Automaton (Singular)

CHAP-5 Finite Automata

1. Finite Automata:

The finite automaton is the mathematical model that has a set of states who control moves from state to state in response to external inputs. The control may be either deterministic meaning that the automation can't be in more then one state at any time or non-deterministic meaning that it may be in several states at once. This distinguishes the class of automata as deterministic finite automata (DFA). ~~The DFA can't be in more then one~~ or non-deterministic finite automata (NFA). The DFA can't be in more the one step at one time. The NFA can be in more then one state at the same time. The finite automata may be generalize output or it may not be.

The finite state machine are used in applications in computer science and data networking for example finite state machine is basic for programs for spell checking, indexing, grammer checking, searching large body of text, recognizing speech, transforming text using mark up languages such as XML and HTML and network protocol that specifies how computer communicate.

The finite state machine can be represented with state transition diagram or state transition table.

$\delta_1 = \delta_2 = 2$

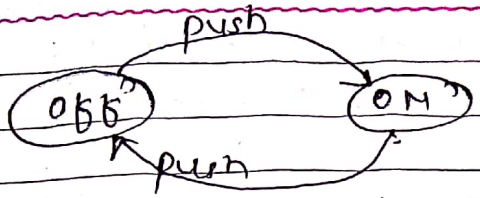


Fig. State transition Diagram

	PUSH
off	ON
ON	OFF

Fig: State transition table of switch.

A. Finite state automation with output is defined by six tuple as $(Q, \Sigma, O, \delta, \beta, q_0)$ where:

Q = Finite set of Status

Σ = Finite set of input alphabets

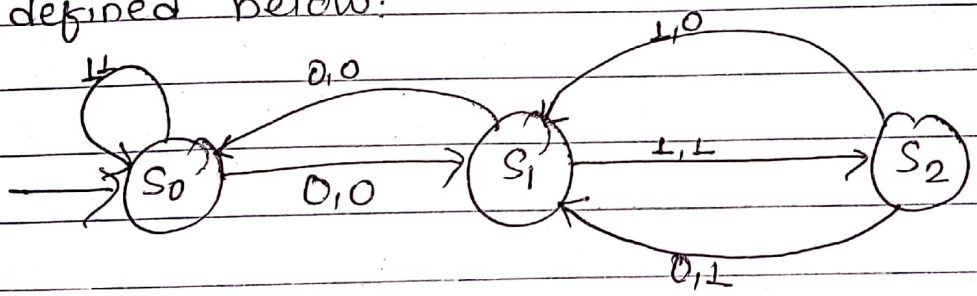
O = Finite set of output alphabets

δ = A transition function that maps $Q \times \Sigma \to Q$

β = An output function that maps $Q \times \Sigma \to O$

q_0 = A start state $q_0 \leq Q$

* Consider a finite state automaton with output as defined below:



State	δ Input		β Input	
	0	1	0	1
S_0	S_1	S_0	0	1
S_1	S_0	S_2	0	1
S_2	S_1	S_1	1	0

The output generated from the input strings 01110 for the above FSM is 01011 and the state transition sequence is :-

$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_1 \rightarrow S_2 \rightarrow S_1$

State Diagram of Binary Adder:

MEES

bit + bit | sum
format.

Date
Page No.

* State diagram of Binary Adder:

Consider a binary adder which adds two 2-digit binary strings. The addition is performed 'bit by bit' starting from the least significant bit (LSB). Note that a carry in from the addition of privious bits should be taken into account. There are two bits in the input with possible combinations are 00, 01, 10 & 11.

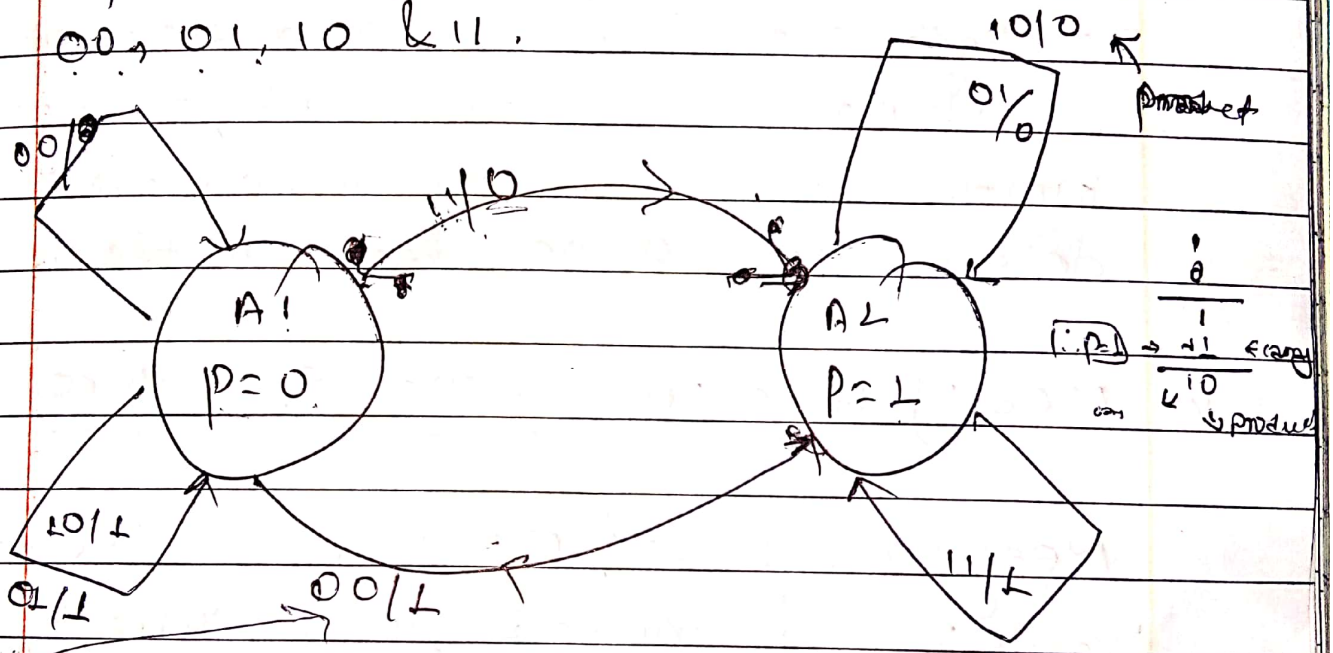


Fig. State transition diagram of Binary adder.

To represent this finite state machine (FSM) let us construct two internal state.

A1 → An addition without carry in (P=0)

A2 → an addition with carry (p=1)

* Finite State machine without output:

The finite state machines with no output are often known as finite state automata. It consist of finite set of state, finite input alphabets, a transition function that assigns a next state to every pair of state and input, An initial or start state and a subset consisting of final or accepting state. The finite state automata can be either deterministic or non deterministic.

