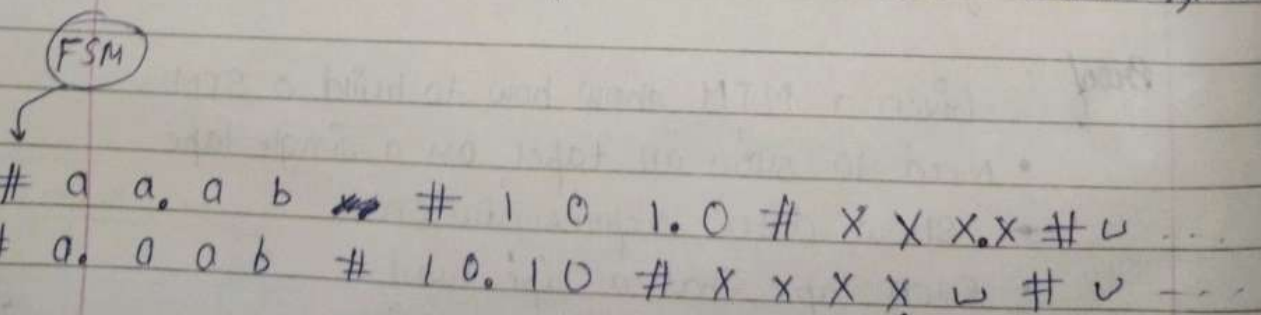
- Need to transform a move in the Multitape TM into one or moves in the single tape TM



STM



- Add dots to show where Head K is (technique of Marking)
- To simulate a transition from state Q, we must scan our Tape to see which symbols are under the K Tape Heads. (by scanning dots)
- Once we determine this and are ready to Make the transition, we must scan across the tape again to update the cells and move the dots.
- Whenever one head moves off the right ~~hand~~ end, we must shift our tape so we can insert a blank symbol (⊔)



P: Power set

Scribble
Page:
Date: / /

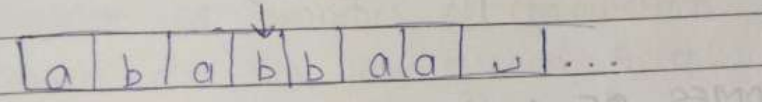
Non-Determinism in Turing Machine

Transition function here, (can go to many state instd of one)

$$\delta : Q \times \Sigma \rightarrow P\{\Gamma \times (R/L) \times Q\}$$

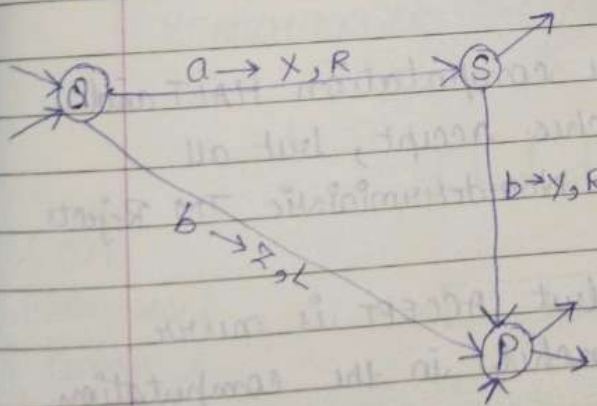
Configuration:

- A way to represent the entire state of a TM at a moment during computation
- A string which captures,
  - → The current state
  - → The current position of the Head
  - → The entire Tape contents

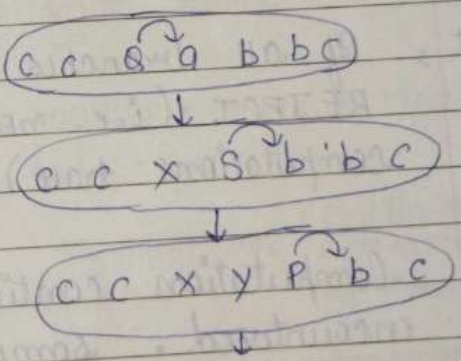


a b a q b a a (Representation)

