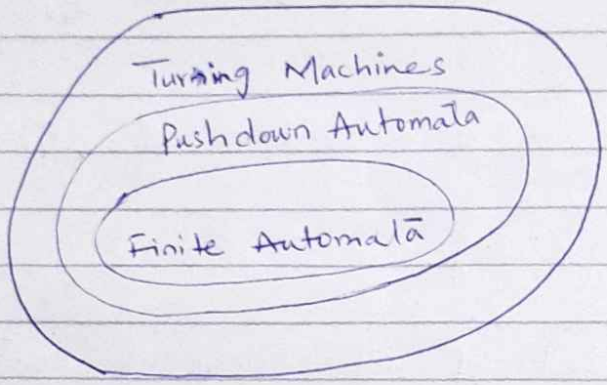
CS-331-TOA

THEORY OF AUTOMATA

DR. AKMAL KHATTAK

→ Course Outline and Introduction:

- Machine
- Regular Expression
- Grammar



Machine → Abstract form of compilations.
↳ Not Hardware → Abstract Machines.

→ Mathematically Modeling of Compilational Tasks.

- Symbols - a, b, c, ----, 0, 1, 2, 3, ----, —, ...
- Alphabets - A collection of Symbols

$\Sigma = \{a, b, c\}$ $\Sigma = \{a, b, 0, 1, 2\}$

- String / Word — A sequence of symbols.

  e.g:- aabc , acbab.

  Input Alphabet → $\Sigma = \{a, b\}$

  a
  b
  ab
  aab
  ] valid string

  aac ✗ Invalid 'Σ'.

- Language → A collection of strings. & Rules

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

  L_1 = Set of all strings of length 2.

  $= \{aa, ab, ba, bb\}$ → Finite Set.

  L_2 = Set of all strings of length 3.

  $= \{aaa, aab, aba, abb, baa, bba, bab, bbb\}$

  L_3 = Set of all strings ~~of length 4.~~ that begins with 'a'

  $= \{a, aa, aaa, ab, abb \ldots\}$

→ Power of Σ

Σ^0 = Set of all strings over Σ of length 0.

= $\{\epsilon\}$ → empty set.

Σ^1 = Set of all strings over Σ of length 1

= $\{a, b\}$

Σ^2 = Set of all strings over Σ of length 2.

Σ^n = " " " " " n.

→ Mother of all Languages / Universal set.

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots \Sigma^n$.

$= \{\epsilon\} \cup \{a, b\} \cup \{ab, aa, ba, bb\} \cup \dots$

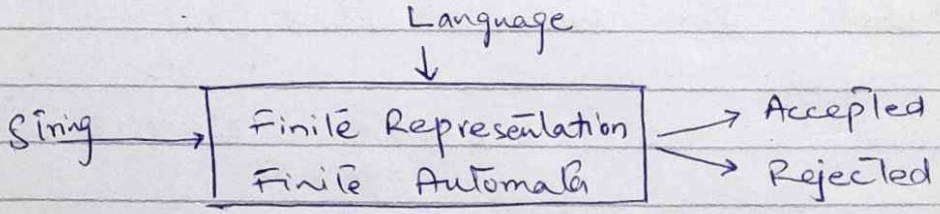
$L_1 \subseteq \Sigma^*$

$L_2 \subseteq \Sigma^*$

$L_3 \subseteq \Sigma^*$.

Σ^* → Infinite set.

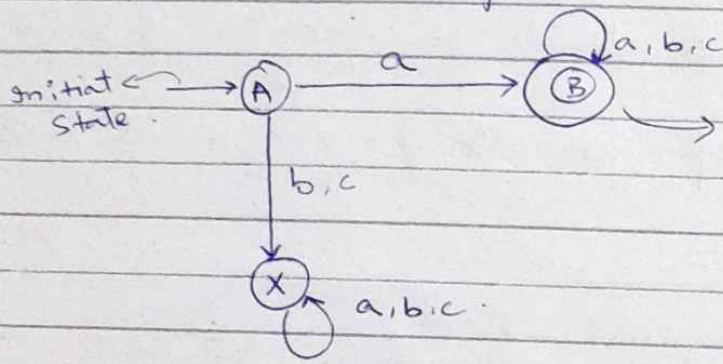
Every language is subset of Σ^*.



L = Set of all Strings That begins with 'a'.

= { a, aa, ab, ... }

State Transition Diagram.



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

ab ab $A \xrightarrow{a} B \xrightarrow{b} B \xrightarrow{a} B \xrightarrow{b} B$.

babb $A \xrightarrow{b} X \xrightarrow{a} X \xrightarrow{b} X \xrightarrow{b} X$

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

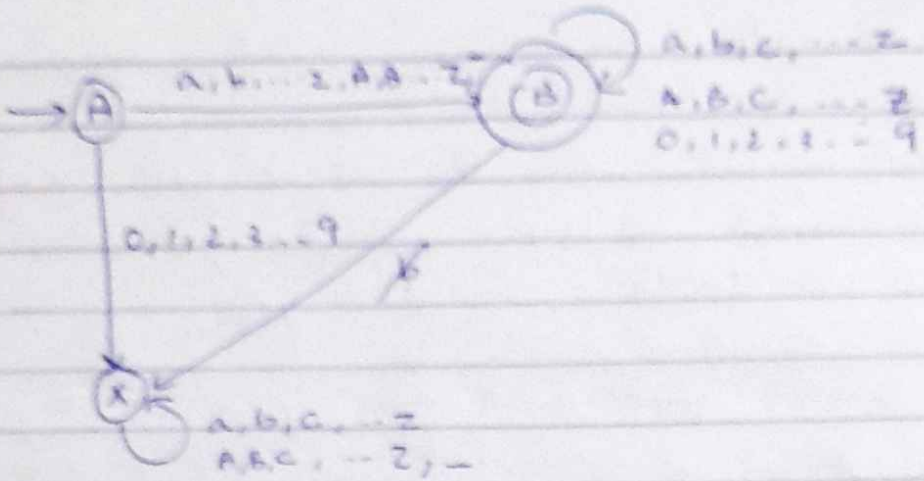
abc.

Model for variable identifier.

$\Sigma = \{a, b, c, \ldots z, A, B, C \ldots Z, 0, 1, 2, 3 \ldots 9, \_, \ldots\ldots\}$.

predefined special characters

Not infinite



friday.

Lecture No. 2

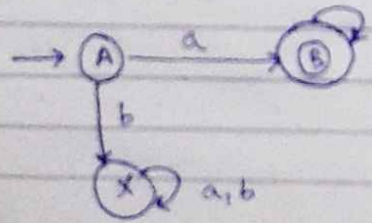
→ DFA - Deterministic finite Automata.

A DFA is represented by a 5-Tuple

$$(Q, \Sigma, \delta, q_0, F).$$

Q → Finite set of states
Σ → Alphabet
q_0 → Initial state
F → Accepting states
δ → delta
↳ is a transition function.

$$\delta : Q \times \Sigma \to Q$$



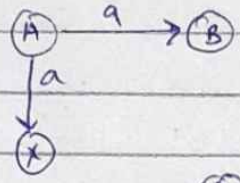
$Q = \{A, B, X\}$

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

① Har state sy har input symbol exhaust hona chiye

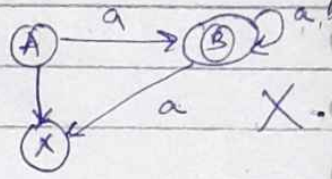
② Har symbol aik he state par map hona chiye. Multiple states par nahi map ho sakta.

Transition function



X.

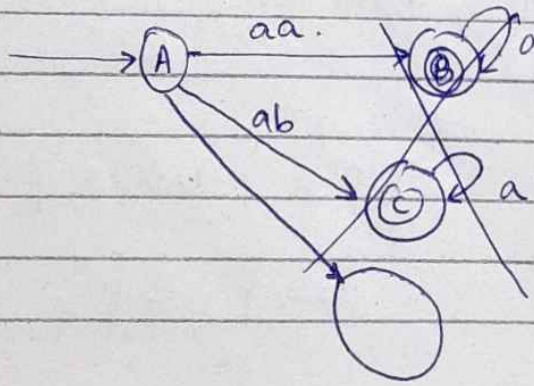
$\delta : (A, a) \rightarrow B$.
$\delta : (A, b) \rightarrow X$
$\delta : (B, a) \rightarrow B$
$\delta : (B, b) \rightarrow B$
$\delta : (X, a) \rightarrow X$
$\delta : (X, b) \rightarrow X$.



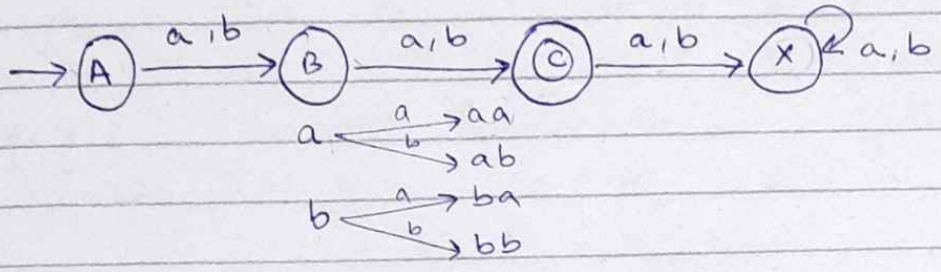
X.

Atleast one accepting state should be in DFA. It is not possible that there is no accepting state. There can be more than one accepting state.

→ Construct a DFA that accepts set of all strings over $\Sigma = \{a, b\}$ of length 2.

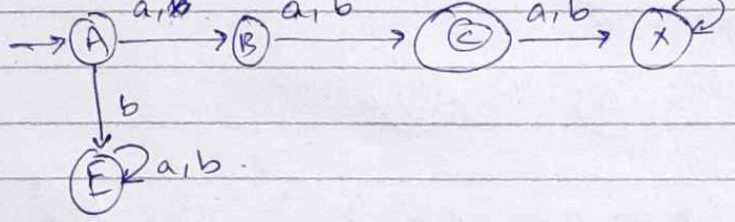


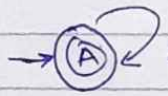
aab



There can be ~~x~~ multiple DFA's for one compilational problems But There is only one minimal DFA for a compilational ~~DFA~~ problem.

String of Length-2 starting with a.





It is accepts all the accepting strings of length-2. But it also accepts non-accepting strings. We should check for both.

ba ✓ A → B → C ✓

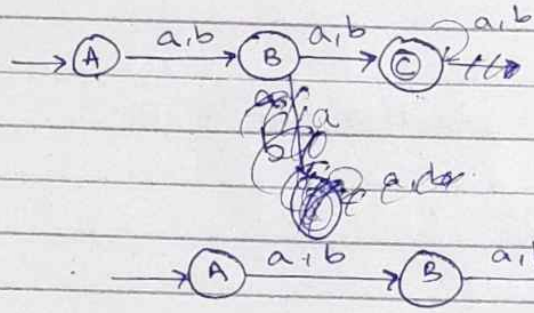
abb ✗ A → B → C → X ✗.