* Mealy machine & moore machine:

Mealy machine is FSM whose output value are determined by its current state and current inputs. In this contrast moore machine whose output values are determined by only its current state. Thus, the moore machine is finite state machine whose output values are determined by its current state only. The finite state machine with output are the concept of

Date
Page No.

mealy machine.

A moore machine defined with following transition diagram:

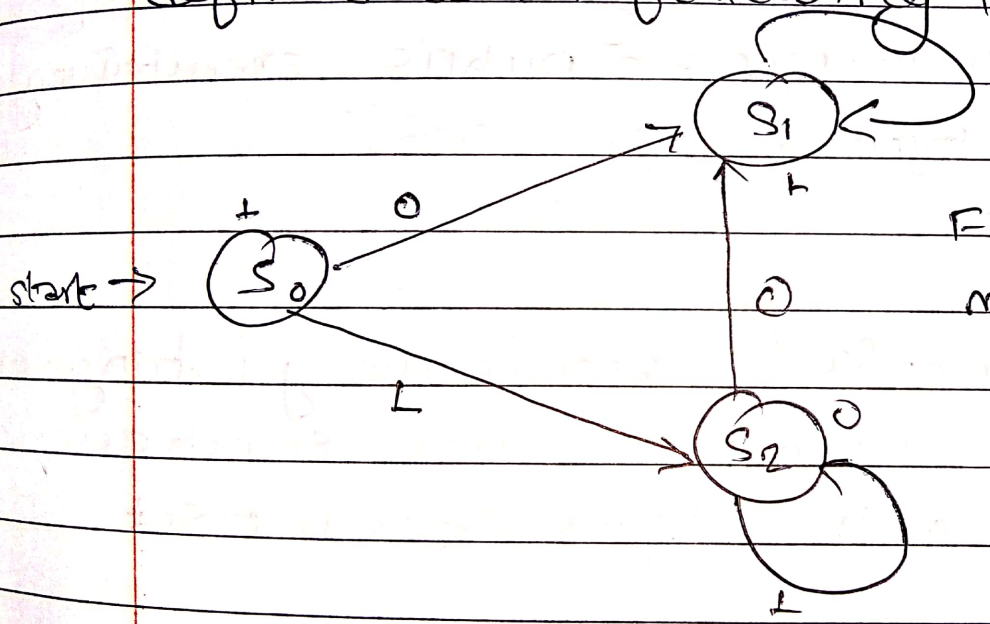


Fig. example of moore machine

MRCS

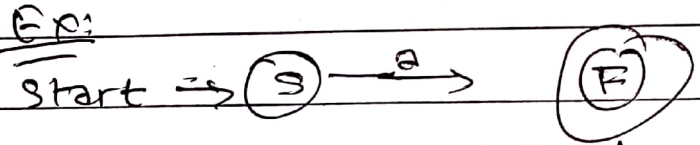
Date
Page No.

Deterministic Finite Automata

A deterministic Finite Automata (DFA) is defined by ~~quin~~ quin tuple (5-tuple) as $(Q, \Sigma, \delta, q_0, F)$ where,

Q = Finite set of states
Σ = finite set of input alphabets
δ = A transition function that maps $Q \times \Sigma \Rightarrow Q$
q_0 = A start state.
F = set of final state(s) ($F \subseteq Q$)

Ex:



DFA having start state S and final state F for accepting input a.

General Notation of Deterministic Finite Automata

There are two Prefered Notation for describing this class of automata.

1) transition table:

Transition table is a conventional tabular representation of the transition function ~~(δ)~~ delta(δ) that make the arguments from $Q \times \Sigma$ and returns a value which is one of the state of the automaton. The row of

Teacher's Signature.....

the table ~~postponed~~ crosspond to the state while column corosspond to the input symbol. The starting state in the table is represented ~~followed by~~ followed by the state 'i' ie → q for q being start state. whereas the final state as

* q for q being final state. The entry for a row corresponding to state q and the column corresponding to input 'a' is the state $\delta(q, a)$.

Consider a DFA

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

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

$q_0 = q_0$

$F = \{q_0\}$,

$\delta = Q \times \Sigma \to Q$

The transition table for the above DFA is as follows:

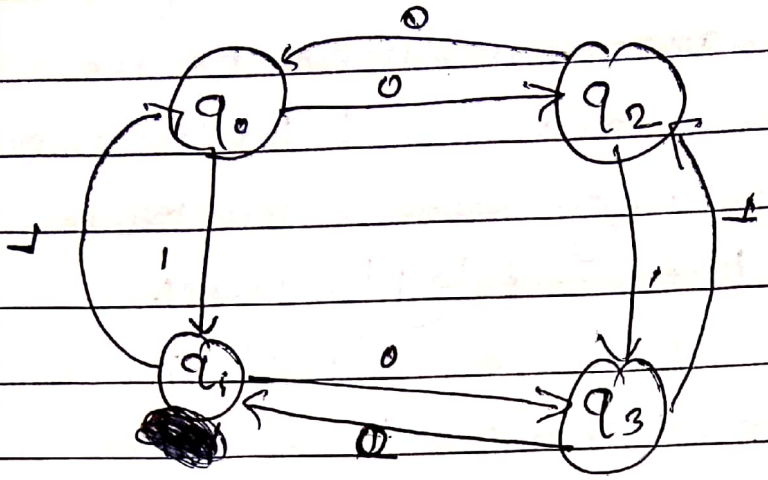
	0	1
$\to q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Fig. State transition table.

The DFA accepts string having both an even number of 0's and even number of 1's.

Ultra White

Teacher's Signature.....



* State transition Diagram:

A transition diagram of a DFA is a graphical representation where,

For each State Q, there is ~~a~~ a node represented by circle.

→ For each state q in Q and even input in Σ, if $\delta(q, a) = p$ then there is an ~~one~~ arc from q to p labeled 'a' in the transition diagram. If more than one input symbol cause the transition from state q to p then arc from q to p is labeled by a list of those symbols.

The start state is labeled by an arrow written with start on the node.

The final state is the accepting state is marked by double circle.