Deterministic TM



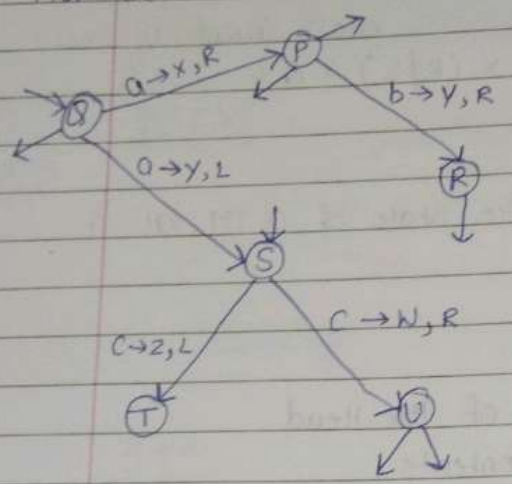
Configuration history



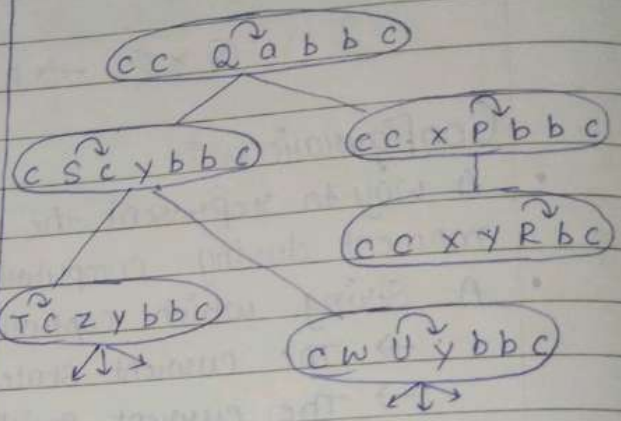
(linear chain)

With Non determinism, At each moment in the computation there can be more than one successor configuration.

Non-Deterministic TM



Computational History



(Tree)

OUTCOMES OF A NONDETERMINISTIC COMPUTATION

Accept, If any branch of the computation accepts, then the nondeterministic TM will Accept.

Reject, If all branches of the computation HALT and REJECT (i.e. no branches accept, but all computations halt) then Non-deterministic TM Rejects

Loop, Computation continues but ACCEPT is never encountered. Some branches in the computation history are infinite.

Page: Scribble
Date: / /

Every NON-DETERMINISTIC TM has an equivalent DETERMINISTIC TURING MACHINE

Proof :-

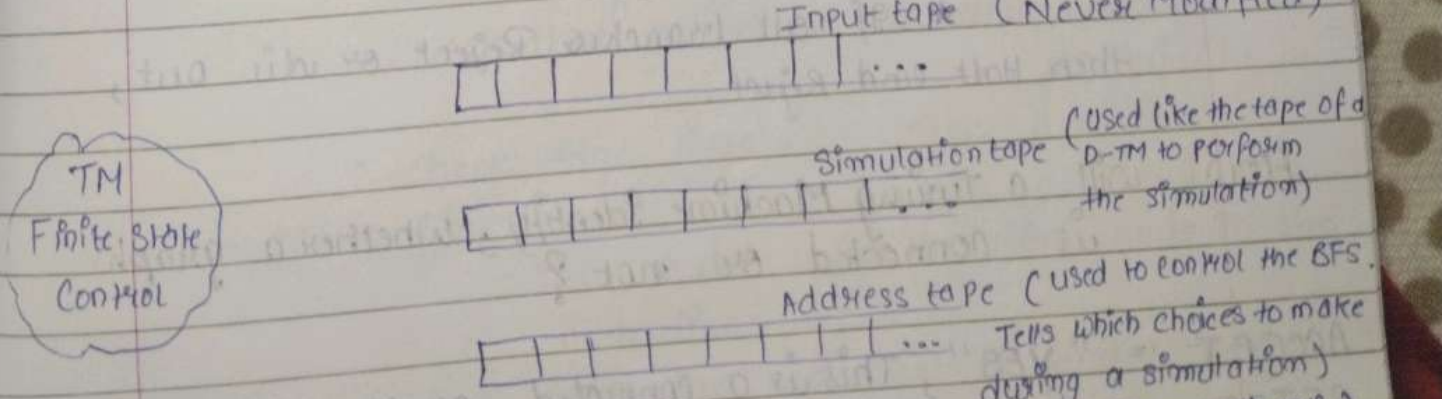
- Given a N-D TM (N), show how to construct an equivalent DET. TM (D)
- If N accepts on any branch, the D will accept
- If N halts on every branch without any accept, the D will Halt and Reject.

Approach :-

- Simulate N
- Simulate all branches of computation
- Search for any way N can Accept

- A path to any Node is given by a number
- Search the tree looking for ACCEPT

SEARCH ORDER :- Breadth first Search



ALGORITHM for constructing a deterministic TM for a Non-Deterministic TM.