Finite set → DFA always exists.
Infinite set → Maybe DFA exists maybe Not.

→ A finite automata is set to accept a language if all the strings in the language are accepted and all the strings not in the language are not accepted.

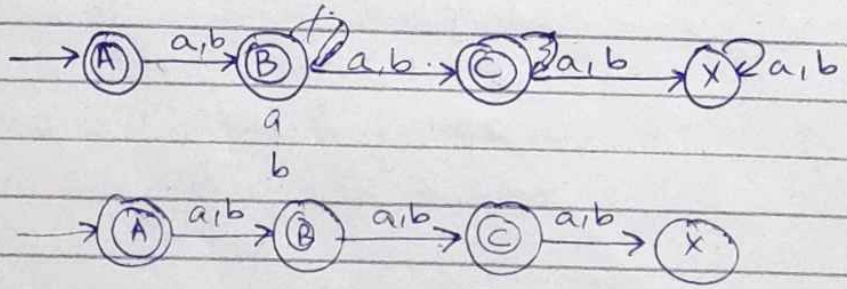
⇒ Construct a DFA that accepts set of all strings $w \in \{a,b\}$ such that $|w| \geq 2$.

~~{a,b,c,d, 1, 2, ...}~~ length.

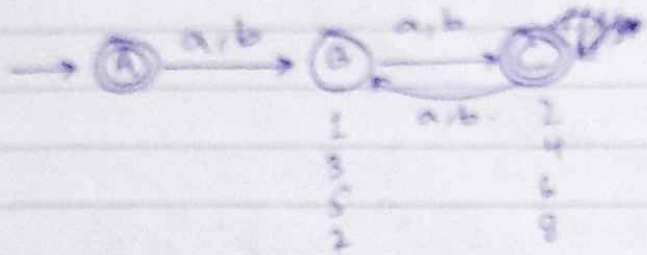


aaabbb → A $\xrightarrow{a}$ B $\xrightarrow{a}$ C $\xrightarrow{a}$ C $\xrightarrow{b}$ C $\xrightarrow{b}$ C $\xrightarrow{b}$ C

⇒ For $|w| \leq 2$.



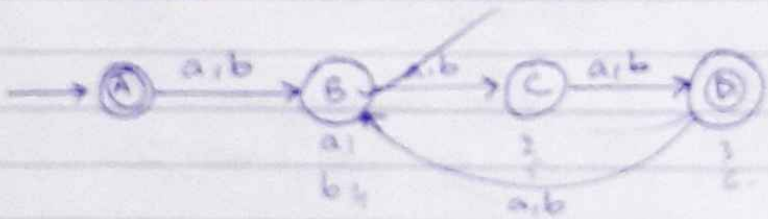
=> Construct a minimal DFA that accepts set of all strings over $\Sigma = \{a, b\}$ such that $|w| \bmod 2 = 0$



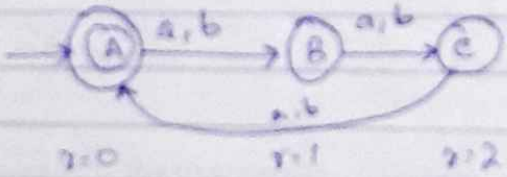
Not minimal

Minimal.

$|w| \bmod 3 = 0$



3 states



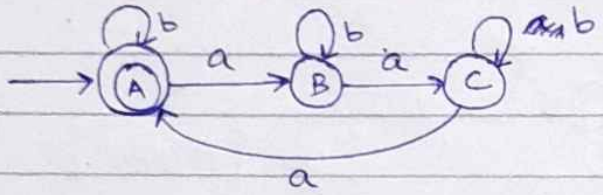
01/03/2024
Friday.

LECTURE NO. 4.

→ DFA's.

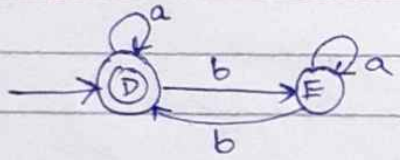
⇒ $n_a(w) \bmod 3 = 0$ && $n_b(w) \bmod 2 = 0$

$n_a(w) \bmod 3 = 0$.



$n_a = \{A, B, C\}$

$n_b(w) \bmod 2 = 0$.

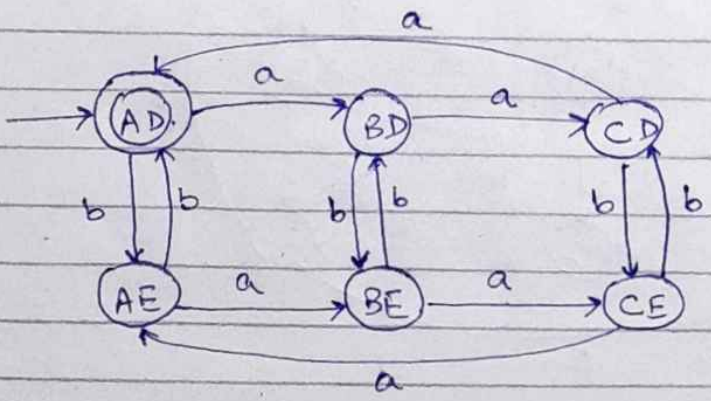


$n_b = \{D, E\}$

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

$\delta = \{A, B, C\} \times \{D, E\}$

$= \{AD, AE, BD, BE, CD, CE\}$



A —a→ B
E E

A —b→ A
E D

B —a→ C
D D

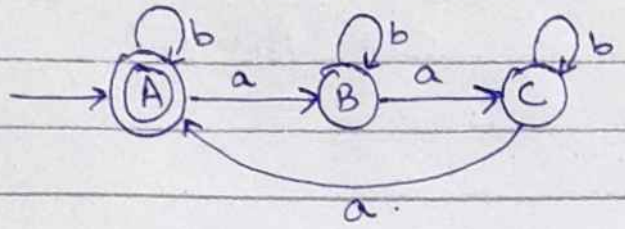
B —b→ B
D E

B —a→ C
E E

B —b→ B
E D

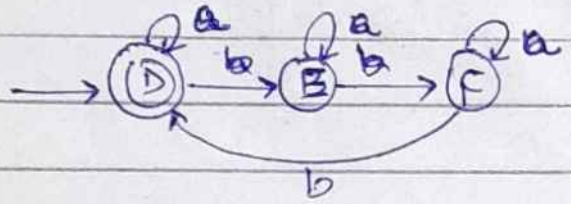
→ Construct a minimal DFA which accepts set of all strings over $\Sigma = \{a, b\}$ where number of a's & b's are divisible by 3.

$n_a(w) \bmod 3 = 0.$

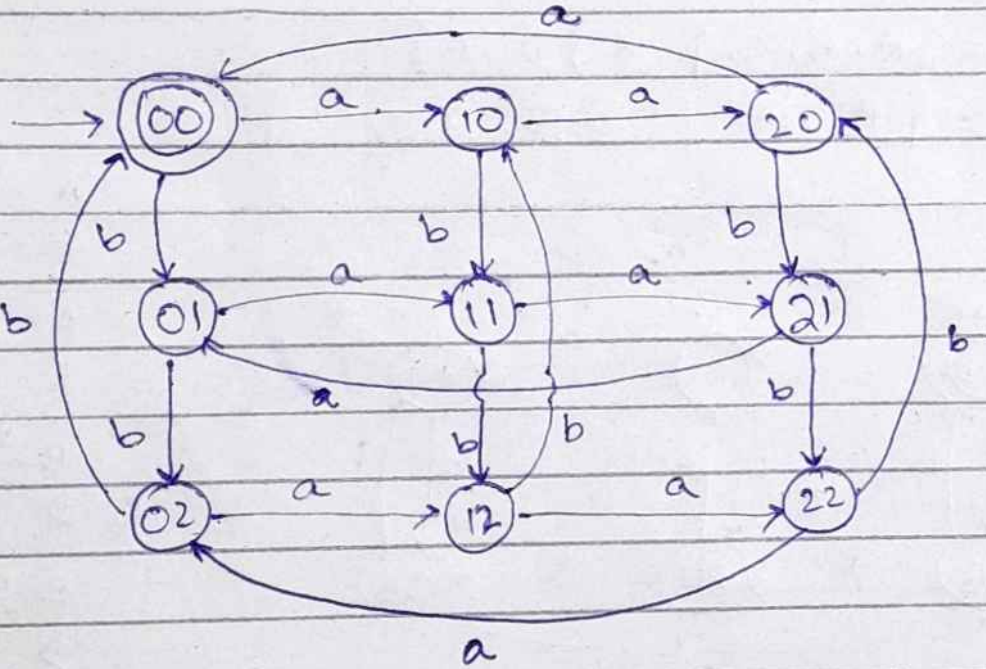


$n_a = \{A, B, C\}$

$n_b(w) \bmod 3 = 0.$



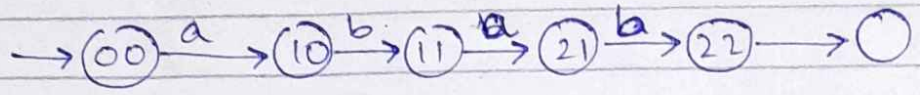
$n_b = \{D, E, F\}$

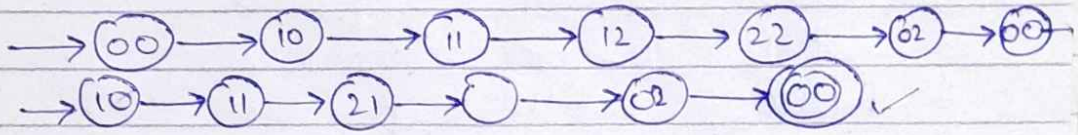


Verification:

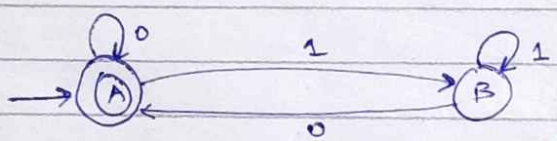
02/ abababbbab ✗ Non-accepting.

00/ abbaababababab ✓ Accepting.





→ Construct a minimal DFA which accepts set of all strings over $\Sigma = \{0,1\}$ which when interpreted as a binary number is divisible by 2.



