THEORY OF COMPUTATION

Name - Prof. Anand Ghapu
college - PVGCOE, NSK
class - TE COMP.
Subject - TOC
Date - 24/08/2019
8087777708

MODEL ANSWER SHEET INSEM - 2019

Q.1. a) Define the foll. term with example. — 3 Marks
i) DFA ii) NFA iii) epsilon NFA.

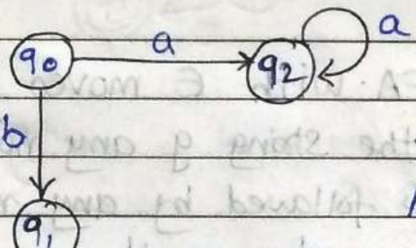
→ 1) DFA :

"A deterministic Finite Automata is collection of foll. things :
1) Finite set of states. which can be denoted by Q.
2) The finite set of i/p symbols Σ.
3) The start state q_0.
4) The set of final state F.
5) The mapping/transition fn i.e. denoted by δ"

Example :

The DFA is five tuple denoted by
$A = (Q, \Sigma, \delta, q_0, F)$

e.g.



∴ Each state contains 1 i/p as transition. As q_0 state has single transition for a input.

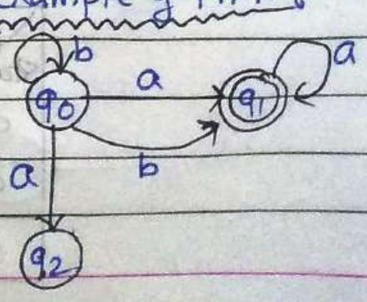
DFA

2) NFA :

"The NFA can be defined as collection of 5-tuple
1) Q is finite set of state
2) Σ finite set of input
3) δ is called next state / transition function
4) q_0 is initial state.
5) F is final state"

- NFA can have multiple final states.
- easy to use.
- more flexible.

example of NFA :



3) Epsilon NFA : ϵ-NFA

ϵ-NFA can be defined as

Let $M = (Q, \Sigma, \delta, q_0, F)$ be NFA with ϵ.

where 1) Q is finite set of state.

2) Σ is I/P set.

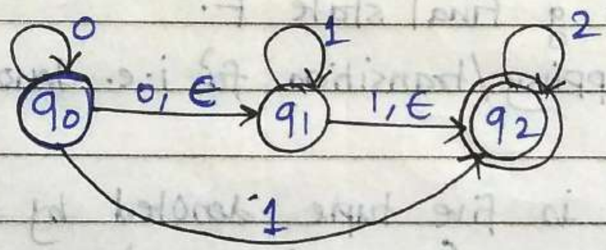
3) δ is transition function

4) q_0 is start state.

5) F is final state.

- It is similar as NFA. only ϵ symbol is added to it
- There can be ϵ transition for any state.
- ϵ-moves used to change state.

Example :

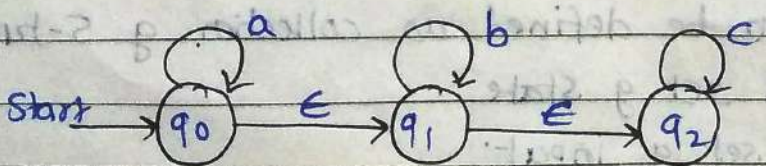


Q.1 3-Marks

b) Construct NFA with ϵ moves which accept lang. consisting the string of any no's of a's followed by any no's of b's followed by any no's of c's

→ Here any no's of a's or b's or c's means. zero or more in number. That means there can be 0 or more nos a's or more b's followed by 0 or more c's.

Hence, NFA-ϵ can be :



Normally ϵ's are not shown in I/P string

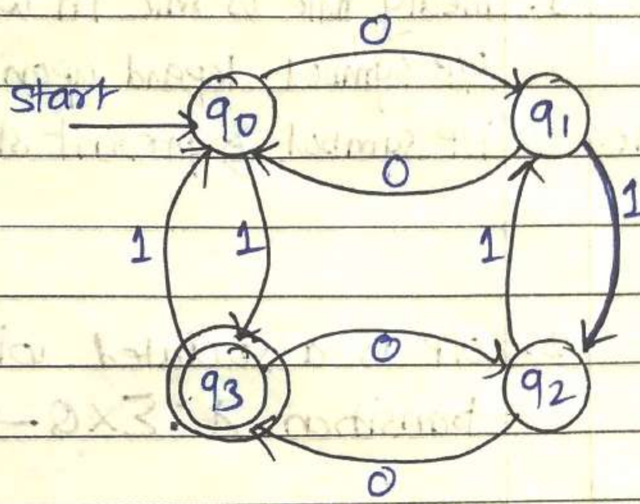
The transition table can be -

input / state	a	b	c	ϵ
q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
q_2	ϕ	ϕ	q_2	ϕ

