Department of Computer Science and Engineering

...ity Chittagong

B. Sc. in CSE Midterm Examination, Autumn 2019

Course Code: CSE 2425 Course Title: Theory of Computing

Course Code: CSE 3609 Course Title: Theory of Computing

Total marks: ***30*** *Time:* ***1*** *hours* ***30*** *minutes*

[Answer any ***three*** questions;

Parts of the same question should be answered sequentially;

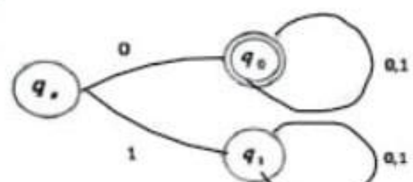
Figures in the right hand margin indicate full marks.]

1.

a) What are the topics of discussion of Theory of Computation? 3

b) Give Formal Definition of Finite Automata. 2

c) Give i) Transition Functions ii) Alphabet and iii) Accepted words (i.e. the language) for the following example: 3



d) Construct a DFA that accepts set of all strings of even length where alphabet is {0,1}. 2

2.

a) Construct a DFA which accepts set of all strings, where every string contains the substring '01'. Assume the input alphabet is {a,b}. 2.5

b) Construct a DFA which accepts set of all strings, where the strings do not contain the substring '111'. Assume the input alphabet is {a,b}. 2.5

c) Construct a Deterministic Finite Automata which accepts set of all strings over Σ = {a,b} where each strings ends with an '**ab**'. 2.5

d) Construct a Deterministic Finite Automata, Σ = {a,b} and **L(M) = {ω| ω starts and ends with different symbol }**. 2.5

3.

a) Give the formal definition of NFA with proper example. 2

b) Construct a NFA with state transition table, where second symbol from Right hand side is an 'a' and also convert this NFA to DFA. Assume the input alphabet is {a,b}. 3

c) Prove that: "**There is an equivalent DFA for every NFA**." 5

4.

a) What is Regular Expression and also show the six cases of regular expression. 4

b) Design Regular Expression for the following language over {a,b} 4

i) Language accepting strings of length exactly 2

ii) Language accepting strings of length at least 2

c) Define the following sets as Regular Expressions: 2

i) {0, 00, 000 ... ...}

ii) {aba, ababa, abababa ... ...}

1a

Theory of Computation is very important as it helps in writing efficient algorithms that operate on computer devices, research and development of programming languages and in compiler design and construction that is efficient,.This branch is divided into 4 parts, namely:

Automata Theory.

Formal Language

Computability theory

Complexity theory.

Automata Theory is a theoretical branch of Computer Science and mathematics and deals with the study of complex computational problems and abstract machines.Finite Automata, also known as the Finite State Machine, is a simple machine that is able to recognize patterns. It is an abstract machine with five components or tuples.Formal Language Theory is a branch of Computer Science and Mathematics dealing with representing languages as a collection of operations on an alphabet.Computability theory, also known as Recursion Theory, is a branch of Mathematics and Computer Science that is primarily concerned with the extent to which an issue may be solved by a computer

: Computational Complexity Theory is that branch of Theory of Computation that classifies computational problems according to their resource usage. These computational problems are solved by different algorithm,

1b

Formal Definition of a DFA

A DFA can be represented by a 5-tuple (Q, ∑, δ, q_0, F) where −

- **Q** is a finite set of states.
- ∑ is a finite set of symbols called the alphabet.
- **δ** is the transition function where δ: Q × ∑ → Q
- $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Graphical Representation of a DFA

A DFA is represented by digraphs called **state diagram**.