0, 00, 0000 = 0

10, 010, 0010 = 2

100, 0100 = 4

0001 = 1

0011 = 3

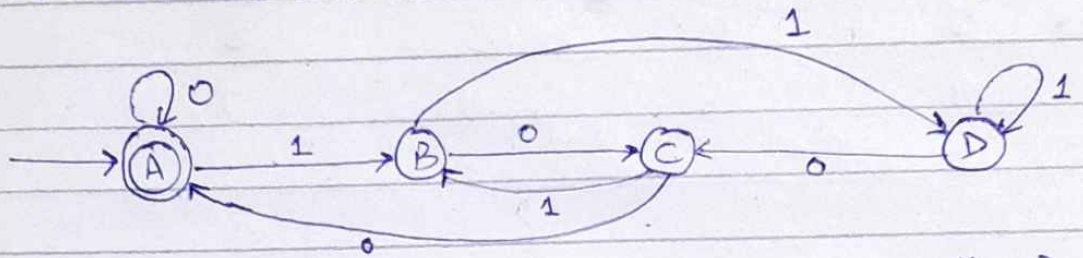
0101 = 5

05/03/2024
Tuesday

LECTURE No. 5

→ Construct a minimal DFA that accepts set of all strings over $\Sigma = \{0, 1\}$ which when interpreted as a binary number is divisible by 4.

Machine/Model/Automaton/State Transition Diagram.



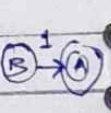
0000 = 0 0001 = 1 0010 = 2 0011 = 3
0100 = 4 0101 = 5 0110 = 6 0111 = 7
1000 = 8 1001 = 9 1010 = 10 1011 = 11
⋮ ⋮ ⋮ ⋮

→ State Transition Table:

	0	1
→* A	A	B
B	C	D
C	A	B
D	C	D

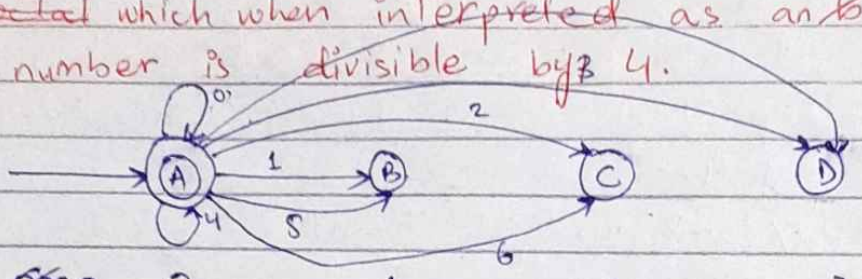
* → Represents Accepting state.

"→" Represents initial state.

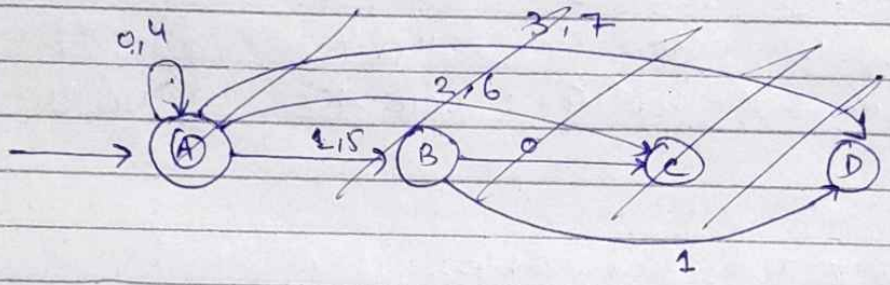


→ Construct a minimal DFA that accepts set of all strings over $\Sigma = \{0,1,2,3,4,5,6,7\}$ ~~octal~~ which when interpreted as an octal number is divisible by 4.

12 = 4
13 = 5
14 = 6.



0 1 2 3
4 5 6 7



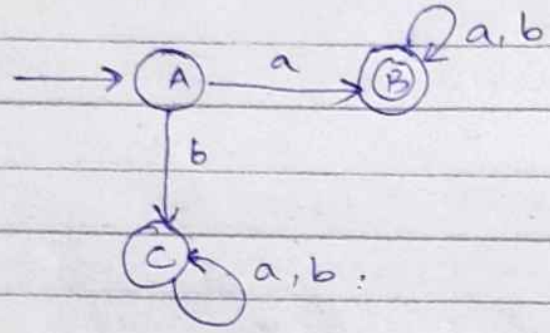
State Transition Table:

	0	1	2	3	4	5	6	7
A	A	B	C	D	A	B	C	D
B	A	B	C	D	A	B	C	D
C	A	B	C	D	A	B	C	D
D	A	B	C	D	A	B	C	D.

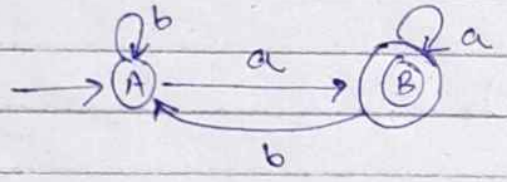
→ Construct a minimal DFA that accepts set of all strings over $\Sigma = \{a, b\}$

i) starting with 'a'.
ii) Ending with 'a'
iii) Containing 'a'.

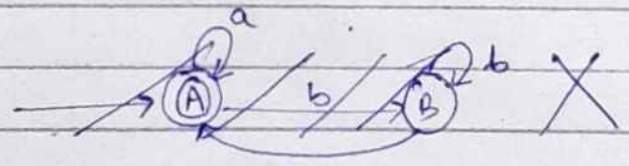
(i) Language = { a, aa, ab, abb ---}.

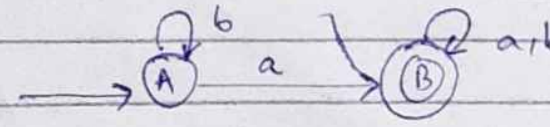


ii) Language = { a, aa, ba, aba, bba ---}.



iii)





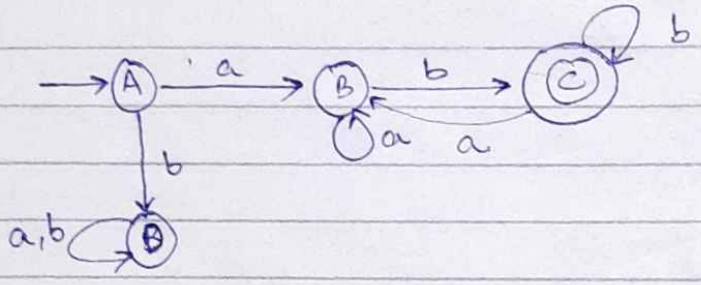
✓

08/03/2024

Lecture No. 6

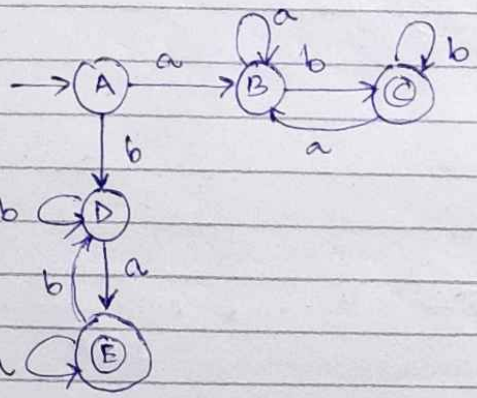
→ Construct a DFA which accepts set of all strings over $\Sigma = \{a, b\}$ that starts with 'a' and ends with 'b'

Language = { ab, abb, ---- }



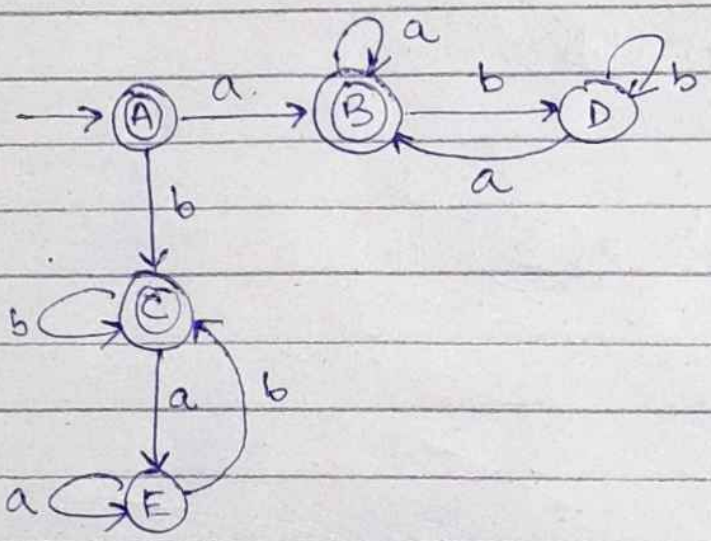
→ starts and ends with the different symbol..

Language = { ab, ba, abb, baa ---- }



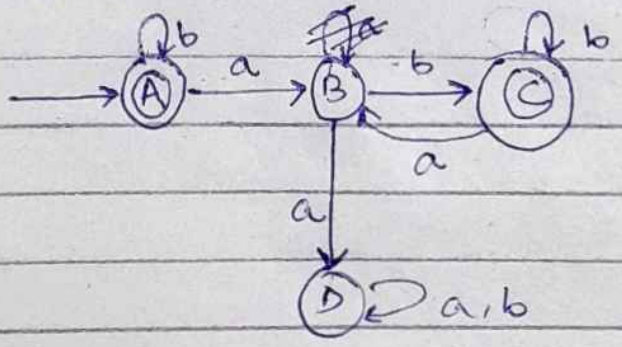
→ That starts and ends with same symbol.

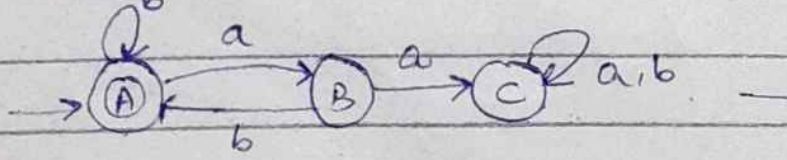
Language = { ε, a, aa, bb, b, aba, bab, ... }



→ That accepts set of all strings where every 'a' is followed by a 'b'.

Language = { ε, ab, abab, aab, b, bb, abb, ... }

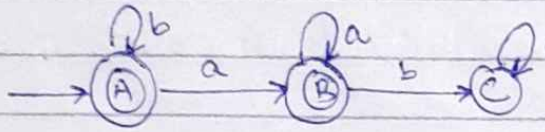




→ Minimal DFA

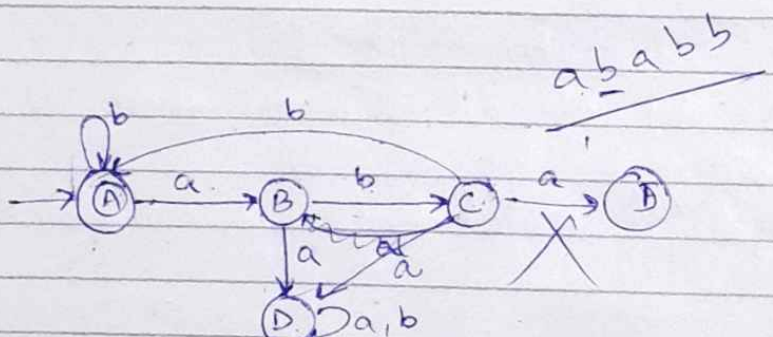
→ That accepts set of all strings where every 'a' is never followed by a 'b'.