Q.1 c) Design finite Automata (FA) for accepting strings over $\Sigma = \{0,1\}$ with even nos 0's & odd no's of 1's.

4-Marks

→ The finite automata will be as follow:

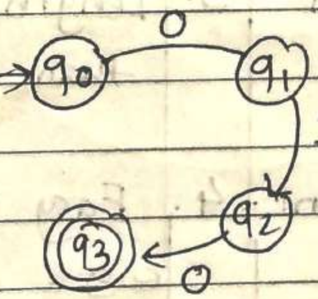


q_0

F.A.

As start from q_0

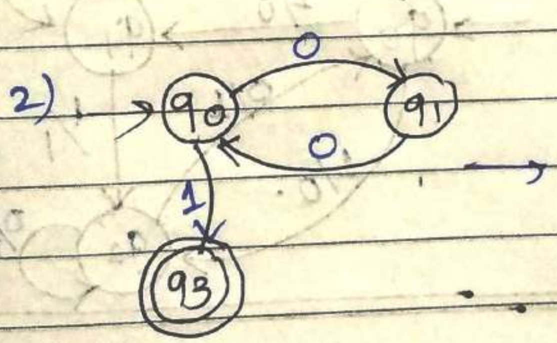
1)



we get even 0's and odd 1's.

→ 0 1 0 //

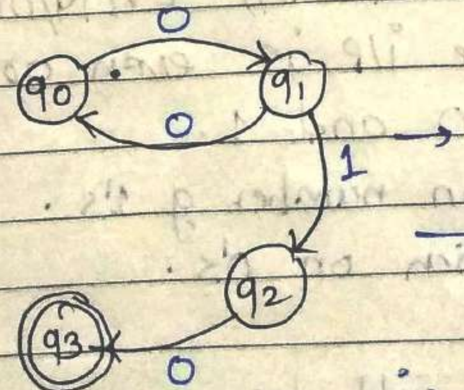
2)



→ Here, we get even 0's and odd 1's

0 0 1.

3)



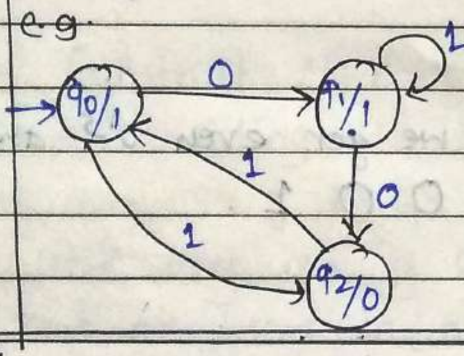
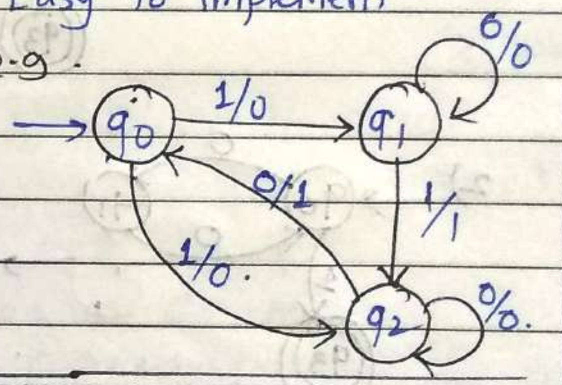
Here, we get even 0's (0000) & odd 1's.