* Construct DFA that recognizes each of these language.

a) A set of strings that begin with two zeros over $\Sigma = \{0, 1\}$

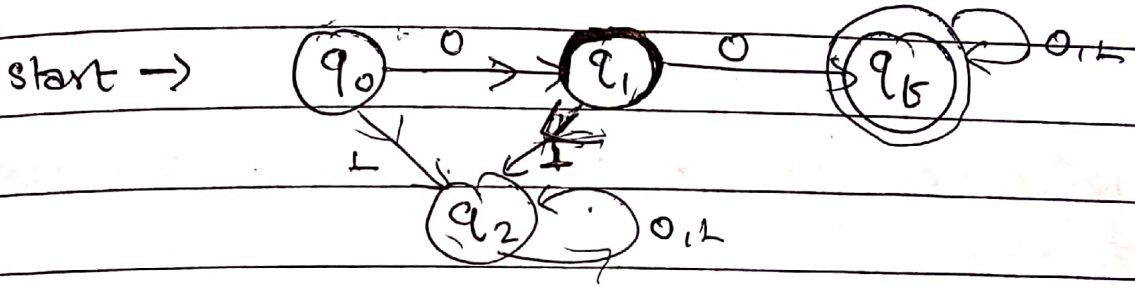
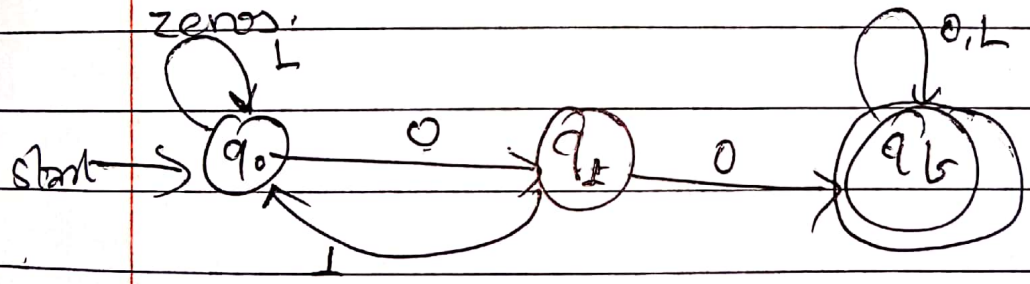
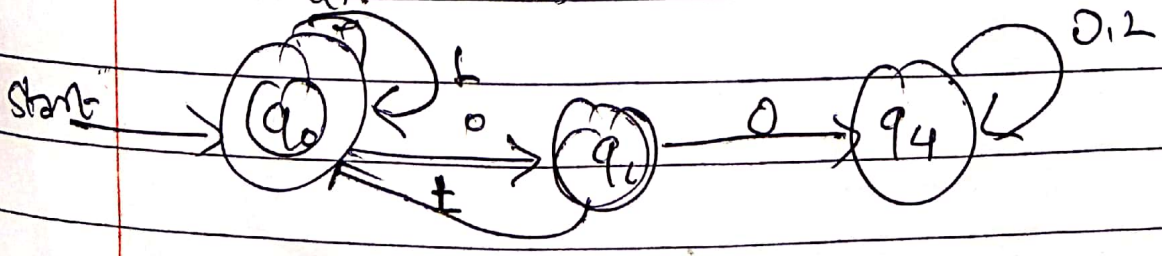


Fig. State transition diagram of DFA that accept, the set of strings begin with two zeros.

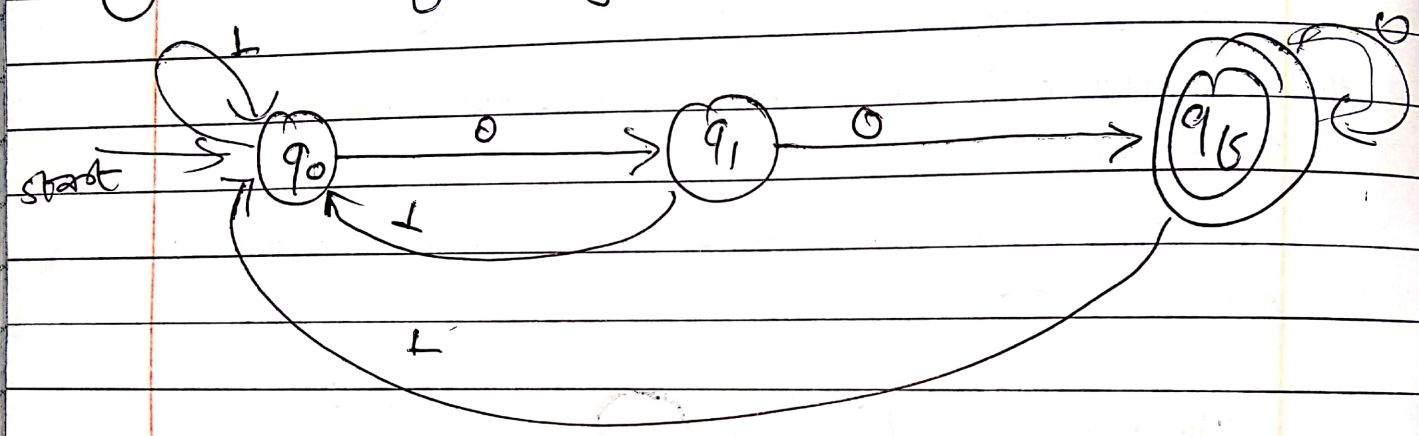
(b) The set of strings that contains two consecutive zeros.



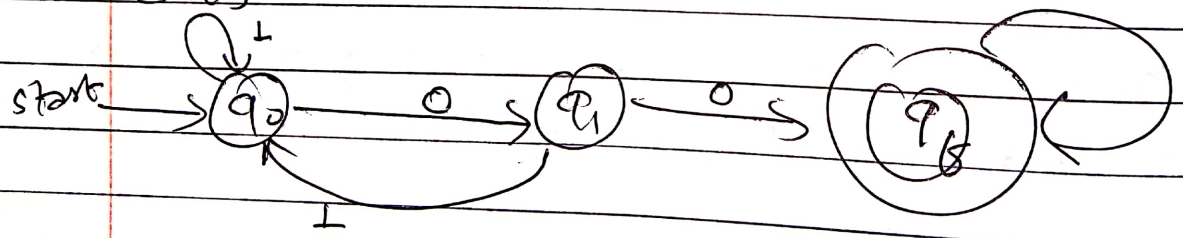
(c) The set of bit strings that do not contains to consecutive zeros.



(d) The set of strings that ends with two zeros.