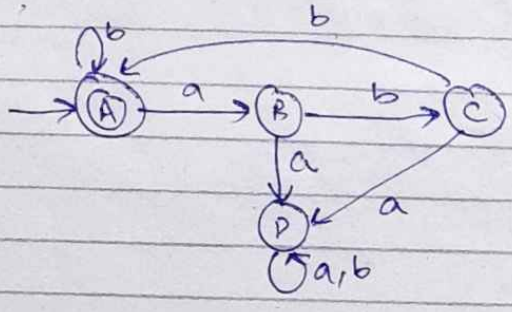
Language = { ε, a, aa, ba, bbb, bba ---- }.



→ where every 'a' is followed by a 'bb'

Language = { ε, b, bb, bbb, abb, abbabb, babb ---- }



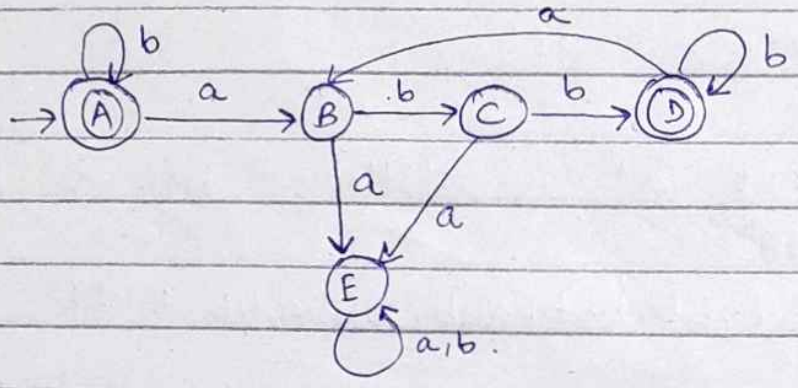


Tuesday.
12/03/2024

Lecture No. 7

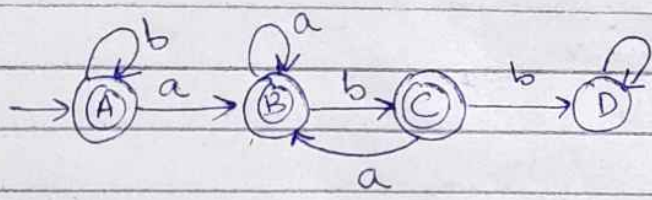
→ Construct a DFA over $\Sigma = \{a, b\}$ that accepts set of all strings where every 'a' is followed by a 'bb'.

Language = { ∈, b, bb, bbb, abb, abbbb, bbabb, abbbabb, ... }



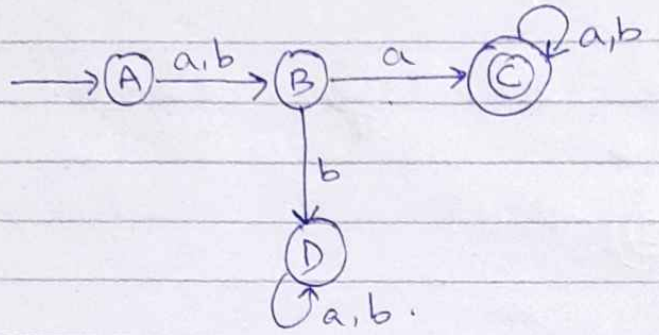
→ accepts set of all strings where 'a' is never followed by a "bb"

Language = { ∈, b, bb, bbb, ab, aab, aa, baa, abab

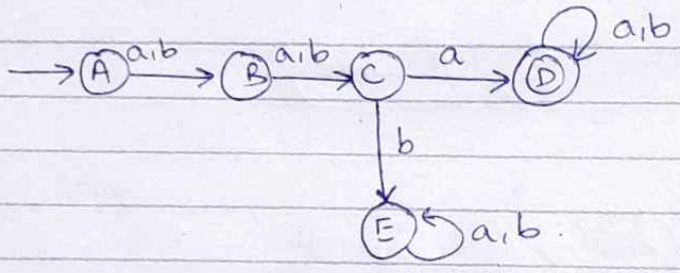


→ That accepts set of all strings where the second symbol is 'a'.

Language = { aa, ba, aab, bab, baaba ... }

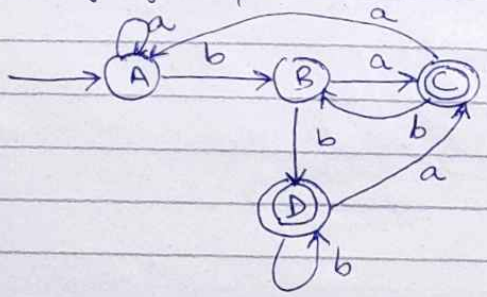


→ 3rd Symbol is 'a'



→ where the 2nd last symbol is "b"

Language = { ba, bb, bba, abb, abbba, aabb ... }

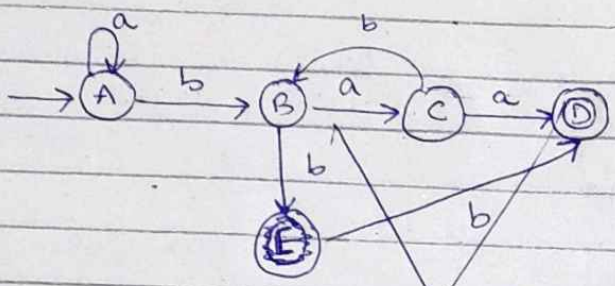


bab | aa ab bb ba

baab
abbb

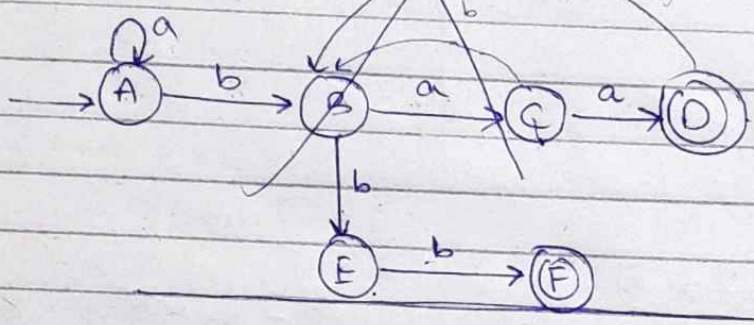
→ 3rd Last Symbol is 'b'.

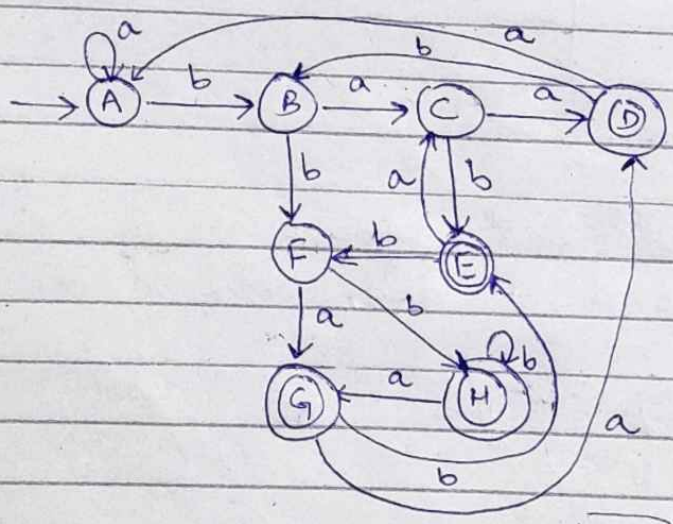
Language = { abab, abbb, baa, abbbb, babab
abaa, bbb, bab, bba }



3rd Last

$2^3 = 8$



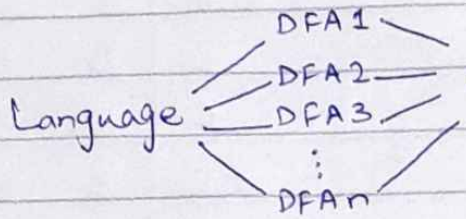


Hbaa

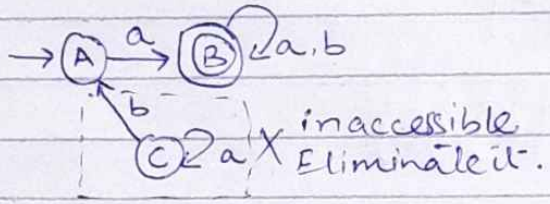
Friday
15/03/2024

LECTURE NO. 8

→ Minimization of DFA:



Step No. 1:
→ Eliminate inaccessible states. (which can't be reached from initial state).



→ Partioning Method:

• Two states can be combined if both are equivalent.

Two states are Equivalent if.

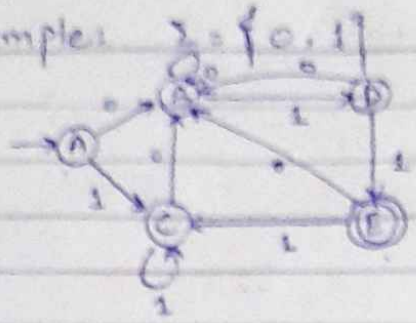
$\delta(A, x) \to F$ and $\delta(B, x) \to F$ OR $\delta(A, x) \not\to F$ and $\delta(B, x) \not\to F$.

A on input string x maps on F state which is accepting state.

where x is an input string.

if $|x| = 0$, Then A & B are 0 - Equivalent.
if $|x| = 1$, Then A & B are 1 - Equivalent.
if $|x| = 2$, Then A & B are 2 - Equivalent.
⋮
if $|x| = n$, Then A & B are n - Equivalent.

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



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

0-Equivalence: Non-Accepting, Accepting.

$\{A, B, C, D\}$ $\{E\}$

1-Equivalence:

$\{A, B, C\}$ $\{D\}$ $\{E\}$

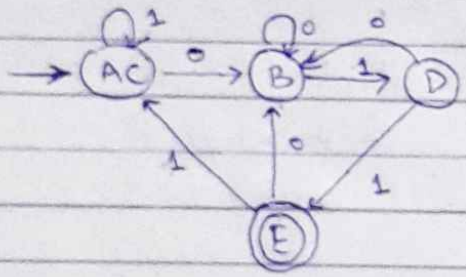
2-Equivalence:

$\{A, C\}$ $\{B\}$ $\{D\}$ $\{E\}$

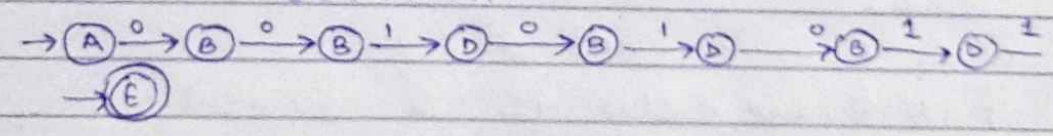
3-Equivalence:

$\{A, C\}$ $\{B\}$ $\{D\}$ $\{E\}$

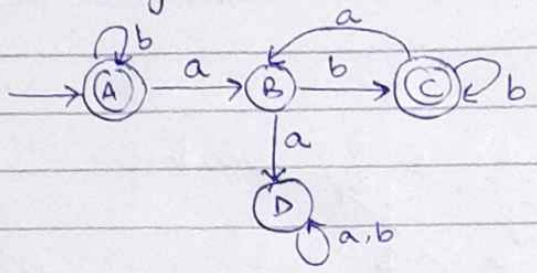
Now Merge



Verifications: 00101011



Example: Every 'a' has to be followed by a 'b'



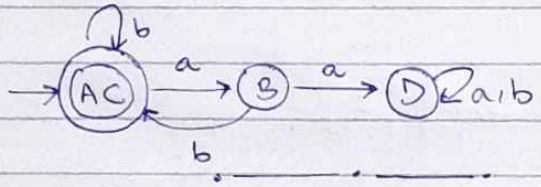
	a	b
→*A	B	A
B	D	C
*C	B	C
D	D	D

No, inaccessible states.