→ 0 0 0 1 0

4 0's & 1 1's.

even odd

Q.2 a) Compare moore & Mealy machine 2-Marks

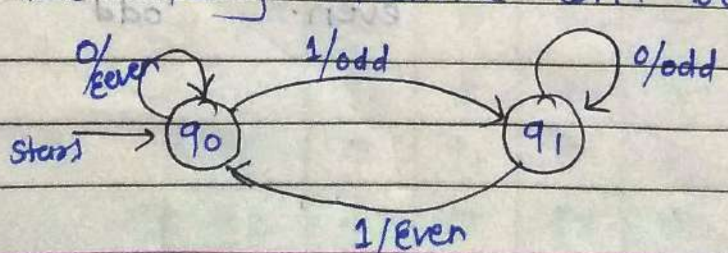
Sr No	Moore Machine	Sr No.	Mealy machine
1.	Moore m/c is finite state m/c in which next state is decided by current state. & i/P symbol	1.	mealy m/c is m/c in which O/P symbol depend upon present i/P symbol & present state m/c.
2.	O/P is associated with state $\lambda : Q \rightarrow O$	2.	O/P is associated with transition $\lambda : \Sigma \times Q \rightarrow O$
3.	Length of moore m/c is one longer by mealy	3.	Length of mealy is shorter than moore
4.	Difficult to implement	4.	Easy to implement
5	e.g. 	5	e.g. 

b) Construct Mealy m/c which can O/P EVEN/ODD if the total No's of 1's in the i/P is even or odd. The input symbol are 0 and 1. 4-Marks

→ Logic: The restriction is on number of 1's. But there is no restriction on 0's.

Then,

The Mealy Machine will be.

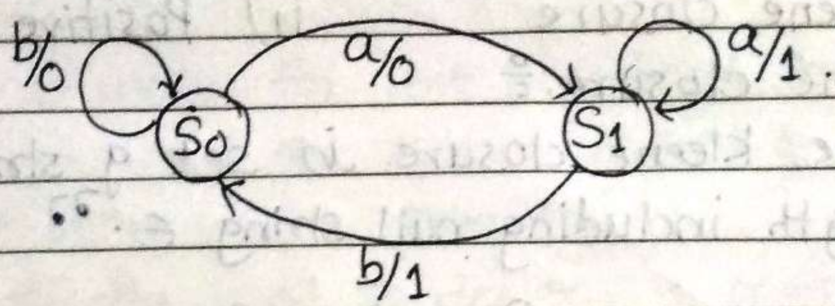


∴ As q1 state acepts odd number 1's

And

∴ q0 state accept even no's of 1's

Q.2. c) Convert the following Mealy to moore machine. 4-Marks



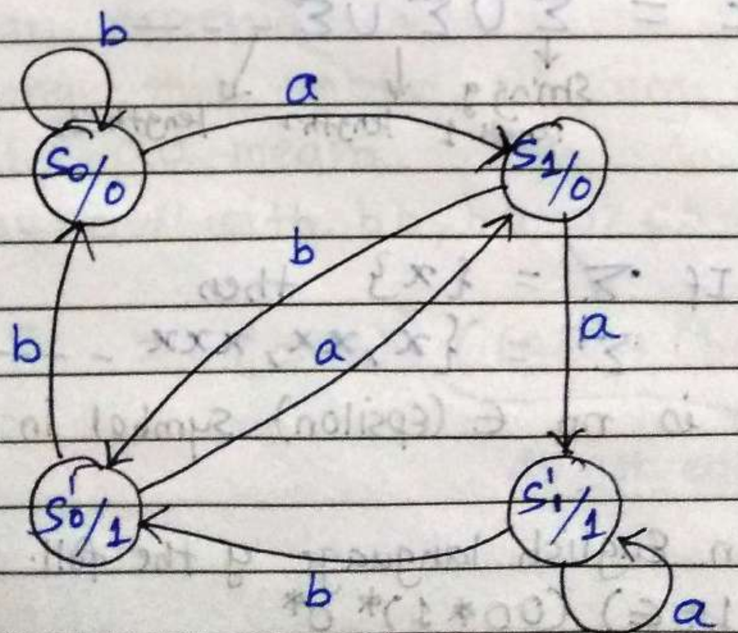
→ Logic :-

- If we see that there are 2 incoming edges for S0 one for i/p b and output 0 and
- other with i/p b and output 1.
- Similarly state S1 gets two incoming edges with o/p 0 and o/p 1.
- Hence, We have to split the state as

$S_0 \Rightarrow S_0$ and S_0' with o/p 0 and 1

$S_1 \Rightarrow S_1$ and S_1' with o/p 0 and 1.

The moore machine will be.