Initially : TAPE 1 contains the Input
TAPE 2 and TAPE 3 are empty

- Copy TAPE 1 to TAPE 2
- Run the simulation
- Use TAPE 2 as "The Tape"
- When choices occur (i.e. when Non-deterministic branch points are encountered) consult TAPE 3
- TAPE 3 contains a path. Each no. tells which choice to make.
- Run the simulation all the way down the branch as far as the address/path goes (or the computation dies).
- Try the next branch
- Increment the address on TAPE 3
- REPEAT
  - * If ACCEPT is ever encountered, Halt and Accept.
  - * If all branches Reject or die out, then Halt and Reject.

How will a Turing Machine Identify, Whether a graph is connected or not?

ACCEPT = "YES", This is a connected graph
REJECT = "NO", This is not a connected graph / or this is not a valid Representation of a graph.
LOOP = This problem is decidable. Our TM will always halt.

Scribble
Page :
Date : / /

Algorithm for it (high level)

Select a Node and Mark it
REPEAT
    For each Node N
        if N is unmarked and there is an edge from N to an already marked node
    Then
        Mark Node N
    End
Until no more nodes can be marked
    For each Node N
        if N is unmarked
    Then REJECT
    End
ACCEPT

Implementation level Algorithm

- Check that input describes a valid graph
- Check Node list
    - Scan "(" followed by digits ...)
    - Check that all nodes are diff. i.e. no repeats
    - Check edge lists ... etc...
- Mark first Node
    - Place a dot under the first node in the node list
    - Scan the node list to find a node that isn't marked

etc...

Decidability & Undecidability

Recursive language :

- A language 'L' is said to be recursive if there exists a Turing Machine which will accept all the strings in 'L' and rejects all the strings not in 'L'.
- The Turing Machine will halt everytime and give an answer (accepted or rejected) for each and every string input.

Recursively Enumerable language :

- A language 'L' is said to be a recursively enumerable language if there exists a Turing Machine which will accept (and therefore halt) for all the input strings which are in 'L'.
- But may or may not halt for all input strings which are not in 'L'.

Decidable languages :

A language 'L' is decidable if it is a recursive language. All decidable languages are recursive languages and vice-versa

Partiall Decidable language :

A language 'L' is partially decidable if 'L' is a recursively enumerable language.

Undecidable Language :

- A language is undecidable if it is not decidable
- An undecidable language may sometimes be partially decidable but not decidable.
- If a language is not even partially decidable, then there exists no Turing machine for that language.

Universal Turing Machine

The language

$A_{TM} = \{ \langle M, w \rangle \mid M$ is a Turing Machine and M accepts $w \}$

is Turing Recognizable.

On an Universal TM

Input : M = the description of some TM
w = an input for M

Action :
- Simulate M
- Behave just like M would (may accept, reject or loop)

* U_{TM} is a recognizer but not a decider for A_{TM}

The halting problem is Undecidable..
(Church - Turing Thesis specifies that this can't have an Algo. to generalize the halting and so the only option remains is to run and check)
(as no TM)

Contradict your proof

Let's assume

H (P, I) (Program, Input)

- Halt
- Not Halt

C (X)
if { H (X, X) == Halt }
loop forever
else
Return

C (C) (running C on itself)

H (C, C) == Halt → Not Halt

H (C, C) = Not Halt → Halt

if C is not halting H tells it does so here occurs the Contradiction & vice-versa

∴ H doesn't exits

(undecidable)

The Post Correspondence Problem

It is an undecidable decision problem that was introduced by Emil Post in 1946

DOMINOS :

We need to find a sequence of dominos such that the top & bottom strings are the same

Ex. $\frac{B}{CA}$ $\frac{A}{AB}$ $\frac{CA}{A}$ $\frac{ABC}{C}$

Arrangement : $\frac{A}{AB}$ $\frac{B}{CA}$ $\frac{CA}{A}$ $\frac{A}{AB}$ $\frac{ABC}{C}$

ABCAAABC =
ABCAAABC =

Ex. $\frac{1}{111}$ $\frac{10111}{10}$ $\frac{10}{0}$

Arrangement : $\frac{10111}{10}$ $\frac{1}{111}$ $\frac{1}{111}$ $\frac{10}{0}$

Ex. $\frac{10}{101}$ $\frac{011}{11}$ $\frac{101}{011}$

1 1
1010
101101 X

1 2
10 011 X
101 11

1 3
10101
101011 ← one left always

1 3 3 ✓ loop

1 3 2 X

1 3 1 X

∴ all PCPs can't be solved
∴ no generalized algo.
can be created and so it is an undecidable prob..