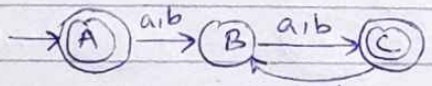
0-Equivalence: {B, D} {A, C}.

1-Equivalence: {B} {D} {A, C}

2-Equivalence: {B} {D} {A, C}



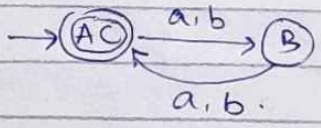
Example: $\Sigma = \{a, b\}$ all strings such that $|w| \bmod 2 = 0$.



	a	b
→*A	B	B
B	C	C
*C	B	B

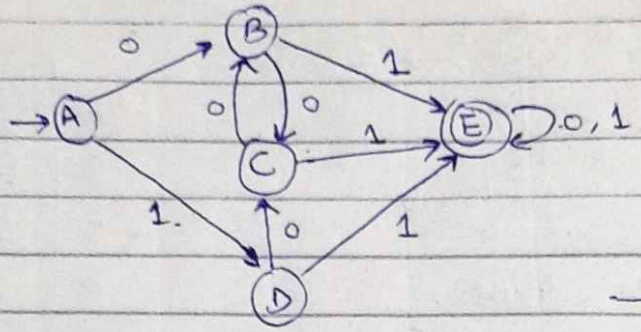
0-Equivalence: {A, C} {B}

1-Equivalence: {A, C} {B}



(Minimal DFA).

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



	0	1
→ A	B	D
B	C	E
C	B	E
D	C	E
*E	E	E

0-Equivalence:

$\{A, B, C, D\}$ $\{E\}$

1-Equivalence:

$\{A\}$ $\{B, C, D\}$ $\{E\}$

2-Equivalence:

$\{A\}$ $\{B, C, D\}$ $\{E\}$.

Minimized DFA:

19/03/2024
Tuesday

LECTURE No. 9

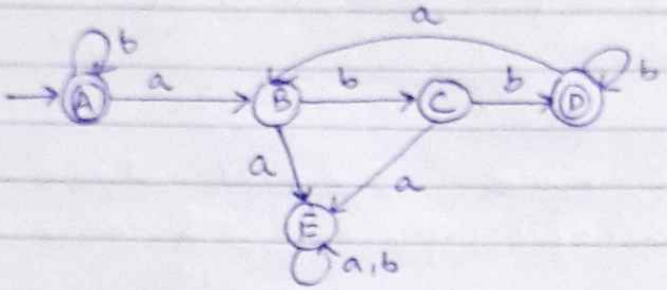
→ Minimization of DFA:

• Partioning Method:

Example # 1

→ 'a' is followed by a 'bb'.

Language = {abb, babb, ...}



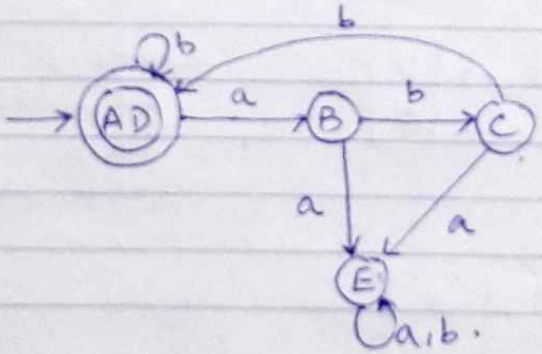
	a	b
→*A	B	A
B	E	C
C	E	B
*D	B	D
*E	E	E

0. Equivalence: {B, C, E} {A, D}

1. Equivalence: {B, E} {C} {A, D}

2. Equivalance: {B} {E} {C} {A, D}.

3. Equivalence: {B} {E} {C} {A, D}.



Verification: abbab ✗

→ AD →a B →b C →b AD →a B →b C

- Table Filling Method:

Example # 1.

	A	B	C	D	E
A					
B	✓				
C	✓	✓			
D	–	✓	✓		
E	✓	✓	✓	✓	

- Iteration-1:

1) Initially choose a pair where 1 state is accepting and 1 is non-accepting.

- Iteration-2:

C, B $\quad \delta(C, a) \rightarrow E$
$\quad \delta(B, a) \rightarrow E$

$\quad \delta(C, b) \rightarrow D$
$\quad \delta(B, b) \rightarrow C$

Check (D, C) cell if marked Then mark C, B other wise leave empty.

D, A $\quad \delta(D, a) \rightarrow B$
$\quad \delta(A, a) \rightarrow B$

$\quad \delta(D, b) \rightarrow D$
$\quad \delta(A, b) \rightarrow A$

will remain unmarked.

E, B $\delta(E,a) \to E$
$\delta(B,a) \to E$

$\delta(E,b) \to E$
$\delta(B,b) \to C$

E, C $\delta(E,a) \to E$
$\delta(C,a) \to E$

$\delta(E,b) \to E$
$\delta(C,b) \to D$

- Iteration-3:

D, A: $\delta(D,a) \to B$
$\delta(A,a) \to B$

$\delta(D,b) \to D$
$\delta(A,b) \to A$ unmarked

E, B:

$\delta(E,a) \to E$
$\delta(B,a) \to E$

$\delta(E,b) \to E$
$\delta(B,b) \to C$ Mark E, B bcz E, C is Marked.

- Iteration-4:

D, A: $\delta(D,a) \to B$
$\delta(A,a) \to B$

$\delta(D,b) \to D$
$\delta(A,b) \to A$ unmarked

29/03/2024.
Friday

LECTURE NO. 12

→ ∈-NFA (Epsilon NFA)

- Extended Type of NFA.
- ∈-NFA is easier to construct.

- ∈-NFA → NFA → DFA → Minimal DFA.
- Can be more than one ∈-NFA, DFA, NFA
- But there is only one minimal DFA.

* ∈-NFA is a 5-Tuple set.

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

Q → Finite set of states
Σ → Input Alphabet.
q_0 → Initial State
F → Accepting States

∈-NFA
$\delta: Q \times \Sigma \cup \{\epsilon\} \to 2^Q$

NFA
$\delta: Q \times \Sigma \to 2^Q$

DFA
$\delta: Q \times \Sigma \to Q$

Example:

Language = $\{a^m b^n c^p \mid m, n, p \geq 0\}$

$a^0 \rightarrow$ 0 occurences of a.

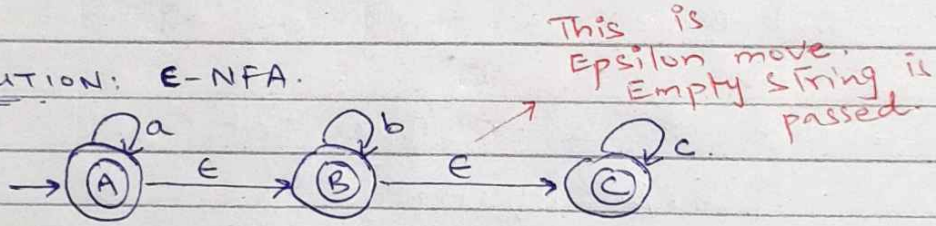
$a^1 \rightarrow$ 1 occurence of a.

$a^0 b^0 c^1 = c$.

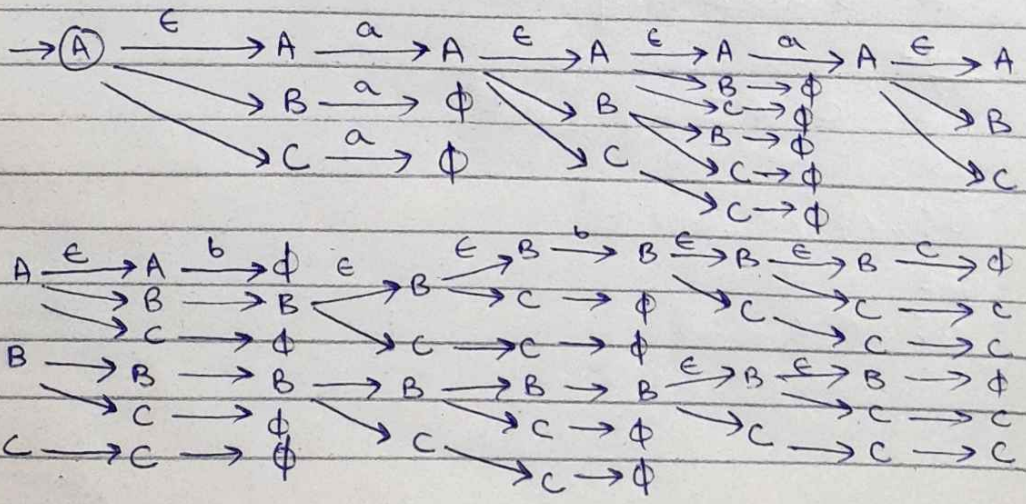
$a^1 b^0 c^1 = ac$

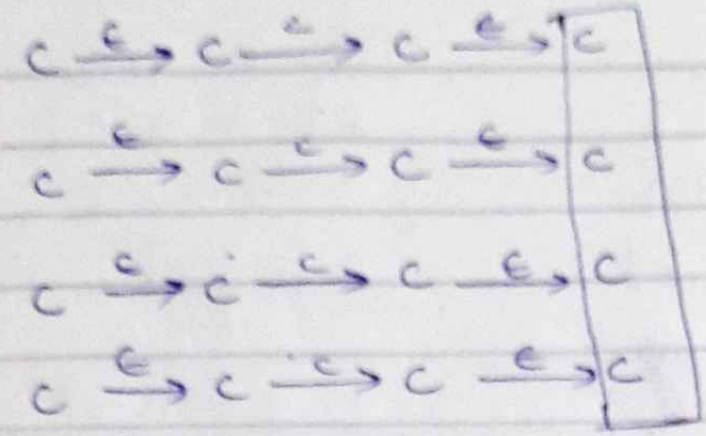
$a^2 b^2 c = aabbc$.