Q. 3 a) Define the following term: 2-Marks

i) Kleene closure ii) Positive closure.

→ i) Kleene closure :-

"The kleene closure is set of strings of any length including null string ϵ."

e.g. $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2$ ---

Σ^0 ↓ ϵ ; Σ^1 ↓ string of length 1 ; Σ^2 ↓ string of length 2.

e.g.

1. If $\Sigma = \{x\}$ then
$\Sigma^* = \{\epsilon, x, xx, xxx ---\}$.

ii) positive closure :-

"The positive closure Σ^+ can be defined as $\Sigma^1 \cup \Sigma^2 \cup \Sigma^3$ --- that means +ve closure consist of all strings of any length except null string

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3$ ---

Σ^1 ↓ string of length 1 ; Σ^2 ↓ length 2 ; Σ^3 ↓ length 3 (No Epsilon)

example :-

1. If $\Sigma = \{x\}$ then
$\Sigma^+ = \{x, xx, xxx ---\}$.

there is no ϵ (Epsilon) symbol in positive closure.

b) Illustrate in English language of the foll. regular expression

i) $(1+\epsilon)(00^*1)^*0^*$ - 2 Marks.

→ The expression generates the string as
$\{\epsilon, 1, 101, 1001, 10, 100, 1010, ...\}$

- This regular expression generates all string in which every 1 is seperated by one or more zero.
- It also accept string which has only 1 or the empty string. ["A lang over $\Sigma(1,0)^*$ in which 2 1's are never together"]

Q.3. b) ii) Explain in brief applications for regular expression.

→ Application :- — 2 Marks

1) it is useful for text processing & string processing.
2) it is used for pattern matching
3) it is used for parsing.
4) Basically used to design of compiler.
5) it can be used to perform all type of text search and text replace operation.
6) it is used for sorting & data validation.

Q.3 c) Determine the regular expression over $\Sigma = \{a, b\}$

i) All string that contains an even number of b's 4-Marks

ii) All string that do not end with 'aa'

→

i) The regular expression is

$$= (a^* b a^* b)^* \rightarrow \text{even no's of b}$$

ii) The regular expression is

The lang. that contains all string do not end with aa means

it may end with bb, ba or ab.

Hence.

$$R.E. = (a+b)^* \underbrace{((ab)^* + (ba)^* + b^*)}_{\text{do not end with aa.}}$$

Date.: / /

OR

Q.4. a) Justify if true or false the following 3-Marks.
Every subset g a regular language is regular.

→ **False**.

* Justification :-

1) - for i/p alphabets a and b, $a^* b^*$ is regular.
   - A DFA can be drawn for $a^* b^*$
   but a n b n for $n \geq 0$. which is subset g $a^* b^*$ is not regular
   As, we cann't define a DFA for it.

2) reason :- " Every language include those we know not to be regular, is subset g regular language Σ^* "

Q.4.

b) Explain the application g regular exprenion in GREP utilities in UNIX. 3-Marks

→ Applications :-

1) In unix, there is GREP utility, for searching desired pattern in the file. (text or string)
2) for searching pattern from the Grep makes use g regular exprenion.
3) When particular string is present in the file, then using GREP we get the line containing that string from the file.

For example :-

If we type GREP command on unix prompt then we will get o/p as follow.

$ grep hello test.txt

o/p → hello friends

test.txt
1 PVG
2 hello friends
3 college

As, grep cmd search hello keyword in test.txt file. if it get it on specific line. Grep displays complete line with searched keyword

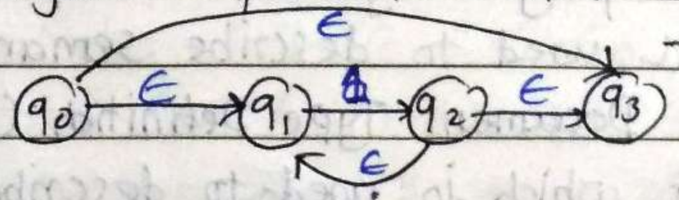
Date: / /

Q.4. c) Construct minimized DFA accepting language represented by regular expression $0^* 1^* 2^*$.