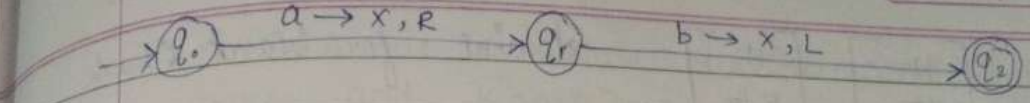
Proof : (BY REDUCTION)

Approach : Take a problem that is already proven to be undecidable and try to convert it to PCP.
If we can successfully convert it to an equivalent PCP then we prove that PCP is also undecidable

ACCEPTANCE PROBLEM OF A TURING MACHINE (This is an undecidable problem)
⇓
Convert this to PCP (called Modified PCP - MPCP)

$\left[\frac{\#}{\#}\right]$

Page: Scribble
Date: / /



$\Sigma = \{a, b\}$ $\quad \Gamma = \{a, b, X, B\}$ (B = blank $\sqcup$)

i/p : $w = aba$

Step 1 : $\boxed{a \mid b \mid a}$ (q_0 on a) $\quad q_0aba \Rightarrow \left[\frac{\#}{\# \; q_0 \; aba \; \#}\right]$

Step 2 : $\delta(q_0, a) = (q_1, X, R)$

$q_0 a = X q_1 \Rightarrow \left[\frac{q_0 \; a}{X \; q_1}\right]$ right can be empty

$X \in \Gamma$

Step 3 : $\delta(q_1, b) = (q_2, X, L)$

$Y \; q_1 b = q_2 Y X \Rightarrow \left[\frac{a \; q_1 \; b}{q_2 \; a \; X}\right]$

~~$a \; q_1 b$~~

$\Rightarrow \left[\frac{X \; q_1 \; b}{q_2 \; X \; X}\right] \Rightarrow \left[\frac{b \; q_1 \; b}{q_2 \; b \; X}\right]$

$\Rightarrow \left[\frac{B \; q_1 \; b}{q_2 \; B \; X}\right]$

not left could be empty :)

Step 4 : dominos for all possible tape symbols

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

$\left[\frac{a}{a}\right], \left[\frac{b}{b}\right], \left[\frac{X}{X}\right], \left[\frac{B}{B}\right]$

Step 5 : For all possible tape symbols after reaching the accepting state (q_2)

$\left[\frac{a \; q_2}{q_2}\right], \left[\frac{q_2 \; a}{q_2}\right], \left[\frac{b \; q_2}{q_2}\right], \left[\frac{q_2 \; b}{q_2}\right], \left[\frac{X \; q_2}{q_2}\right], \left[\frac{q_2 \; X}{q_2}\right], \left[\frac{B \; q_2}{q_2}\right], \left[\frac{q_2 \; B}{q_2}\right]$

Step 6 : $\left[\frac{\#}{\#}\right]\left[\frac{\#}{B\#}\right]$ domino for blank & hash

Step 7 : $\left[\frac{q_2\#\#}{\#}\right]$ for final

Solution $\left[\frac{\#}{\#q_0 b a \#}\right]\left[\frac{q_0 a}{x q_1}\right]\left[\frac{b}{b}\right]\left[\frac{a}{a}\right]\left[\frac{\#}{\#}\right]$

$\left[\frac{\#}{\#x q_1 b a \#}\right]\left[\frac{x q_1 b}{q_2 x x}\right]\left[\frac{a}{a}\right]\left[\frac{\#}{\#}\right]$

$\left[\frac{\#}{\#q_2 x x a \#}\right]\left[\frac{q_2 x}{q_2}\right]\left[\frac{x}{x}\right]\left[\frac{a}{a}\right]\left[\frac{\#}{\#}\right]$

$\left[\frac{\#}{\#q_2 x a \#}\right]\left[\frac{q_2 x}{q_2}\right]\left[\frac{a}{a}\right]\left[\frac{\#}{\#}\right]$

$\left[\frac{\#}{\#q_2 a \#}\right]\left[\frac{q_2 a}{q_2}\right]\left[\frac{\#}{\#}\right]$

$\left[\frac{\#}{\#q_2 \#}\right]\left[\frac{q_2\#\#}{\#}\right]$ ✓ ∴ PCP is undecidable

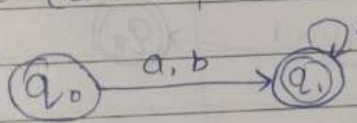
Successor
Constant
Projections
Composition
Primitive Recursion

1.a) A DFA is a state Machine consisting of states and transitions that can either accept or reject a finite string, which consits of a series of symbols, and compare it to a predefined language across a predetermined set of characters. We use circles to represent states, & directed arrows to represent transitions. Every state must have each symbol going outwards from the state or else it will not be defined as a DFA

Used for :

- Speech Recognisation, Pattern matching, compilers etc.