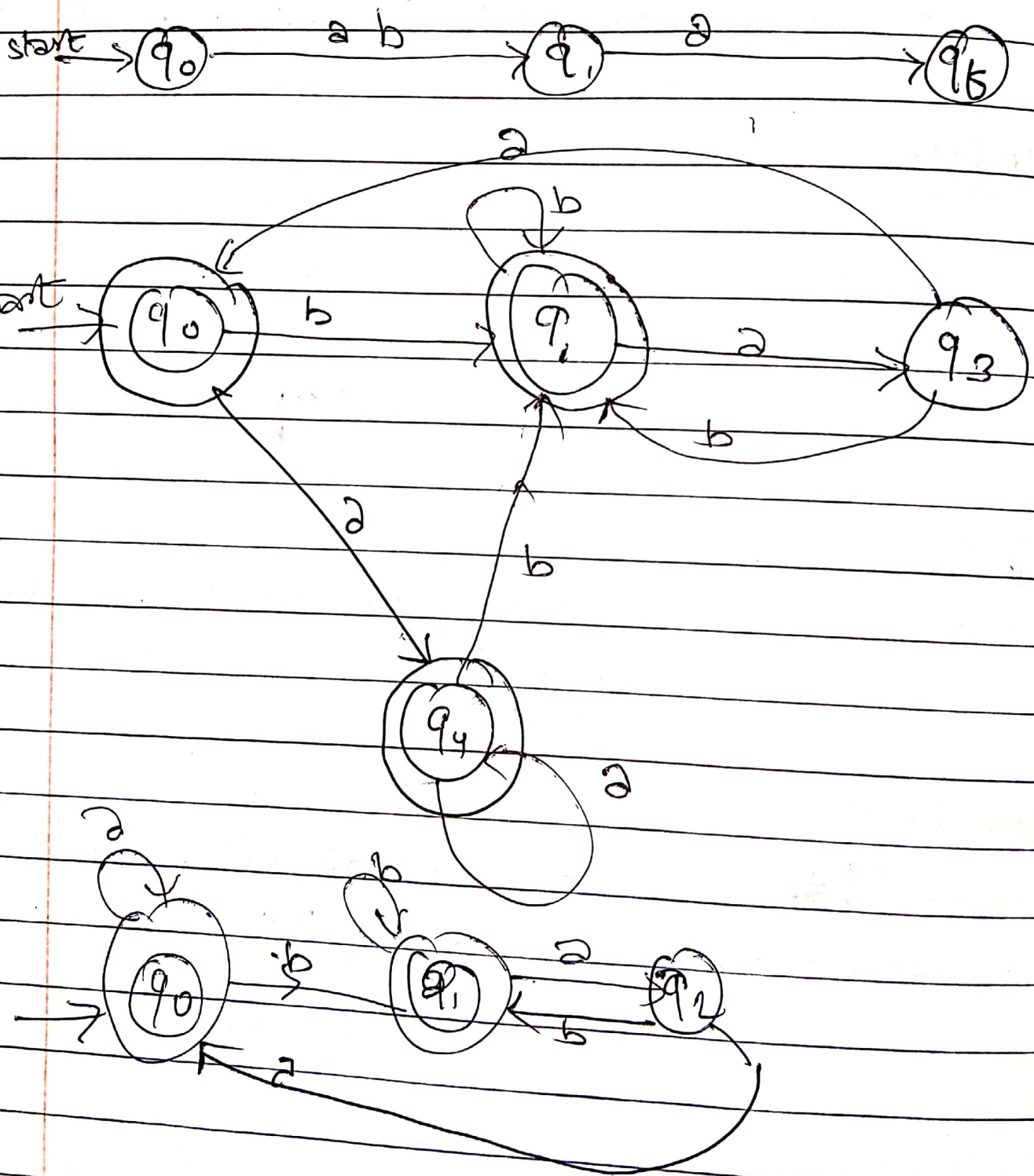
(e) The set of strings that contains at least two zeros.



Date
Page No.