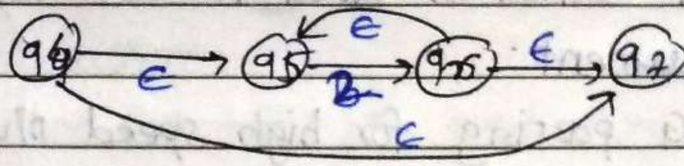
Convert given regular expression to NFA with ϵ moves

$\longrightarrow$ — 4 Marks.

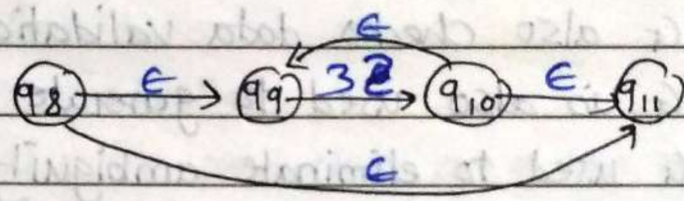
Step 1 : For a^*



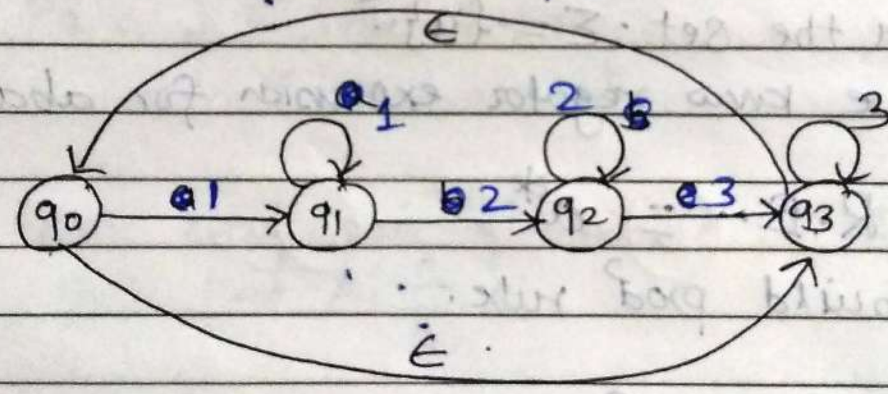
For b^*



For c^*



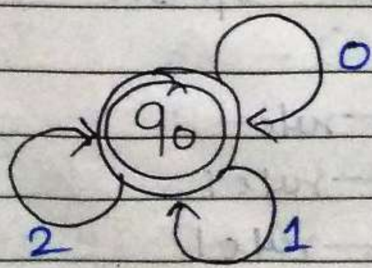
Step 2 : Remove ϵ from step 1.



Step 3 : The DFA will be.

For R.E.

~~$0^* \; 2^* \; 3^*$~~ $0^* \; 1^* \; 2^*$.



Q.5. a) Discuss application of CFG in XML

→ 1) CFG can be used to describe a statement of a prog^n lang. or markup lang i.e. HTML, XML. 3-Marks

2) XML used to describe semantic of text

3) The Document Type Definition (DTD) is a kind of CFG which is used to describe structure of XML document.

4) CFG parsing for high speed n/w application.

5) CFG also checks data validation for XML

6) CFG is also used to generate pattern of string

7) CFG used to eliminate ambiguity of grammar.

b) Construct the CFG for language having any no's of a's over the set $\Sigma = \{a\}$.

→ As we know regular expression for above lang

$R.E. = a^*$

Let us build prod^n rule.

$S \rightarrow aS$ Rule-1

$S \rightarrow \epsilon$ Rule-2

Now, if want 'aaaaa' string to be derived we can start with start symbol

S

aS — rule 1

aaS — rule 1

aaaS — rule 1

aaaaS — rule 1

aaaaaS — rule 1

aaaaaϵ — rule 2

aaaaa // string accepted.

Q.5 c) Simplify the Grammar:

$S \to Ab$, $A \to a$, $B \to C|b$; $C \to D$, $D \to E$, $E \to a$

→ To simplify grammar:

1) Elimintate ϵ production from G & obtain G1
2) Eliminate unit prod from G1 and obtain G2.
3) Eliminate useless symbol from G2 and obtain G3.

Step 1: Eliminate ϵ production

But Grammar does not have ϵ production

$\therefore G_1 = G$.