SOLUTION: ϵ-NFA.



- aabbcc → verification string



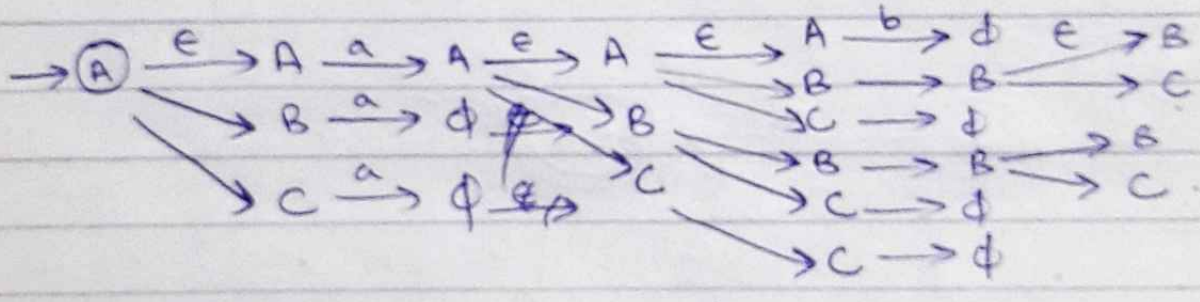


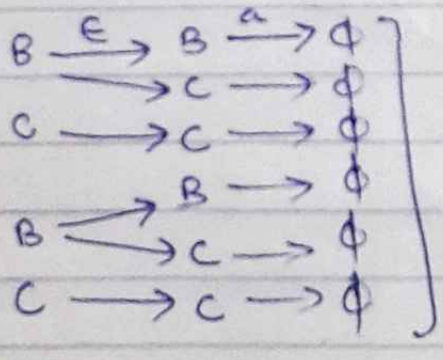
If atleast one of them is accepting state then string is accepted

Mathematical Formula:

$$\epsilon\text{-closure}\left(\delta(\epsilon\text{-closure}(A), x)\right)$$

. abac





Not accepted.

- ∈-NFA to NFA:

NFA

	a	b	c
→ *A	{A,B,C}	{B,C}	{C}
*B	ϕ	{B,C}	{C}
*C	ϕ	ϕ	{C}

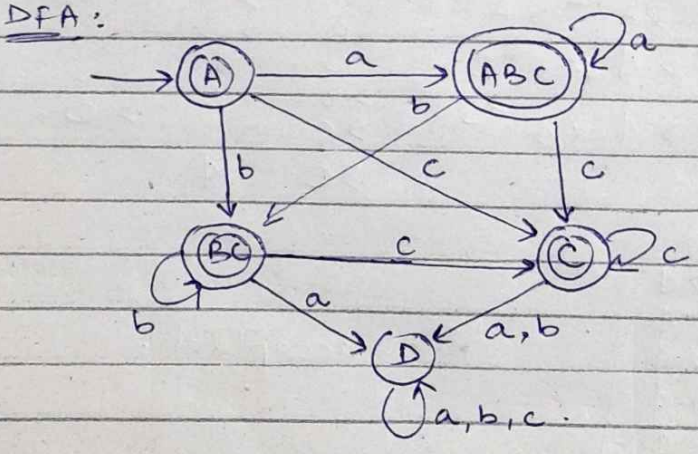
∈- Closure.

A	{A, B, C}
B	{B, C}
C	{C}

- NFA to DFA.

	a	b	c
→ *A	[ABC]	[BC]	[C]
*[ABC]	[ABC]	[BC]	[C]
*[BC]	[D]	[BC]	[C]
*[C]	[D]	[D]	[C]
[D]	[D]	[D]	[D]

DFA:



abaac:

→ A —a→ ABC —b→ BC —a→ D —c→ D

• DFA To Min DFA.

0- Equivalence: {A, ABC, BC, C} {D}.

1-Equivalence:

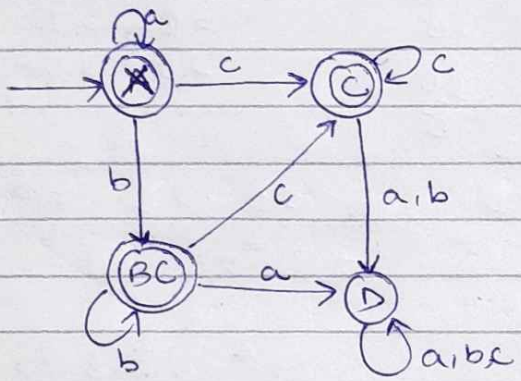
{A, ABC,} {BC}{C}{D}

2-Equivalence:

{A, ABC} {BC} {C} {D}.

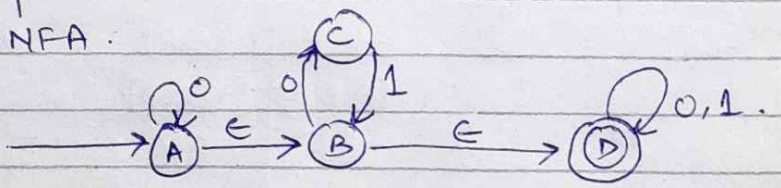
↳x

	a	b	c
→*A	ABC	BC	C
ABC	ABC	BC	C
BC	D	BC.	C
C	D	D	C
D	D	D	D



Example-2:

ϵ-NFA.



$0^*(01)^*(0+1)$

101101 ?

Imp

* All those states in NFA will be accepting states which through we can reach accepting state of ∈-NFA via epsilon move.

• ∈NFA to NFA.

So A, B will also be accepting states in NFA

	0	1
→*A	{A,B,C,D}	{D}
*B	{B,D}	{D}
C	ϕ	{BD}
*D	{D}	{D}

	∈-Closure.
A	{A,B,D}
B	{B,D}
C	{C}
D	{D}.

→ A ∈→ A →0 A ∈→ A
→ B →C →B
→ D →ϕ →D
→D

→ A ∈→ A →1 ϕ ∈
→ B →ϕ
→ D → D →D

→ B ∈→ B →0 C ∈→ C
→ D → D → D

→ B ∈→ B →1 ϕ ∈
→ D → D → D

→ C ∈→ C →0 ϕ

→ C ∈→ C →1 B ∈→ B
→ D

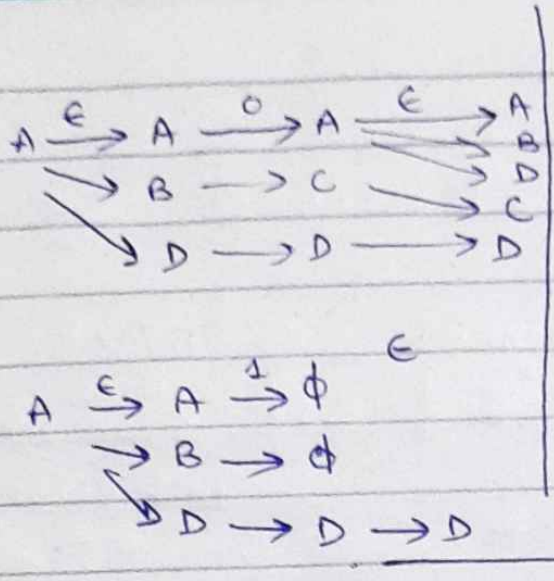
→ D ∈→ D →0 D ∈→ D

→ D ∈→ D →1 D ∈→ D

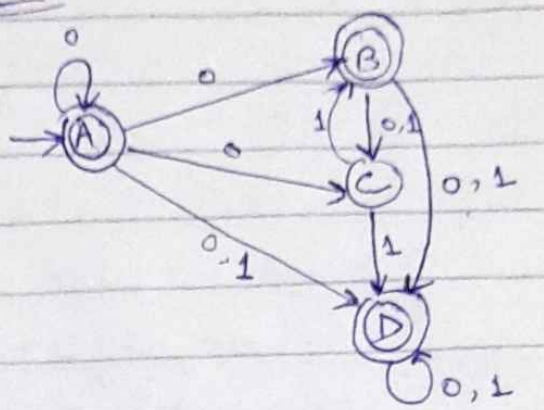
• NFA to DFA.

	0	1
→A	[ABD]	[D]
[ABD]	[ABCD]	[D]
D	[ABCD]	[BD]
[ABCD]	[ABCD]	[BD]
[BD]		

	0	1
→A	[ABD]	[D]
X *[ABD]	[ABCD]	[D]
*[D]	[D]	[D]
Y *[ABCD]	[ABCD]	[BD]
Z *[BD]	[CD]	[D]
*[CD]	[D]	[BD]



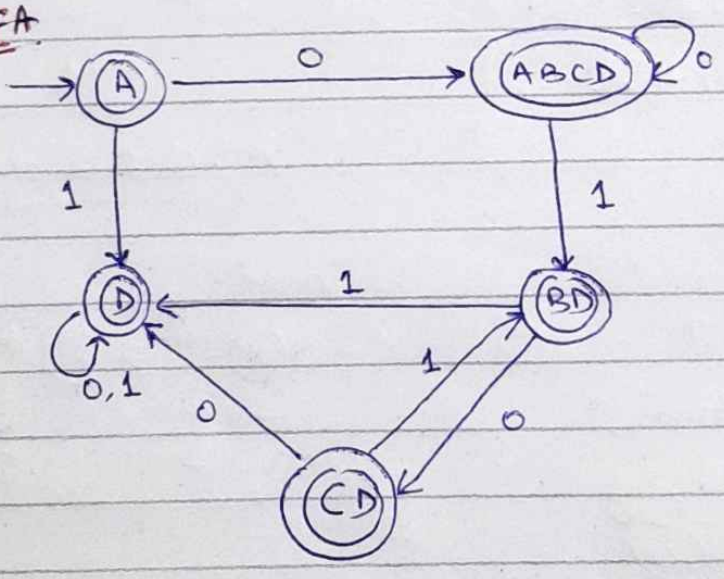
NFA



- NFA to DFA.

	0	1
→* [A]	[ABCD]	[D]
* [ABCD]	[ABCD]	[BD]
* [D]	[D]	[D]
* [BD]	[CD]	[BD]
* [CD]	[D]	[BD]

DFA



→ DFA to Minimal DFA.

Mid Term
19/04/2024
Friday (Mubarik Din)

LECTURE NO. 13

02/04/2024
Tuesday

→ Regular Language:

A Language is said to be regular if it can be described by a Finite Automata.

~~For / Exp~~ Non-Regular Language. → Context Free Grammer

Language = $\{0^n 1^n \mid n \geq 1\}$

Finite Automata Does not Exists.

Push Down Automata.

→ Finite Automata with Output.

Moore's Machine.

$Q, \Sigma, \delta, q_0, \Delta, \lambda$.

Q → Finite set of states

Σ → Input alphabet.