- The vertices represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

Example

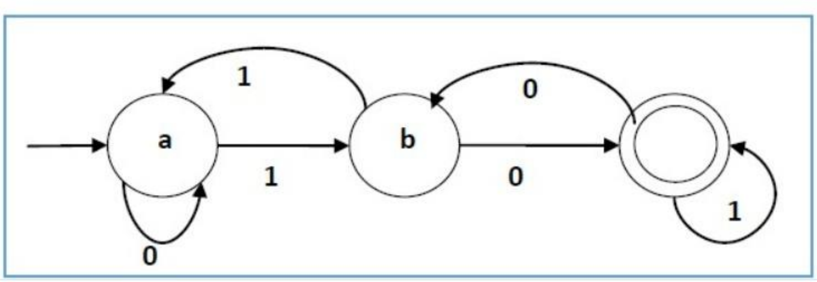
Let a deterministic finite automaton be →

- Q = {a, b, c},
- ∑ = {0, 1},
- q_0 = {a},
- F = {c}, and

Transition function δ as shown by the following table −

Present State	Next State for Input 0	Next State for Input 1
a	a	b
b	c	a
c	b	c

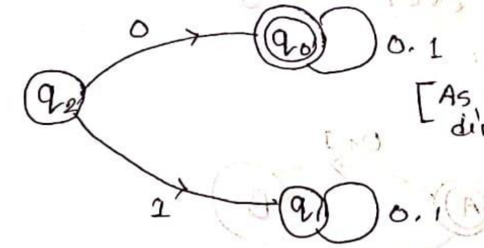
Its graphical representation would be as follows −



1c

(1e)

Give (i)



[As there's have no direction, assume direction]

(i) Transition function: $\delta \rightarrow Q \times \Sigma$

State	Input	Next state
q_0	0	q_0
q_0	1	q_0
q_1	0	q_1
q_1	1	q_1
q_2	0	q_0
q_2	1	q_1

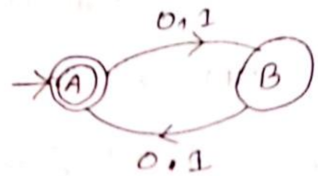
(ii) Alphabet $= \{0, 1\}$

(iii) Accepted words $\rightarrow \{q_0\}$

1d

1(d)

Construct a DFA that accepts set of all strings of even length where alphabet is $\{0,1\}$.



in this DFA,

→ set of states : $Q = \{A, B\}$

→ Alphabet = $\{0, 1\}$

→ transition function $\delta \rightarrow Q \times \Sigma = Q$

State	Input	Next state
A	0	B
A	1	B
B	0	A
A	1	A

→ Initial state and final state $= \{A\}$.

2a

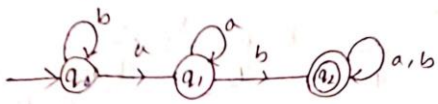
$\boxed{2a}$

Construct a DFA which accepts set of all strings where every string contains the substring '01',

Assume the input Alphabet is $\{a, b\}$

Let, $a = 0$
$b = 1$

substring '01' = 'ab'



→ $Q = \{q_0, q_1, q_2\}$

→ $\Sigma = \{a, b\}$

→ transition function $\delta \rightarrow Q \times \Sigma \rightarrow Q$

~~state~~	~~Input~~	~~Next state~~
~~q_0~~	~~a~~	~~q_1~~
~~q_0~~	~~b~~	~~q_0~~

	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_2	q_2

→ Initial state → $\{q_0\}$

→ final / Accept state → $\{q_2\}$,

2b

(2b)

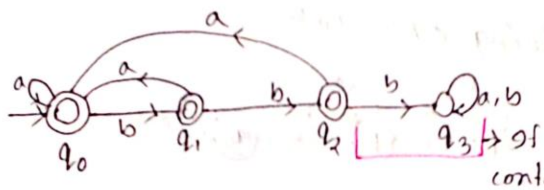
construct A dfA where string do not contain '111',

As alphabet = {a, b}

Let, a = '0'

b = '1')

'111' = 'bbb' [consecutive 3 b]



→ If any ~~input~~ string contain ~~com~~ 'bbb' that IS not accepted in this state.