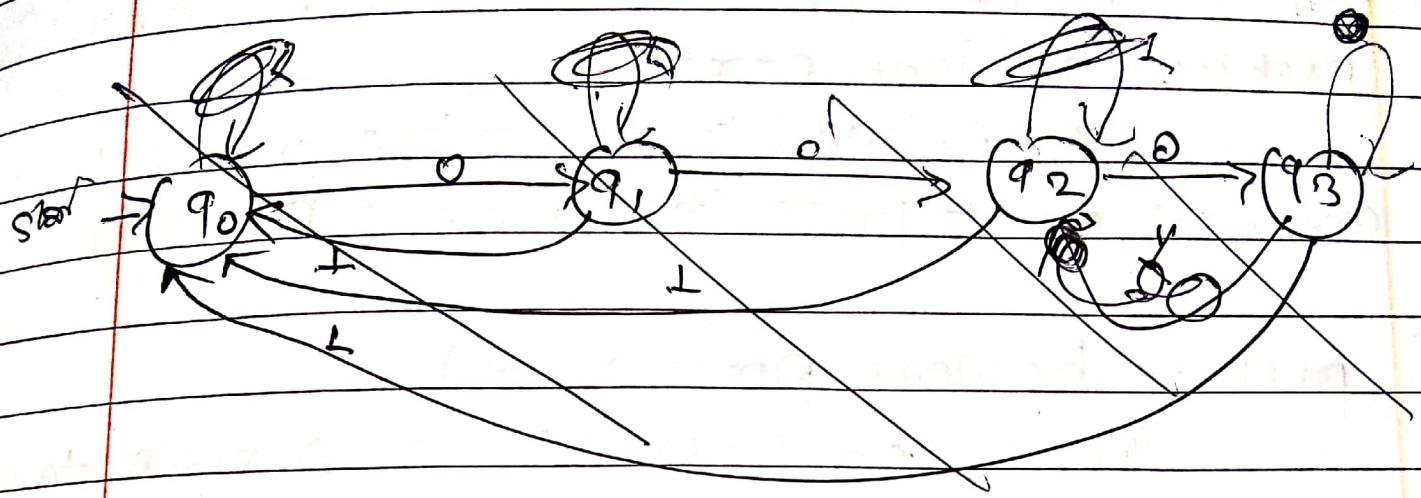
MFG

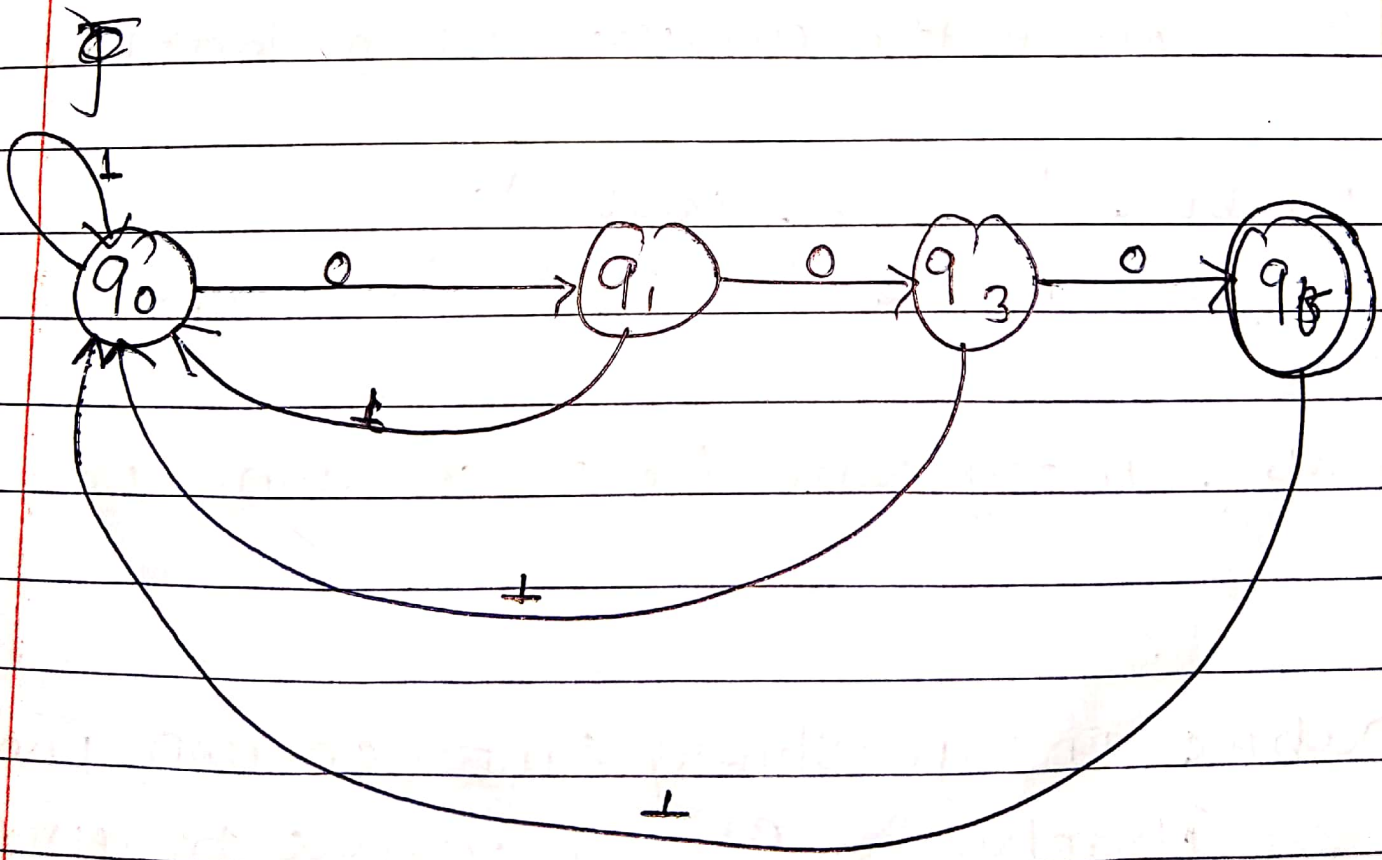
f) That accepts all strings over $\Sigma = \{a, b\}$ that donot end with 'ba'.



Date
Page No.

Q. That accepts all strings over $\Sigma = \{0, 1\}$ ending with three consecutive 0s.

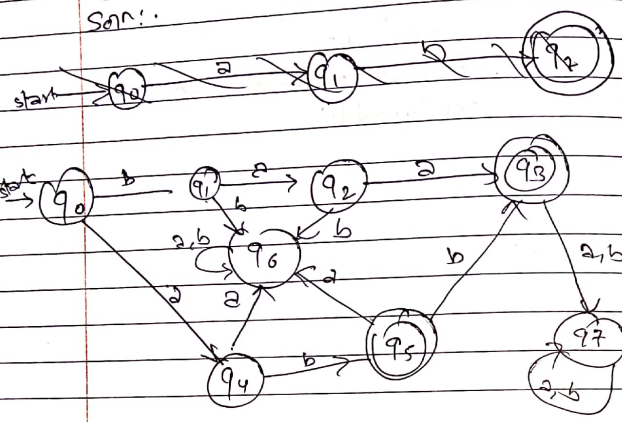




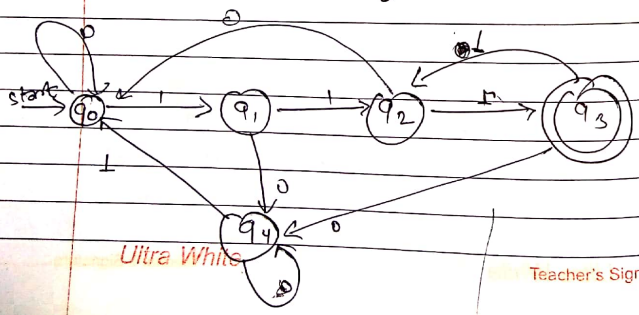
MFCS

Q. Construct a DFA over $\Sigma = \{a, b\}$ accepting {baa, ab, abb}.

Soln:



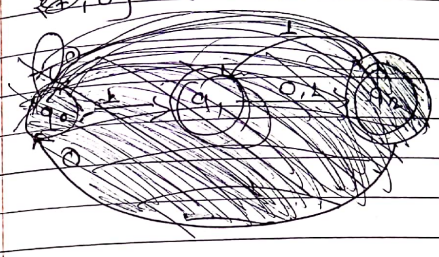
Q. Construct a DFA over $\Sigma = \{0, 1\}$ accepting set of strings ending with 111 that contains odd no. of 1s.

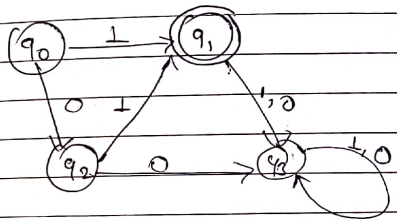


Ultra White

Teacher's Signature.....

Construct a DFA over $\Sigma = \{0, 1\}$ accepting {1, 01} Σ {1, 01}





Ultra White

Teacher's Signature.....

MFCS

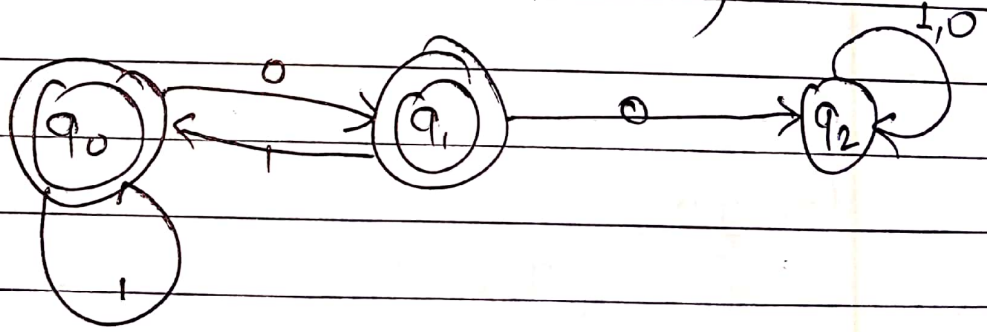
Date
Page No.