Ex $\{a^n \mid n \in (a+b)^*\}$



1.b) Finite automata are used to recognize patterns. It takes the string of symbol as an input and changes its state accordingly. When the desired symbol is found, then the transition occurs.
It has two states — Accept / Reject

A finite automation is a collection of 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where,

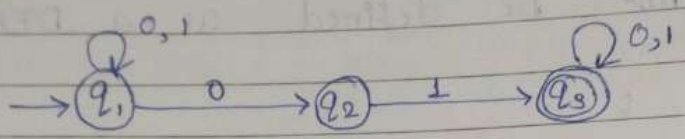
Q : finite set of states
Σ : finite set of the input symbol
q_0 : initial state
F : final state
δ : Transition function

DFA NFA

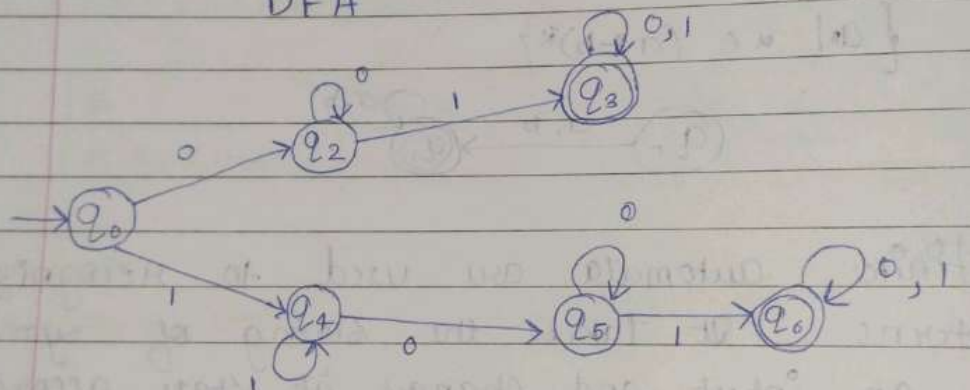
Finite automata can be represented by input tape and finite control.

- input tape ← linear tape having some no. of cells. Each input symbol is placed in each cell.
- Finite control ← decides next state by getting input from input tape (one input at a time).

1. c) NFA



DFA

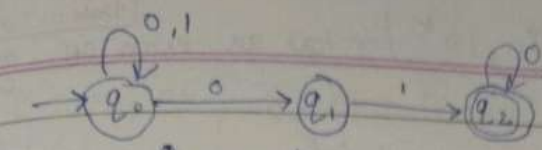


1. d) Regular expressions are a powerful tool for describing languages. They allow us to define finite patterns of strings or symbols. Each pattern corresponds to a set of strings, effectively serving as a name for that set. It consists of symbols, operators and paranthesis.

EX. $(0 \cup \epsilon)(1 \cup \epsilon)$ describes the language $\{01, 0, 1, \epsilon\}$.

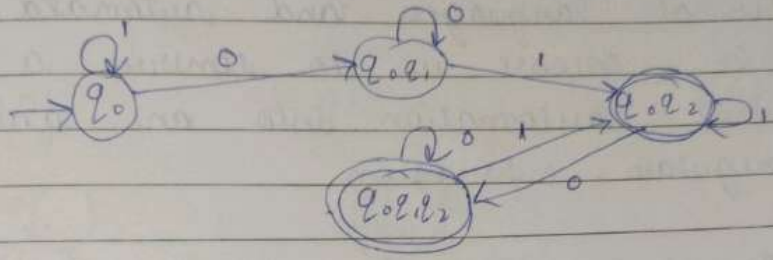
Scribble
Page:
Date: / /

2. a)



	0	1
q_0	q_0, q_1	q_0
q_1	ϕ	q_2
q_2	q_2	q_2

	0	1
q_0	$q_0 q_1$	q_0
$q_0 q_1$	$q_0 q_1$	$q_0 q_2$
$q_0 q_2$	$q_0 q_1 q_2$	$q_0 q_2$
$q_0 q_1 q_2$	$q_0 q_1 q_2$	$q_0 q_2$