OR

→ Q → {q_0, q_1, q_2, q_3}

→ Σ → {a, b}

→ transition function δ →

	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_3

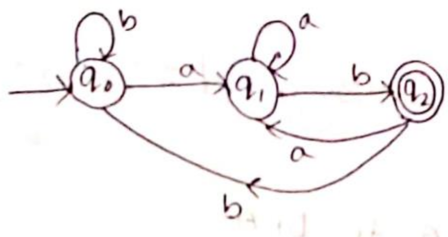
→ Initial state → {q_0}

→ Final state → {q_0, q_1, q_2}

2c

2 (a)

Construct DFA which ends with an 'ab'



$Q \rightarrow \{q_0, q_1, q_2\}$

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

Initial state $\rightarrow \{q_0\}$

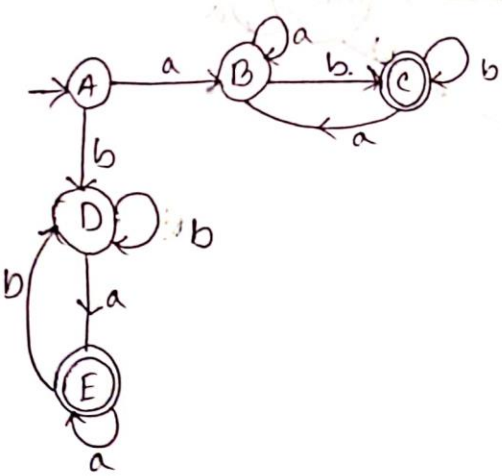
final state $\rightarrow \{q_2\}$

transition function $\delta \rightarrow$

	Next state	
	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_0

2d

$L(M) = \{w \mid w$ starts and ends with different symbol$\}$



3a

In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called **Non-deterministic Automaton**. As it has finite number of states, the machine is called **Non-deterministic Finite Machine** or **Non-deterministic Finite Automaton**.

Formal Definition of an NDFA

An NDFA can be represented by a 5-tuple (Q, $\sum$, δ, q_0, F) where −

- **Q** is a finite set of states.
- $\sum$ is a finite set of symbols called the alphabets.
- **δ** is the transition function where $\delta: Q \times \sum \rightarrow 2^Q$

  (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Graphical Representation of an NDFA: (same as DFA)

An NDFA is represented by digraphs called state diagram.

- The vertices represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

Example

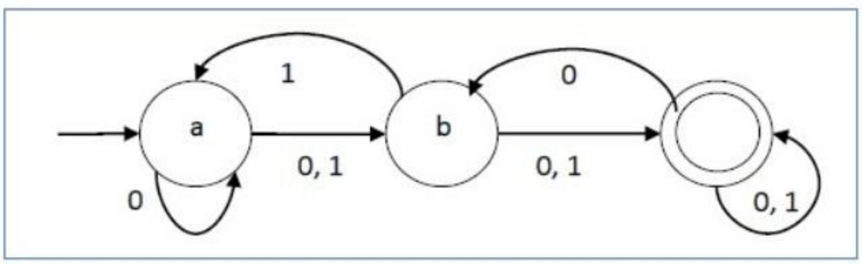
Let a non-deterministic finite automaton be →

- Q = {a, b, c}
- ∑ = {0, 1}
- q_0 = {a}
- F = {c}

The transition function δ as shown below −

Present State	Next State for Input 0	Next State for Input 1
a	a, b	b
b	c	a, c
c	b, c	c

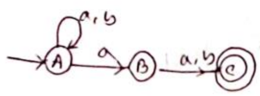
Its graphical representation would be as follows −



3b

L = {w|w from RHS 2nd symbol is 'a'}

where Alphabet is Σ = {a, b}



[Σ*, a, Σ*]