Extended transition function of DFA:

The extended transition function of DFA, denoted by $\hat{\delta}$ is a transition function that takes two arguments as input, one is state q of Q and other is string $w \in \Sigma^*$, and generates a state $p \in Q$. This state p is that automation reaches when starting in state q when processing the sequence of inputs w.

i.e. $\hat{\delta}(q, w) = p$

Example: Given a DFA as below; compute $\hat{\delta}(q_0, 101)$ & $\hat{\delta}(q_0, 1011)$



Now, $\hat{\delta}(\hat{\delta}(q_0, 10), 1) = \delta(\hat{\delta}(q_0, 10), 1) =$
$\delta(\delta(\hat{\delta}(q_0, 1), 0), 1) = \delta(\delta(\delta(\hat{\delta}(q_0, \varepsilon), 1), 0), 1)$
$= \delta(\delta(\delta(q_0, 1), 0), 1)$
$= \delta(\delta(q_0, 0), 1)$
$= \delta(q_1, 1)$
$= q_0$ (∴ the string is accepted)

Ultra White

Teacher's Signature.....

Language of DFA:

The language of a DFA is set of all strings w that take DFA starting from start state to one of the accepting state. The language of DFA is called regular language.

i.e. $L(M) = \{w \mid \hat{\delta}(q_0, w) = p \in F\}$

Date
Page No.

Application of DFA:

Deterministic finite automata have many practical application as described below:

i) Almost all compiler and other language processing system use DFA, like C, to divide an input program into token like identifiers, constants, an keywords and to remove comments and white space.

ii) For many application that accept typed commands, the command language is quite complex, almost like a little programming language. such application use DFA to process the input command.

iii) Text processor often used DFA to search a text file for string that match a given pattern. this include must version of unique tools like AWK, Egrep, and preekmail along with number of platform independent system such as mysql. Speech processing and other signal processing system often used a DFA like technique to transform and incoming signal.

(5) Controller used DFA like remiare to ~~track~~ ~~dog type~~ track the current stage of a wide variety of finite state system for

industrial processes to video games. They can be implemented hardware or software. Sometimes the DFA like code in some application is generated automatically by special tools such as "lex".

MFCS

Non-Deterministic Finite Automata(NFA)

The finite automata in which the exact state to which the machine move, cannot be determine is called Non-Deterministic finite automata (NFA).

A non-deterministic finite automata is a mathematical model that consist of

Teacher's Signature.....

1) A finite set of states, Q.
2) A finite set of input symbols, Σ.
3) A transition function that maps state symbol pair to sets of states, δ
4) A state $q_0 \in Q$, that is the start state.
5) A set of final state F, $F \subseteq Q$

Thus, NFA can also be interpreted by quintuple $(Q, \Sigma, \delta, q_0, F)$ where $\delta = Q \times \Sigma = 2^Q$

Here, The power set 2^Q has been taken because in case of NFA, from a state, transition can occur to any combination of state.

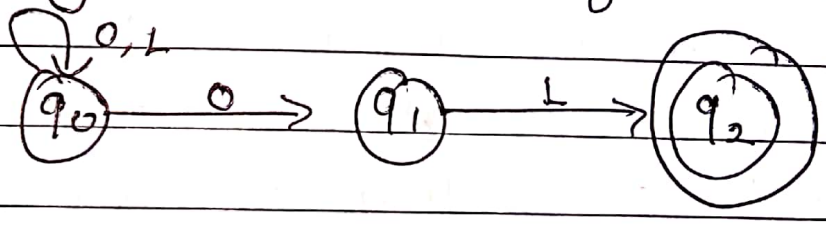


fig. NFA accepting a string that end in 01.

* Difference between NFA & DFA:

DFAs and NFAs accept exactly the same set of languages, that is non-determinism doesn't make NFA more powerful. Any language accepted by NFA can also be accepted by some DFA and

Vice-Versa. Thus, NFA & DFA has same capability.

The only difference betn NFA & DFA is found in transition function. A NFA's transition function is ~~ben re~~ less restrictive than DFA's because it allows to have several transition from a state to 0, 1 or more state for the same input symbol. On other hand, a DFA specifies exactly one state that may be entered for a given state.

Thus in DFA, for a given state, on a given input, we reach to determinisht'e and unique state but in NFA we may lead to more than one state for a given input. A DFA is a subject of NFA. Every NFA have its equivalent DFA.

* Regular Expression:

Regular expression are those algebric expression used for representation of regular language, the language of finite Automata.

Date
Page No.

Regular expression after a descriptive way to express the string we want to accept by some FA. This is what the regular expression (RE) offer that the automata doesn't.

A regular expression is built up out of simpler regular expression using a set of defined rules. Each regular expression 'r' denotes a language $L(r)$ the defining rules specifies how $L(r)$ is formed by combining in various way the language denoted by the subexpression of r.

Let, Σ be an alphabet, the regular expression over the alphabet Σ^n are defined inductively as follows:

1) $\emptyset$ is a regular expression representing empty language.
2) ε (Eliplion) is a regular expression representing the language of empty string.
3) If 'a' is a symbol is Σ then 'a' is a regular expression representing the language 'a'.

4) i) If r and s are the regular expression representing the language $L(r)$ & $L(s)$ then r∪s is a regular expression representing the language $L(r) \cup L(s)$.

ii) r.s is a regular expression denoting the language $L(r) . L(s)$.

iii) r^* (r closer) is a regular expression denoting the language $[L(r)]^*$.

iv) (r) is a regular expression denoting the language $[L(r)]$.

MFCS

Date
Page No.

Regular Operators:

Basically there are three operators that are used to generate the language that are regular. They are:

(1) union: If L_1 and L_2 are any two languages then, $L_1 \cup L_2 = \{s / s \in L_1 \text{ or } s \in L_2\}$

ex:

$L_1 = \{00, 11\}$, $L_2 = \{\varepsilon, 10\}$

Then,

$L_1 \cup L_2 = \{\varepsilon, 00, 10, 11\}$

(2) Concatenation: If L_1 and L_2 are any two languages:

$L_1 . L_2 = \{x_1 . x_2 \mid x_1 \in L_1 \text{ & } x_2 \in L_2\}$

(3) Kleen closure (*) and positive closure (+):

If L be any language then kleen closure of L is

$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup \ldots\ldots$$

and positive closure of L is

$$L^+ = L^* - L^0$$

$$= L^1 \cup L^2 \cup L^3 \cup \ldots$$

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

$L^0 = \{\}$ ε (empty)

$L^1 = \{a, b\}$

$L^2 = \{aa, bb, ab, ba\}$

$L^3 = \{aaa, aab, bbb, \ldots\}$

$L^0 \cup L^1 = \{\varepsilon, a, b\}$

Date

Page No.

