Problem-1

1. Explain the fundamental difference b/w Deterministic Finite Automata (DFA) and Non Deterministic Finite Automata (NFA) How do these differences impact the expressive power and Complexity of these two types of automata.

A DFA is deterministic, which means that for each state and input Symbol, there is exactly one possible next state. An NFA is non-deterministic, which means that for each state and input Symbol, there may be zero or more Possible next states.

Expressive Power: DFAs are strictly less expressive than NFA's This means, there are some languages that can be recognized by an NFA but not by a DFA.

Complexity: NFA's are generally more complex than DFA's. This is because NFA's have to explore all possible paths through the state graph, while DFA's can only explore one Path at a time. As a result, NFA's can take longer to run than DFA's

In general, DFA's are preferred over NFA's when possible because they are more efficient. However, NFA's can be useful when the language to be ~~reg~~ recognized is not strictly regular, or when it is easier to construct an NFA than a DFA.

Consider the language $L = \{ w/w$ is a binary string containing an even number of 0's and an odd number of 1's $\}$. Design a DFA to recognize this language.

A Consider the given Language 'L'

Step1:

$L = \{ w/w$ Contains binary string an even number of 0's and odd no. of 1's $\}$

$$L = \{ 1, 001, 100, 010, 00100 \cdots \}$$

Step2: To Construct DFA for $M = (Q, \Sigma, q_0, F, \delta)$

where, Q - finite set of states

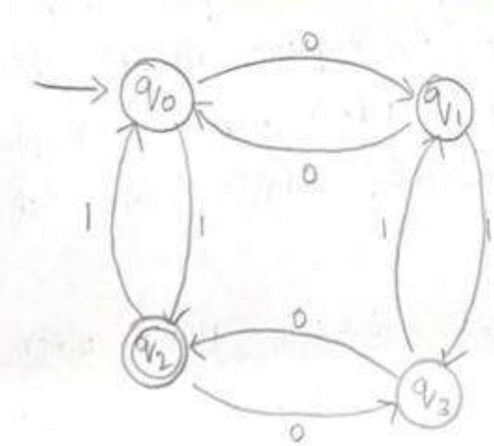
Σ - finite input alphabets

q_0 - initial state

F - final state

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

Step3: Construct transition diagram for step 1



Step 4: Construct transition table

δ	0	1
q_0	q_1	q_2
q_1	q_0	q_3
q_2	q_3	q_0
q_3	q_2	q_1

Step 5:

finalization of 5 tuples.

Q : set of finite state $\{q_0, q_1, q_2, q_3\}$

q_0 : $q_0 \in Q$ q_0 is initial

Σ : input symbol $\{0, 1\}$

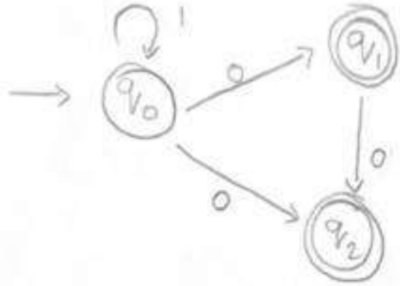
F : $F \subseteq Q$, $F = \{q_2\}$

δ : $Q \times \Sigma \rightarrow Q$

$\delta(q_0, 0) \rightarrow q_1$; $\delta(q_1, 0) \rightarrow q_0$; $\delta(q_2, 0) \rightarrow q_3$; $\delta(q_3, 0) \rightarrow q_2$

$\delta(q_0, 1) \rightarrow q_2$; $\delta(q_1, 1) \rightarrow q_3$; $\delta(q_2, 1) \rightarrow q_0$; $\delta(q_3, 1) \rightarrow q_1$

2 Given the following NFA M:



Perform the NFA to DFA conversion for the given NFA M. show each step of the subset construction process, including the states and transition of the resulting DFA.

NFA to DFA conversion.

Step1 : 5 tuples :

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

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

$q_0 : q_0 \in Q$

$F : q_2 \subseteq Q$

δ : transition function : $Q \times \Sigma \to Q$

Step2 : Table:

δ	0	1
$\rightarrow q_0$	q_1, q_2	q_0
q_1^*	q_2	ϕ
q_2^*	ϕ	q_2

Step 3 :

→ Mapping function : $q_0 \times 0 \rightarrow q_1, q_2$

$q_0 \times 1 \rightarrow q_0$

$q_1 \times 0 \rightarrow q_2$

$q_1 \times 1 \rightarrow \phi$ initial state : q_0

$q_2 \times 0 \rightarrow \phi$ final state : q_2, q_1

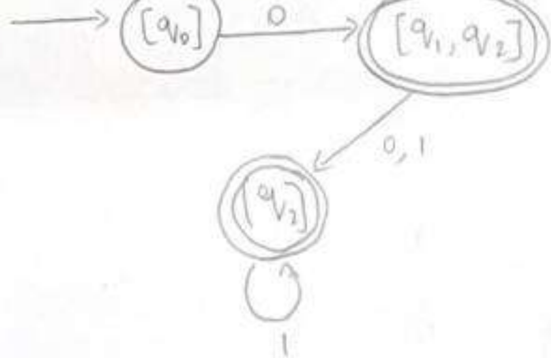
$q_2 \times 1 \rightarrow q_2$

The given one is a NFA containing more than one state to the result function. convert into DFA by constructing DFA construction table.