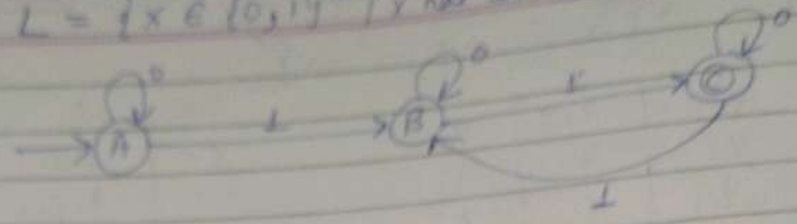


2. b)

NFA	ε - NFA
• It transits from a given state on a given input symbol	It moves from one state to another without consuming any input symbol.
• Does not have epsilon transitions.	Can include both regular as well as epsilon transitions
• These transitions are represented by input symbol	These transitions are represented by Greek letter ε.
• Recognises regular languages	Recognises same set of languages as NFA.
• less flexibility is provided	More flexibility in state transitions

$L = \{x \in \{0,1\}^* \mid x$ has an even no. of 1's$\}$

2. c)



2. d) It is a powerful tool in the realm of formal languages and automata theory. It allows us to convert a given finite automation into an equivalent regular expression.

Statement : Let P and Q be two regular expressions. If P does not contain the null string (i.e. the empty language), then the equation $R = Q + RP$ has a unique solution, which is $R = QP^*$.

→ The transition diagram must not have NULL transitions.
→ It must have only one initial state.

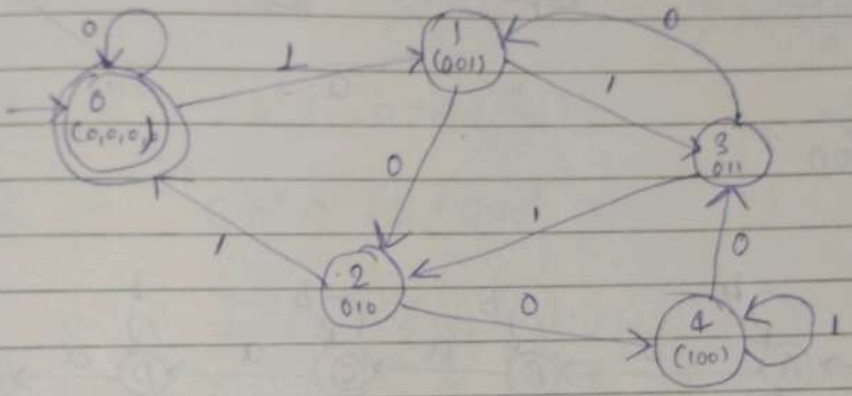
Ex. $q_1 = q_1 \cdot 0 + \epsilon$
$q_2 = q_1 \cdot 1 + q_2 \cdot 0$
$q_3 = q_2 \cdot 1 + q_3 \cdot 0 + q_3 \cdot 1$

Now → $q_1 = \epsilon + q_1 \cdot 0 = \epsilon 0^* = 0^*$
$q_2 = 0^* 1 + q_2 \cdot 0 = 0^* 1 0^*$

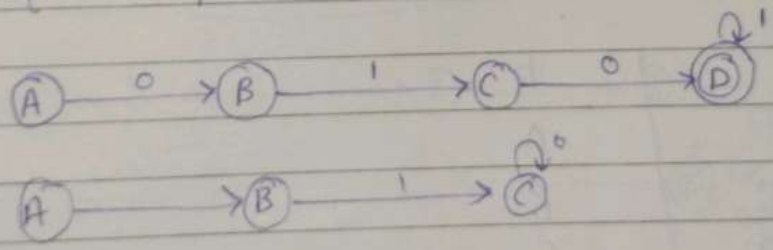
1. a)