* Precedence of Regular operator

1) Closure (*) has highest precedence.
2) Concatenation (.) has next highest precedence.
3) Union (∪) has lowest precedence.

Regular language

Let, Σ be an alphabet, the class of regular language over Σ is defined inductively as:

- ϕ is a regular language representing the empty language.
- $\{\}$
- $\{\varepsilon\}$ is a regular language representing language of empty string.
- For each Σ, $\{a\}$ is a regular language.
- If $L_1, L_2, \ldots, L_n$ are regular languages then $L_1 \cup L_2 \cup \ldots \cup L_n$ is also

Ultra White

Teacher's Signature.....

regular languages.

- If $L_1, L_2, L_3 \ldots\ldots L_n$ are regular languages then so is $L_1 . L_2 . L_3 \ldots\ldots L_n$ is also regular langu
- If L is a regular language then so is L^*.

Every language can be written using as an expression using the operators, union, concatenation and Kleen closure.

* Application of Regular Expressions

1) Validation:

Determining that ~~String~~ a string complies with a set of formatting ~~constants~~ constraints like email address validation, password validation etc..

(2) Search and selection:

Identifying a subset of items from a larger set on the basis of ~~pattern~~ pattern match.

(3) Tokenization:
Converting the a sequence of character into words, tokens (like keyword identifier) for later interpretation

* Similarities in Regular Expression and Automata.

An automaton defines a pattern namely the set of string labeling paths from a initial state to some accepting state in the graph of automaton. Regular expressions (RE) are an algebric way to define those pattern. Regular expression are analogus to the algebra of arithemetic operation. Interestingly the set of patterns that can be expressed in the RE algebra is exactly the same set of patterns that can be described by automata.

A expression is one way of describing a finite state automaton (FSA)

Teacher's Signature.....

Both ~~regular~~ RE and finite automaton are used to describe regular languages. RE and finite automaton have equivalent ~~explicit po~~ expressive power.

(I) For every RE (R) there is a corresponding finite automaton that accepts the set of strings generated by R.

(II) For ~~a~~ Finite automaton (A) there is a corresponding ~~expression~~ that generate the set of string accepted by 'A'.

MFCS

Date
Page No.

* Construct a DFA with transition that accepts the multiple of 3. $\Sigma\{0,1\}$

language of DFA = string that are multiple of 3. = {0, 11, 110, 1001, 1100, ...}

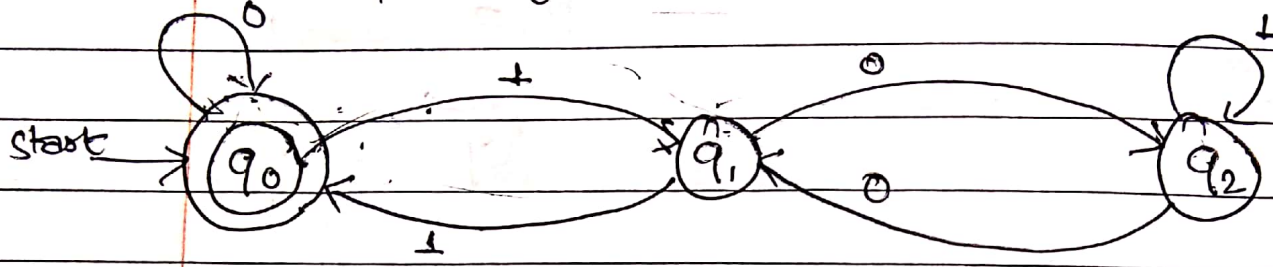


Fig. Transition Diagram of DFA accepting the strings which are multiple of 3.

States	Inputs	
	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

Fig. S.T table of DFA accepting the strings which are multiple of 3.

* Graph coloring:

A coloring of simple graph is the assignment of color to each vertex of the graph so that no two adjacent vertex are assigned the same color. A graph can be coloured by asigning a different color to each of its vertices. For most

Teacher's Signature.....

graph a coloring can be found that uses fewer color than the number of vertices in the graph.

The chromatic no. of graph is the least no. of color needed for the coloring of graph. The chromatic no. of graph G is denoted by $\chi(G)$ (chi). (Here χ is the Greek letter chi).

The chromatic no. of a planer graph is no more than 4.

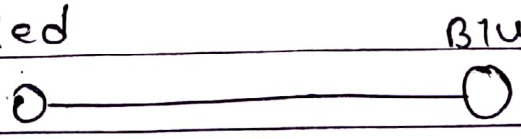
* problem related to the coloring of maps of region s can be solved using graph coloring method with a small numbers of colors.

A coloring of

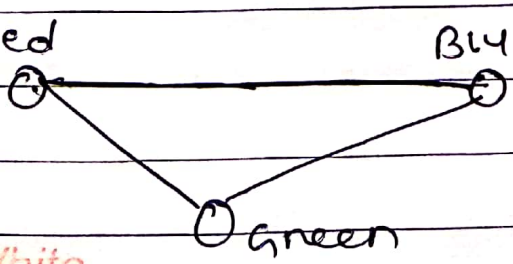
* what is the chromatic number of K_n?

⇒ A coloring of K_n can be constructed using 'n' colors by assigning a different color to each vertex.

(a)

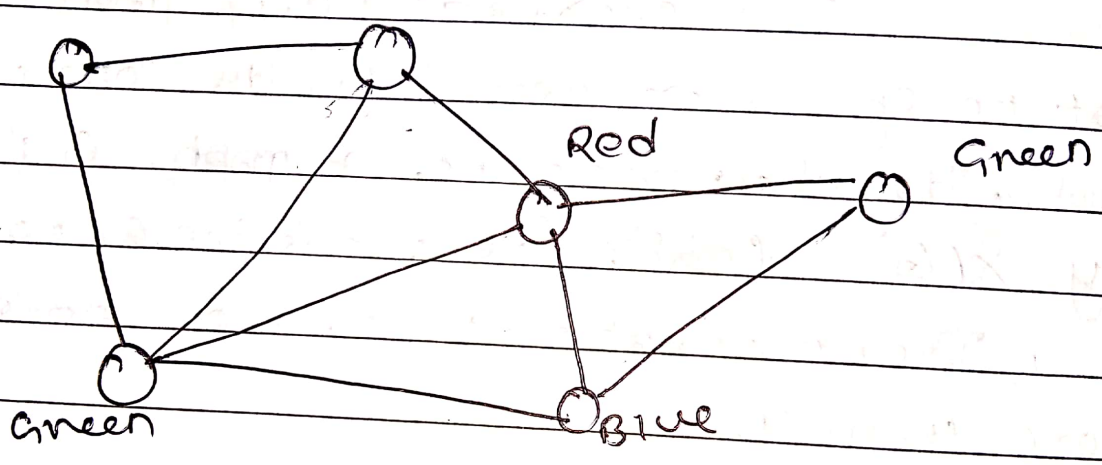


(b)



c) Red Green Red

d) Red Blue Red Green Green Blue



$\chi(C) = 3$

Languages and Grammers

Date
Page No.