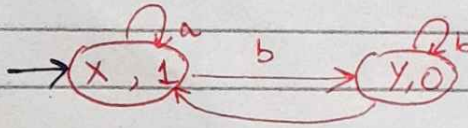
δ → Transition function.

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

q_0 → Initial State

Δ → Output Alphabet.

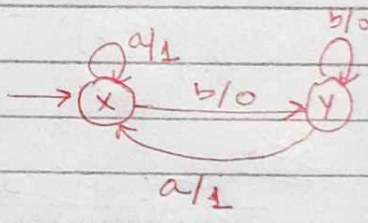
λ → Output function.



$\lambda : Q \rightarrow \Delta$

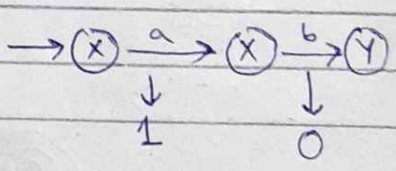
Each State is associated to an output symbol.

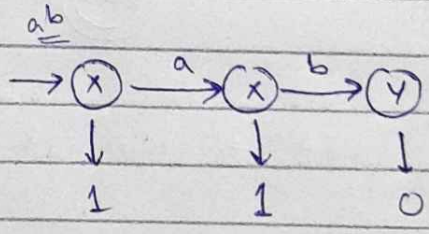
Mealy's Machine



$\lambda : Q \times \Sigma \rightarrow \Delta$

"ab"

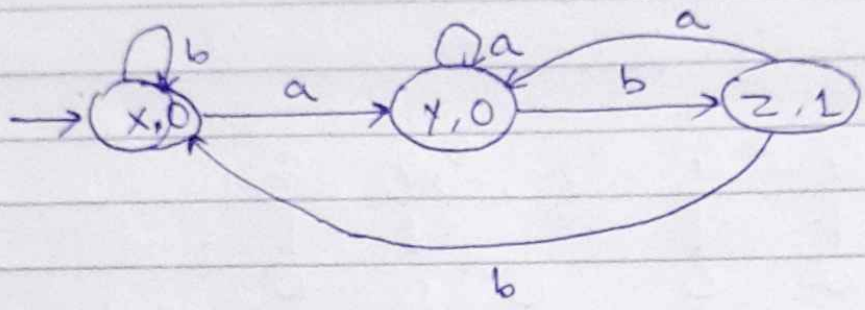




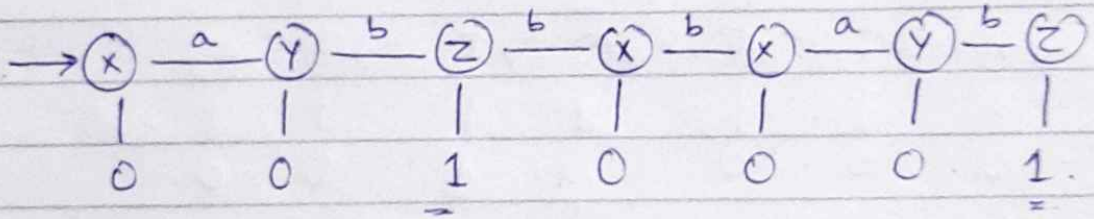
→ Moore's Machine:

Construct a Moore's Machine that takes set of all strings over $\Sigma = \{a, b\}$ and prints '1' as output for every occurrence of "ab".

Input Alphabet : $\Sigma = \{a, b\}$
Output Alphabet : $\Delta = \{0, 1\}$

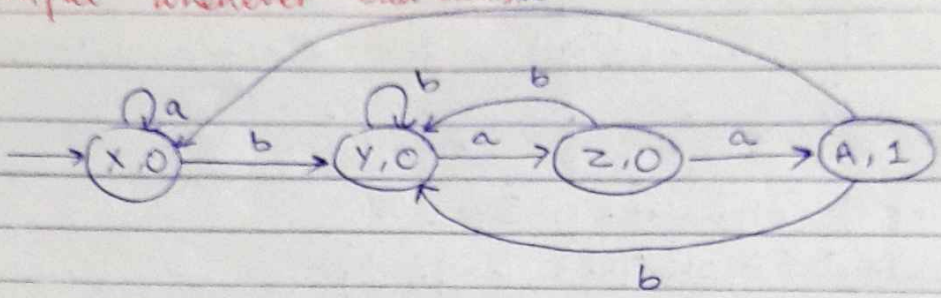


abbb ab

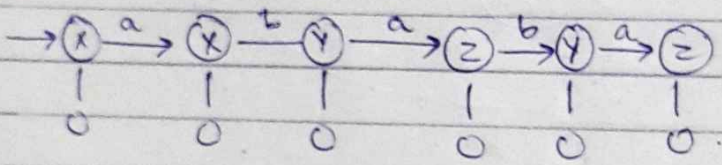


Moore's Machine is not for Accepting String or Non-Accepting String. It is an output generator.

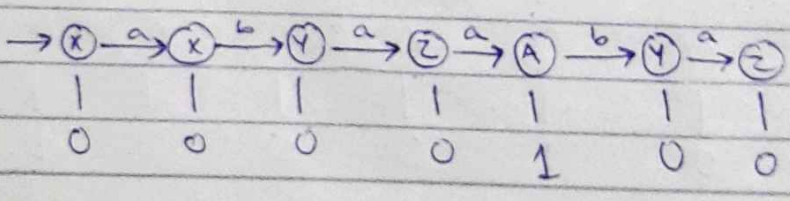
Construct a Moore's Machine that takes set of all strings over $\Sigma = \{a, b\}$ & prints '1' as output whenever "baa" occurs a



ababa



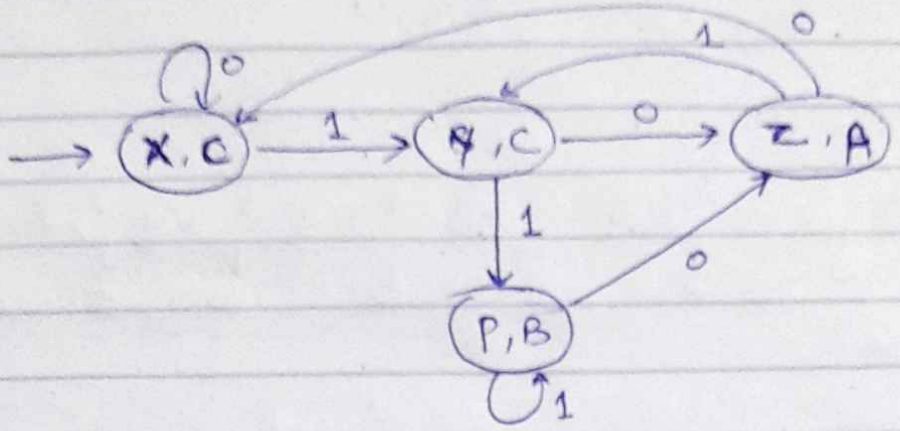
abaaba.



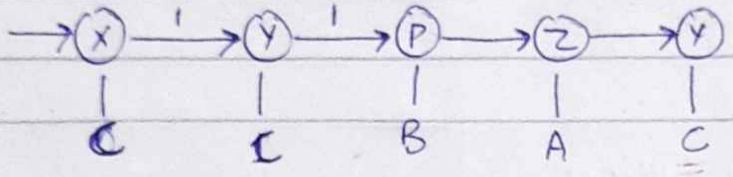
Construct a Moore's Machine that takes set of all strings over $\Sigma = \{0, 1\}$ & prints 'A' as output if the input ends with "10", produces "B" if the strings end with 11 otherwise produces "C".

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

Output Alphabet $\Delta = \{A, B, C\}$



. 1101 (Not Ending at 10 or 11.



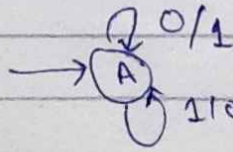
→ Mealy's Machine:

Construct a Mealy's Machine that takes binary number as input and produces 2's compliment of that number as output.

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

1's Compliment.

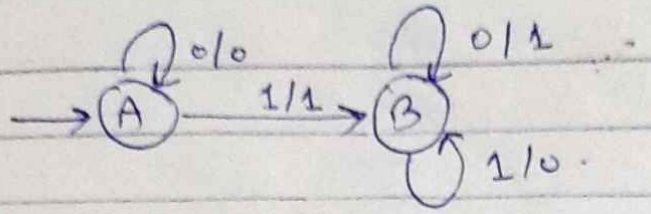
101011 → 010100

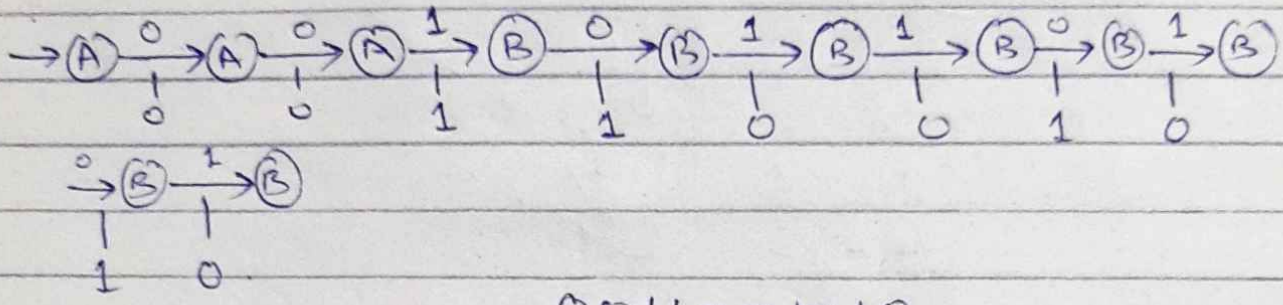


2's Compliment.

1010110100 ← Binary

0101001100 ← 2's Comp.