Binary no.	Remainder
01	1
10	2
11	3
100	4
101	0
110	1
...	...
1010	0

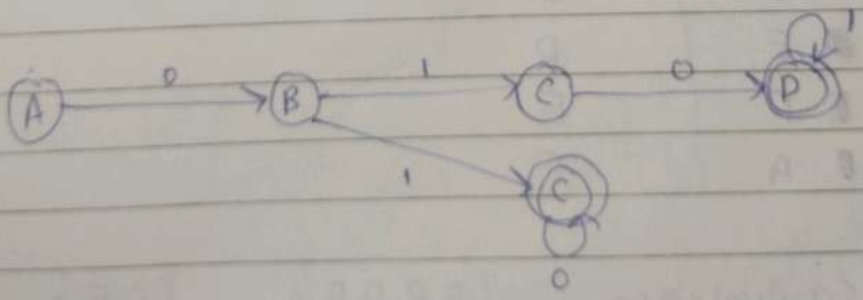
5 states



1. b) $\{0101^p \mid p \geq 0\} \cup \{010^p \mid p \geq 0\}$

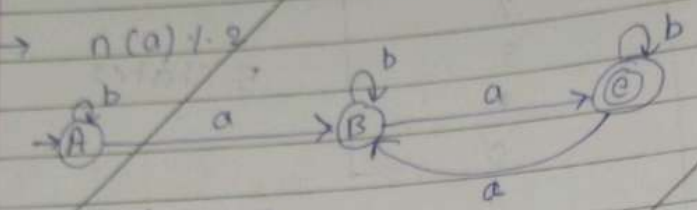


⇒

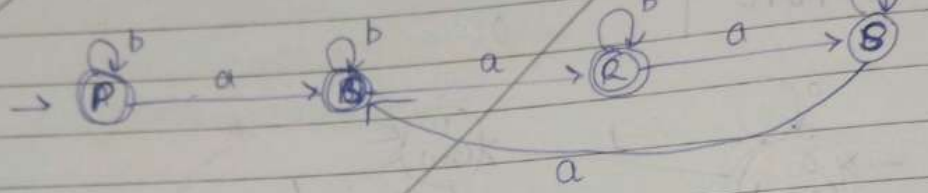


1. a) $L = \{x \in (a,b)^* \mid$ n(a) %. 2 not by 3 $\}$

→ n(a) %. 2

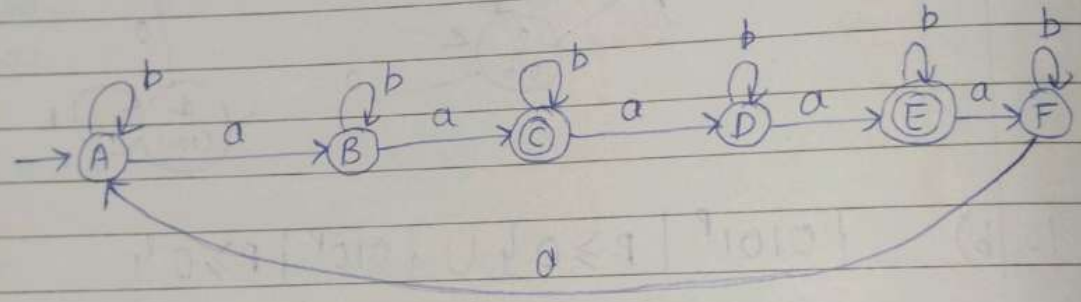


→ n(a) %. 3



union

1. c)



	a	b
A	B	A
B	C	B
C	D	C
D	E	D
E	F	E
F	A	F

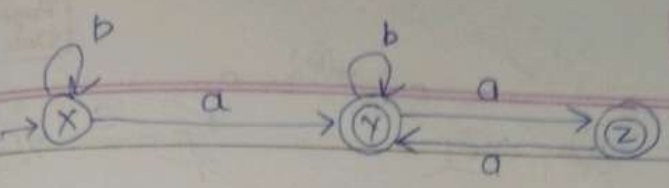
0 Equivalence {A B D F} {C E}

1 Equivalence {A} {B D} {C E}
{F}

2 Equivalence {A} {B} {D} {C} {E} {F}

[R = Q + RP] [R = QP*]

Scribble
Page:
Date: / /



10 d)

$X = \epsilon + Xb$

$Y = Xa + Yb + Za$

$Z = Ya$

$X = \underbrace{\epsilon}_{Q} + \underbrace{X}_{R}\underbrace{b}_{P} \Rightarrow X = \epsilon b^*$