Step 2: Eliminate unit production

i) Add all non-unit production of given grammar G1 to Production P2 of G2.

$$P_2 = \left\{ \begin{array}{l} S \to Ab, A \to a \\ E \to a \end{array} \right\}$$

ii) Locate every pair of variable (A_i, A_j) Such that $A_i \overset{*}{\underset{G_1}{\Rightarrow}} A_j$

1) (B, C) due to unit prod $B \to C$
2) (C, D) —— 11 —— $C \to D$.
3) (D, E) —— 11 —— $D \to E$
4) (B, D) due to (B, C) & (C, D)
5) (B, E) due to (B, C) (C, D) & (B, E)
6) (C, E) due to (C, D) & (D, E)

iii) Unit Prod are removed through expansion.

from step ii) $S \to Ab$, $A \to a$, $E \to a$

pair (B, C)	$B \to a$	there is chain $B \to C \to D \to E \to a$
pair (C, D)	$C \to a$	—— 11 —— $C \to D \to E \to a$
pair (D, E)	$D \to a$	—— 11 —— $D \to E \to a$
pair (B, D)	$B \to a$	—— 11 —— $B \to C \to D \to E \to a$
pair (B, E)	$B \to a$	—— 11 —— —— 11 ——
pair (C, E)	$C \to a$	—— 11 —— $C \to D \to E \to a$

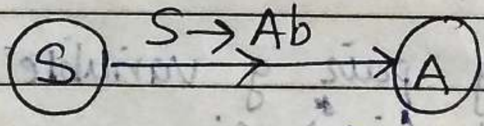
Thus, the production after elimination of unit production is :

$$P_2 = \begin{Bmatrix} S \to Ab \\ A \to a \\ E \to a \\ B \to a \\ C \to a \\ D \to a \end{Bmatrix}$$

<u>Step 3 :</u>

<u>Elimination of useless symbol.</u>

- every symbol in the grammar are generating.
- reachable symbol can be located by drawing a dependancy graph.



Two symbols S & A are reachable. B, C, D, E are non-reachable.

Find grammar G_3 with Production P_3 is obtained by deleting useless symbol

$$P_3 = \begin{Bmatrix} S \to Ab \\ A \to a \end{Bmatrix}.$$

//

OR

Q.6 a) Discuss application of CFG in syntax analysis of compiler.

→ Applications :

1) Context Free Grammar is used ~~to~~ for parsing the programming construct.
2) it is used for syntactical error from source program.
3) it is used to check syntax of program.
4) it is also used to create syntax tree of program.
6) it checks whether given program satisfies the rules or not.
7) lexical analyzer can identify token by using CFG.
8) CFG is helpful tool in describing the syntax of prog language.

b) Describe the language L for given ~~CG~~ CFG

$G = [\{S\}, \{a, b\}, P, \{S\}]$

where $P = \{S \rightarrow aSb, S \rightarrow ab\}$ 3-Marks.

→ Solution :-

$S \rightarrow aSb \mid ab$ is a rule. | it indicates 'or' operator.

$$S \rightarrow aSb$$

If this rule can be recursively applied then,

S

aSb.

$aaSbb$

$aaaSbbb$.

If finally we put $S \rightarrow ab$ then we get.

→ $aaaabbbb$.

Thus, we can have any nos of a's first then equal nos of b's

Hence, we can show language.

$$\{L = a^n b^n \text{ where } n \geq 1\}$$

Q.6 c) Optimize the CFG given below by reducing the grammar. where S is a start symbol

$S \rightarrow A \mid 0C1$

$A \rightarrow B \mid 01 \mid 10$

$C \rightarrow \epsilon \mid CD$.

→ solution

$S \rightarrow A \rightarrow B$ is a unit production.

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

$C \rightarrow CD$ B and D are useless symbol.

Reducing the grammar, we have to avoid above conditn

Let, $S \rightarrow A$.

i.e.

$A \rightarrow B$ is useless symbol, because B is not defined further.

$S \rightarrow 01 \mid 10$ → $A \rightarrow 01 \mid 10$ & ~~case~~.

$S \rightarrow 01 \mid 10 \mid 0C1$.

But But $C \rightarrow \epsilon$.

Hence, $S \rightarrow 01 \mid 10$.

$A \rightarrow B$ But we can remove this production since B is a useless symbol

Hence, $A \rightarrow 01 \mid 10$

$S \rightarrow 01 \mid 10$.

There is no A in the derivation of A so by considering A also as a useless symbol

We get final CFG as

$S \rightarrow 01 \mid 10$