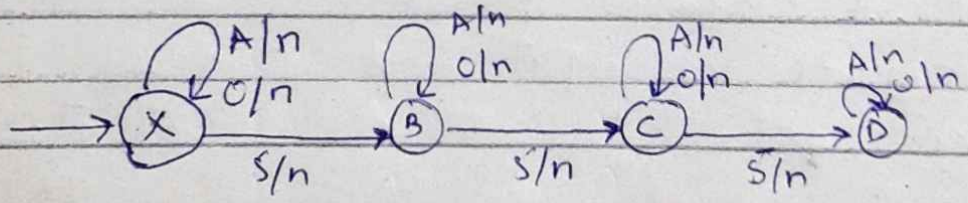


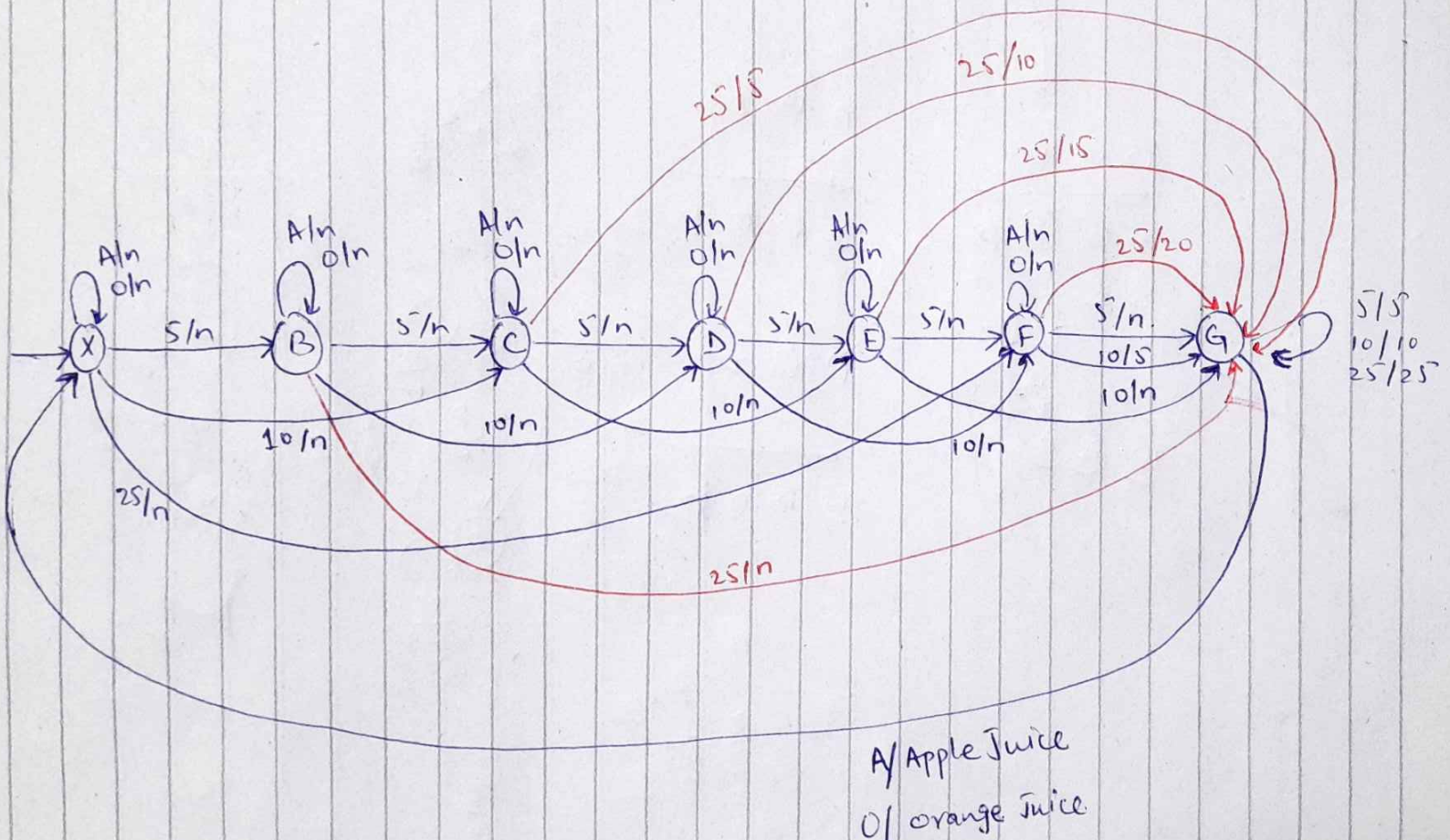


0 0 1 1 0 0 1 0 1 0

A ⟶ Apple Juice
O ⟶ Orange Juice — 30 cents

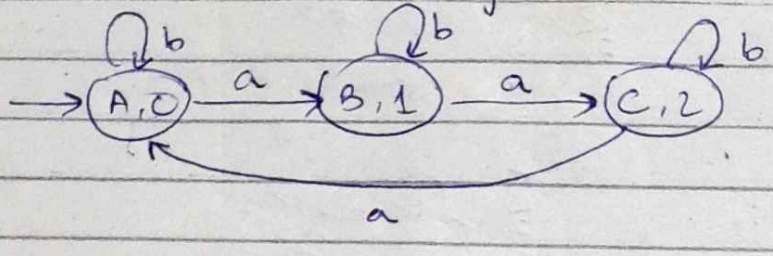
Coins → 5, 10, 25 cents. n → Nothing




A/n
O/n
5/n
10/n
25/n
25/5
25/10
25/15
25/20
10/5
5/5
10/10
25/25
X
B
C
D
E
F
G
A/Apple Juice
O/ orange Juice

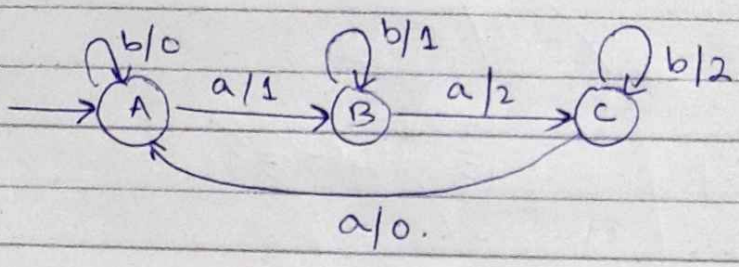
→ Moore's ⇄ Mealy's.

• Moore's To Mealy's.



Moore's

↓



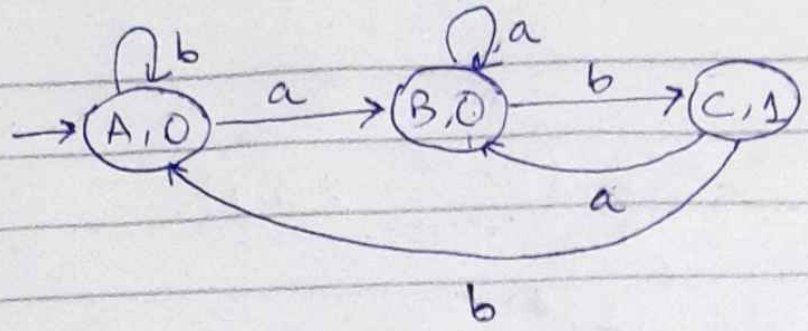
Mealy's

	a	b	Δ
→ A	B	A	0
B	C	B	1
C	A	C	2.

Moore's ← Machine Table

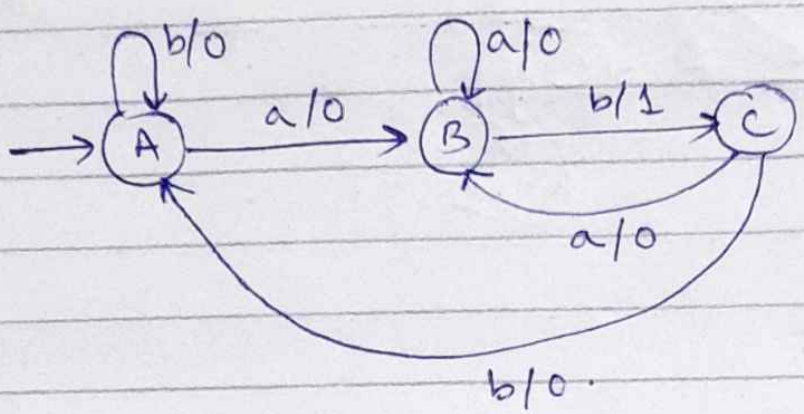
	a	b
→ A	(B, 1)	(A, 0)
B	(C, 2)	(B, 1)
C	A, 0)	(C, 2)

———



Moore's

↓

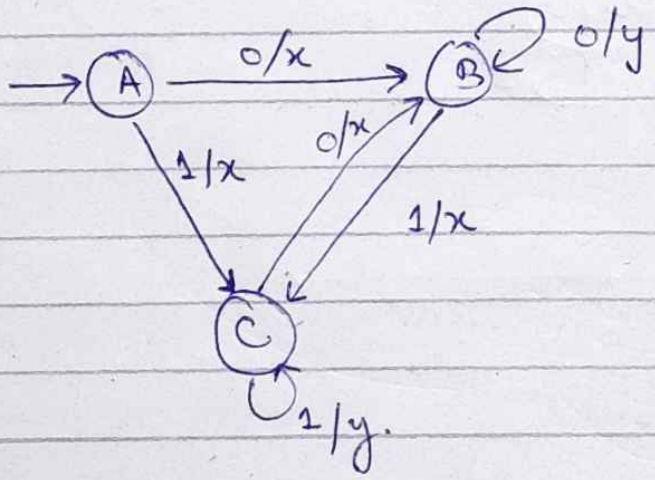


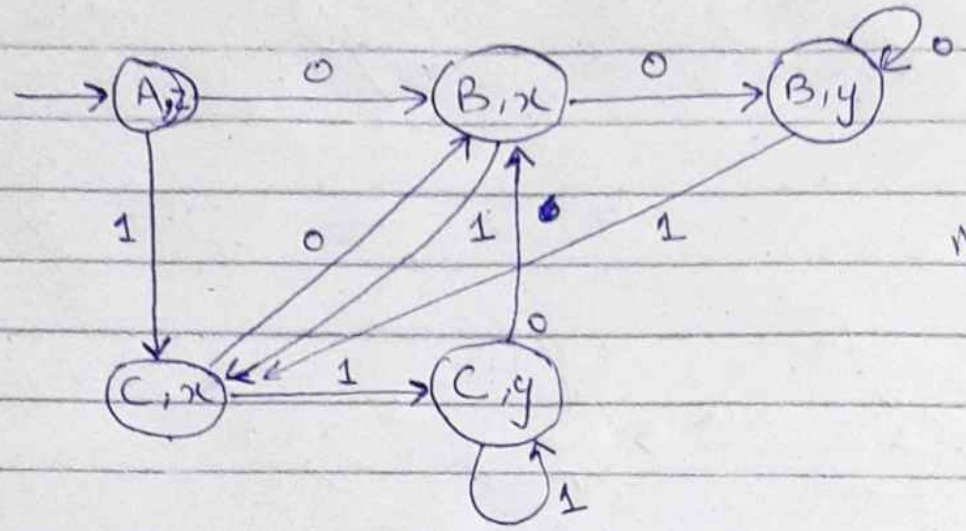
Mealy's.

	a	b	Δ
→ A	B	A	0
B	B	C	0
C	B	A	1

	a	b
→ A	(B,0)	(A,0)
B	(B,0)	(C,1)
C	(B,0)	(A,0)

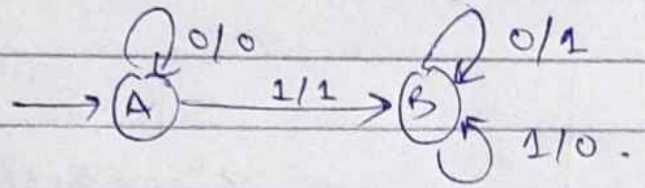
- Mealy's to Moore's.





Moore's Machine.

Exp #2



Mealy's Machine

↓

Moore's Machine.