$Y = \epsilon b^* a + Yb + Yaa$

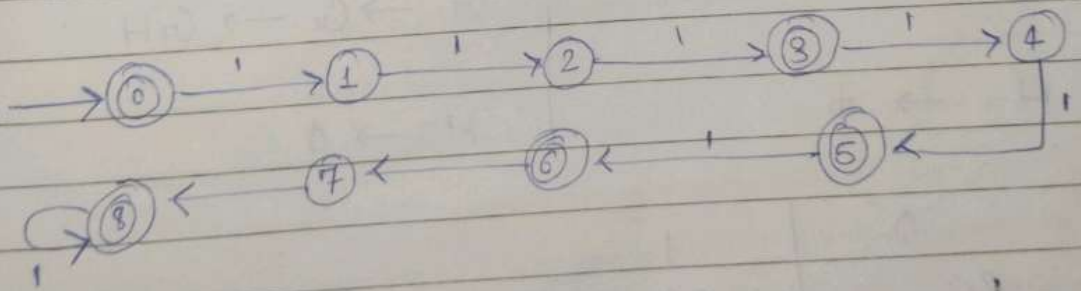
$Y = \underbrace{\epsilon b^* a}_{Q} + \underbrace{Y}_{R}\underbrace{(b + aa)}_{P}$

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

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

20 a) $R = (111 + 11111)^*$ $(a+b)^*$

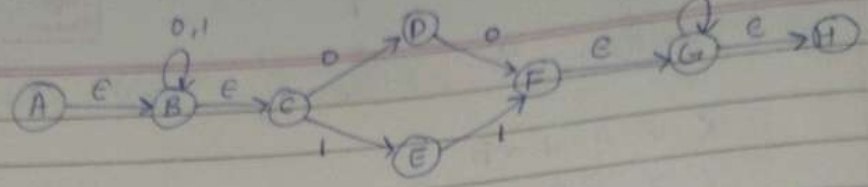
$L = \{111, 11111, 111111, 1111111, 11111111\}$



Scribble

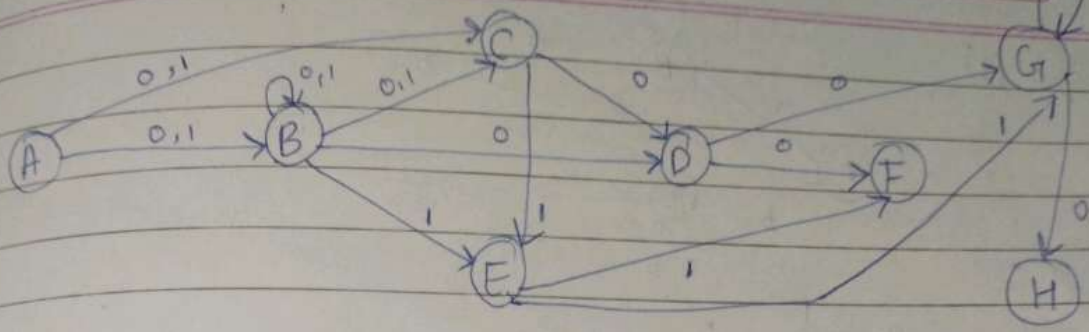
Page
Date

(2) b)



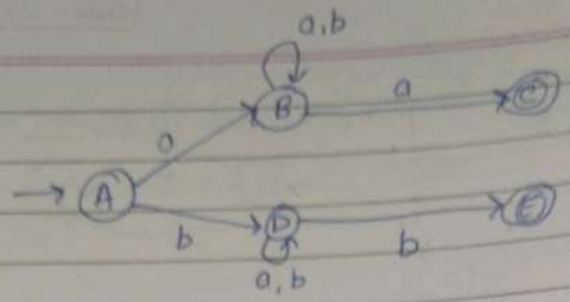
	$\varepsilon^* \; 0 \; \varepsilon^*$	$\varepsilon^* \; 1 \; \varepsilon^*$
A	$A \to \phi$ $B \to B \to BC$	$A \to \phi$ $B \to B \to BC$
B	$B \to B \to BC$ $C \to D \to D$	$B \to B \to BC$ $C \to E \to E$
C	$C \to D \to D$	$C \to E \to E$
D	$D \to F \to FG$	$D \to \phi$
E	$E \to \phi$	$E \to F \to FG$
F	$F \to \phi$	$F \to \phi$
G	$G \to G \to GH$	$G \to G \to GH$
H	$H \to \phi$	$H \to \phi$

	0	1
A	BC	BC
B	BCD	BCE
C	D	E
D	FG	ϕ
E	ϕ	FG
F	ϕ	ϕ
G	GH	GH
H	ϕ	ϕ



(2) c) I) $\{1^a 2^b 3^c 4^d \mid a, b, c, d \geq 0\}$

② d)



	a	b
A	B	D
B	B,C	B
C	ϕ	ϕ
D	D, C	D, E
E	ϕ	ϕ

	a	b
A	B	D
B	BC	B
(BC)	BC	B
D	D	DE
(DE)	D	DE