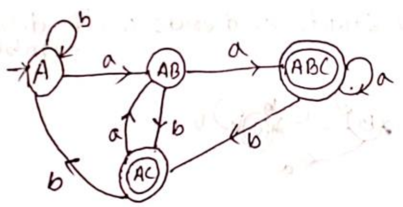
Conversion NFA to DFA

for NFA: state transition Table

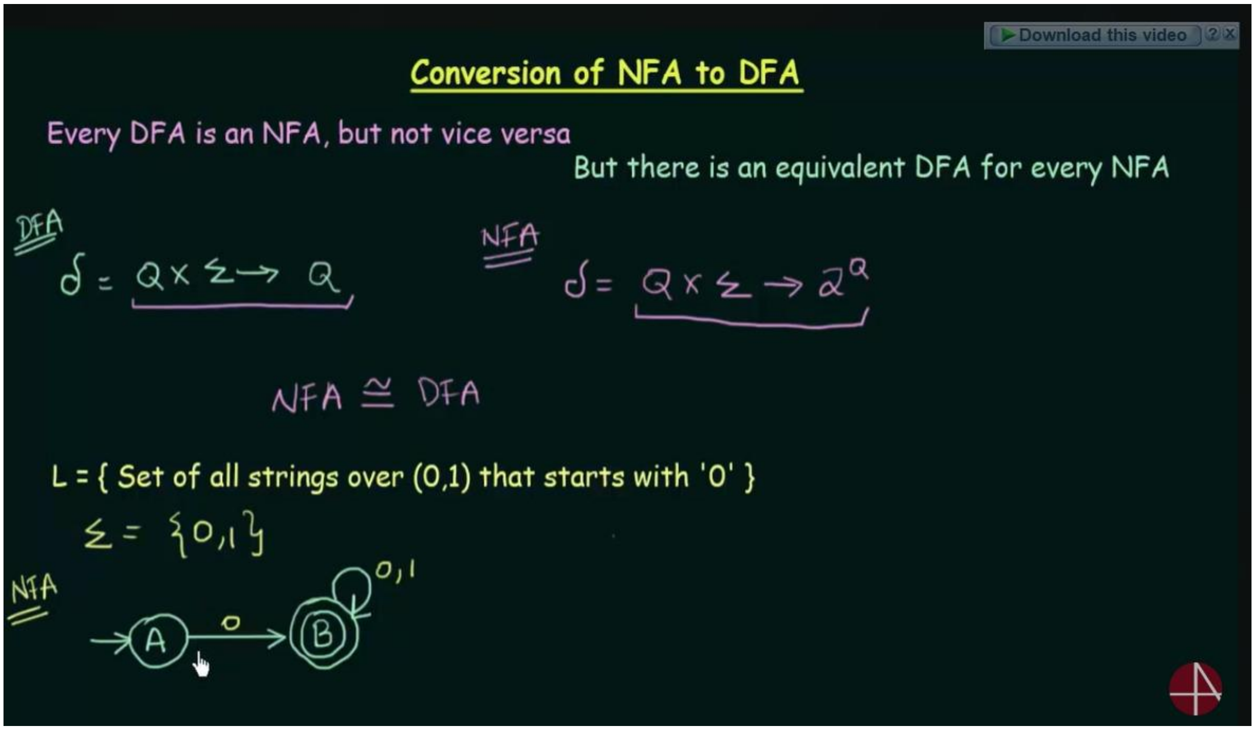
	a	b
→ A	{A, B}	{A}
B	{C}	{C}
* C	{ }	{ }