* Natural language & Formal language:-

Natural language are the language that people speaks such as Nepali, English, Hindi etc. They were not designed by people. Although, people try to impose some order on them, they evolved naturally. It doesn't seem possible to specify all rules of syntax for a natural language.

The formal language unlike a natural language is specified by well defined set of rule of syntax. Formal languages are languages that are designed by people for specific applications. A formal language is set of string of symbols that may be constrained by rules that are specific to alphabets. Sometimes the set of words are grouped into expression, where as rules & constraints may be formulated for the creation of well formed expression.

* Syntax:

A syntax or grammer for a language 'L' is a set of rule through which the strings of 'L' may either be generated or through which any elements of ~~A~~ A^* can be

Teacher's Signature.....

determined to be an element of $S(A^*)$. The syntax of a language determines the forms of a sentences in the language.

Semantics is the structure of a language L that gives it meaning relating it to its modal. It is an interpretation of a language 'L' interms of some modal (M).

Phrase Structure Grammer

phrase structure grammer is a type of generating grammer consisting one of the rule determining the structure and interpretation of sentences. phrase structured grammers are used to generate the words of a language and to determine whether a word is in a language.

Formal language which are generated by grammers, provide modals for both natural languages such as ~~english~~ english and for programming language such as C & Java.

Date
Page No.

A Grammer G is defined by 4-type (VTPs) where,

V = a vocabulary
T = set of terminal symbols
P = set of production rules.
S = start symbols and $S \in V$.

For examples: a phrase structure grammer defined by $G = \{V, T, P, S\}$ where,

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

$T = \{a, b\}$

S is the start symbol and

$P = \{S \to ABa,\ A \to BB,\ B \to ab,\ B \to b\}$

$S = ABa$

$= BBa = ababa$ or bba

Q. what is the language of following Grammer?

$S \to \varepsilon$

$S \to 0S1$

(i) $S \to \varepsilon$

(ii) $S \to 0S1$

$\to 01$

∴ The language of this grammer

Teacher's Signature.....

(iii) $S \to 0SL$
$\to 00S11$

(iv) $S \to 000111$

This is a grammer representing the language over $\Sigma = \{0, 1\}$ which has equal number of zeros followed by equal no. of 1.

* Construct a grammer representing a language over $\Sigma = \{a, b\}$ which is palindrome language.

Let, $G = \{V, T, P, S\}$

Given that, $T = \{a, b\}$

S is a start symbol

$V = \{a, b, S\}$

and,

P is defined as:

$S \to \varepsilon / a / b$ (खाली)

$S \to aSa / bbb$

* Types of phrase structured Grammers:

phrase structured Grammer can be classified according to the types of production that are allowed. Noam chomsky has define the phrase structured grammar with

Teacher's Signature.....

Date
Page No.

following hierrachey.

(1) Type - 0 (unrestricted) Grammar :

Type-0 Grammer (unrestricted grammer) include all formal grammer. They generate exactly all languages they can be recognized by a Turing machine. These language are also known as recursively enumerable languages. An unrestricted grammer is a formal grammer on which no restriction are made on the production. The production rule is defined as $\alpha \to \beta$ where α & β are string of terminals and non-terminals and α is not the empty string.

example :

$S \to Abc$

$Ab \to c$

$cC \to Sc | \varepsilon$

(2) Type - 1 (context sensitive) Grammer :

↳ Non-deterministic turing machine

Type-1 Grammer (context sensitive Grammers) generates context sensitive language. A context sensitive ~~to~~ Grammer is a formal Grammer. $G = (V, T, S, P)$ where V is the set of non-

Teacher's Signature.....

terminals, T is the set of terminals.
P → Set of production rule of the form
$\alpha A \beta \rightarrow \alpha \gamma \beta$ where,
α and β are strings of terminals and non-terminals and S is specially designed start symbols. The string α and β may be empty but γ must be non-empty:
eg.

$S \rightarrow aSC / aBC$
$CB \rightarrow HB$
$HB \rightarrow HC$
$HC \rightarrow BC$
$aB \rightarrow ab$
$bB \rightarrow bb$
$bC \rightarrow bc$
$cc \rightarrow cc$

(3) Type-2 (context free) Grammar:-

Type-2 Grammers are the context free grammar & generate the context free languages. A context free grammar is the formal grammar $G = (V, T, S, P)$ where
V is the set of non-terminals symbols.
~~Set of~~
T = Set of terminal Symbol

Teacher's Signature.....

P = set of production rules
S = specially designed start symbol of the form $A \rightarrow \beta$ where β is the set of terminals and non-terminals.

The language generated by context free grammar are exactly all languages that can be recognized by a non-deterministic push down automata.

Ex: The CFG for balance parenthesis over $\Sigma = \{(,)\}$ can be defined by the following production rules:

$S \rightarrow SS$
$S \rightarrow (S)$
$S \rightarrow ()$

4) Type 3 (Regular) Grammar :-

Type-3 Grammar (Regular Grammar) generates the Regular Grammar. Such a grammar restricts rules to a single non-terminal on the left hand side and a ~~single~~ right hand side. Consisting a single terminal followed a single non-terminal (right regular) actively the right hand side of a grammer can consist of single terminal

possibly preceeded by a single non-terminal (left regular).

This language are exactly all languages that can be accepted by a finite state automata.

A regular gramar is a formula grammar $G = (V, T, S, P)$ where

V = non terminal symbols - set

T = terminal symbols set

S = specially designed start symbols

P = production of one of the following form.

$S \to a$	$S \to b$
$S \to aA$	$S \to Bb$
$S \to \varepsilon$	$S \to \varepsilon$
(Right regular)	(left regular)

Summary

Grammar

Summary:

Grammar	Language	Automation	Production
Type - 3	Regular language	Finite state Automata	$A \to a$, $A \to aB$, $A \to \varepsilon$
Type - 2	context free	Non-deterministic pushdown automata	$S \to SS$, $S \to (S)$, $S \to ()$
Type - 1	context Sensitive	Non-deterministic turing machine	$S \to aSc \mid aBc$, $CB \to HB$, $HB \to Hc$
Type - 0	Unrestricted	Turing machine	$S \to Abc$, $Ab \to c$, $cc \to Sc \mid \varepsilon$