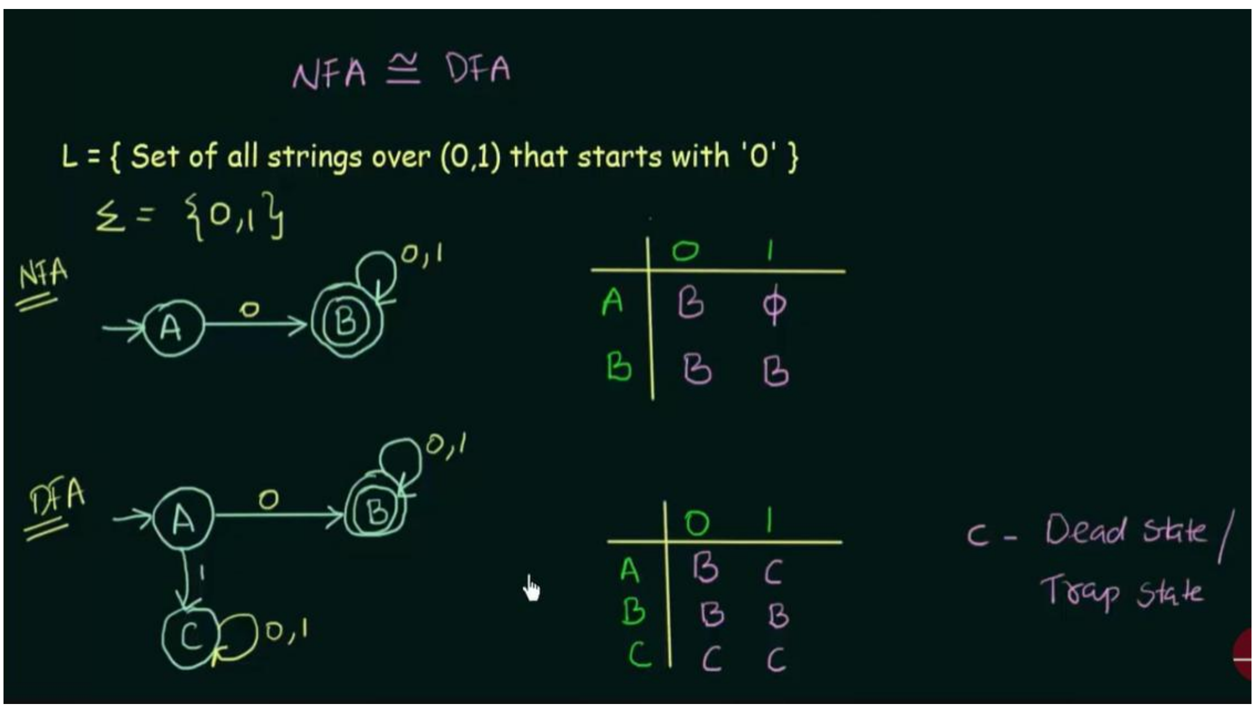
DFA state T. Table

	a	b
→ A	[AB]	[A]
[AB]	[ABC]	[AC]
final state * [AC]	[AB]	[A]
final state * [ABC]	[ABC]	[AC]



3c: There is an equivalent DFA for every NFA.





4a

A **Regular Expression** can be recursively defined as follows −

- **ε** is a Regular Expression indicates the language containing an empty string. **(L (ε) = {ε})**
- **φ** is a Regular Expression denoting an empty language. **(L (φ) = { })**
- **x** is a Regular Expression where **L = {x}**
- If **X** is a Regular Expression denoting the language **L(X)** and **Y** is a Regular Expression denoting the language **L(Y)**, then
  - **X + Y** is a Regular Expression corresponding to the language **L(X) ∪ L(Y)** where **L(X+Y) = L(X) ∪ L(Y)**.
  - **X . Y** is a Regular Expression corresponding to the language **L(X) . L(Y)** where **L(X.Y) = L(X) . L(Y)**
  - **R*** is a Regular Expression corresponding to the language **L(R*)**where **L(R*) = (L(R))***
- If we apply any of the rules several times from 1 to 5, they are Regular Expressions.

(4C)

i) $\{0, 00, 000 - - -\}$

$= 0^+$

(i) $\{aba, ababa, abababa\}$

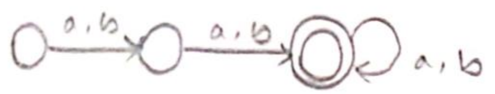
$= \cancel{a(ba)^*}\ a(ba)^+$

4B

(i) Language accepting string of Length exactly 2.

$(a+b)^+(a+b)^+$

(ii) Language accepting string of length at least 2.



$RE \Rightarrow (a+b)^+(a+b)^+a^*\, b^*$

If there is any mistake, do correction by yourself.