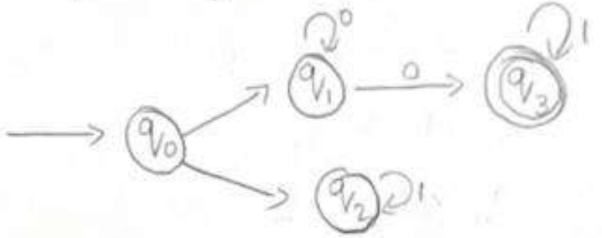
Step 4 :

δ'	0	1
$\rightarrow [q_0]$	$[q_1, q_2]$	$[q_0]$
$[q_1, q_2]^*$	$[q_2, \phi]$	$[\phi, q_2]$
$[q_2]^*$	ϕ	$[q_2]$

Steps : To construction of transition Diagram



3. Given the following NFA N :



~~Ref~~ Perform the NFA to DFA conversion for the given NFA N. consider the epsilon transitions during the Conversion. show each step of the subject construct process, including the states and transition of the resulting DFA

Now to find the λ-closure of each state from given diagram

ϵ-closure (q_0) = $\{q_0, q_1, q_2\}$

ϵ-closure (q_1) = $\{q_1\}$

ϵ-closure (q_2) = $\{q_2\}$

ϵ-closure (q_3) = $\{q_3\}$

transition table

δ	ϵ	0	1
$\rightarrow q_0$	q_1, q_2	$\emptyset$	$\emptyset$
q_1	$\emptyset$	q_1, q_3	$\emptyset$
q_2	$\emptyset$	$\emptyset$	q_2
q_3^*	$\emptyset$	$\emptyset$	q_3

$$\delta'\{q_0, q_1, q_2\} = \epsilon\text{-closure}\{(\delta'(q_0, q_1, q_2))\}$$

$$= \epsilon\text{-closure}((q_0, 0) \cup (q_1, 0) \cup (q_2, 0))$$

$$= (\emptyset \cup q_1, q_3 \cup \emptyset)$$

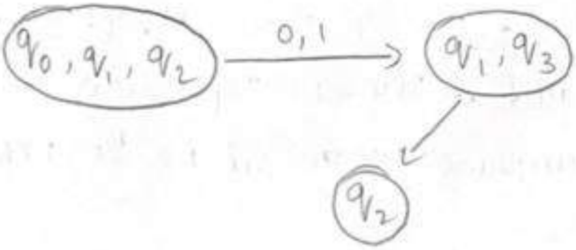
$$= q_1, q_3$$

$$\delta'\{q_0, q_1, q_2\} = \epsilon\text{-closure}\{(\delta'(q_0, q_1, q_2)\}, 1))\}$$

$$= \epsilon\text{-closure}((q_0, 1) \cup (q_1, 1) \cup (q_2, 1))$$

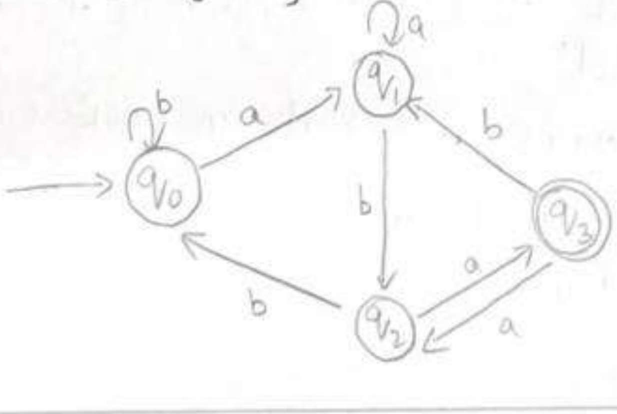
$$= (\emptyset \cup \emptyset \cup q_2)$$

$$= q_2$$



4. Explain the process of converting a Deterministic finite function into a Regular Expression (RE). How does Arden's theorem play a crucial role in this conversion? provide a step by step explanation of the conversion process by Arden theorem.

Consider a DFA M that recognizes the language l over the alphabet $\Sigma = \{a, b\}$ as shown below:



Use Arden's theorem to find a regular expression that represents the Language L recognized by DFA M.

Converting a Determinantic Finite Automata (DFA) into a Regular Expression (RE) is a techique used to simplify and represent the language recognized by the DFA in a more concise and expression from Arden's theorem is a crucial tool in this conversion process, as it provides a systematic way to derive the regular expression based on the DFA's transition and states. The process can be broken down into the following steps:

1. Setup:

Let's assume, we have a DFA with states Q, alphabet symbol Σ, transition function δ, start state q_0, set of finite states F. the goal is to find a regular expression that represents that the language recognized by the DFA

2. Create Equations:

For each state q in Q, we will create an equation that represents the language accepted the DFA starting from that state, The idea to express the language as a combination of individual symbols and concentenations.

- for final states q_f in F: create an equation for each final state q_f such that it accepts only its own symbol i.e q_f = "symbol".
- for non final states q: create an equation for each non final state q as a sum of concatenations of its outgoing transitions

3. Apply Arden's theorem: Arden's theorem states that for any regular expressions A and B and a variable x, the equation $x = Ax|B$ can be solved for x to yeald $x = A^*B$, where * represents the Kleene star operation.

4. Solve Equations: Start solving the equations you've created using Arden's theorem. Begin with equations of final states since they are already in correct form (i.e symbol) for non final states, apply Arden's theorem to solve their corresponding regular expressions

5. Substitute Equations: As you solve equations for different states, substitute the expressions you've found into other equations to replace references to those states. continue this substitution process iteratively until all references are replaced by regular expressions.

6. Final Expression: Once you've solved all equations and performed all substitutions, the equation representing the start state q0 will be your final regular expression that represents the language recognized by the DFA.

5. Explain the pumping Lemma for Regular Language and its significance in proving that certain language are not regular. Describe the key components of the pumping Lemma statement and how it can be used to demonstrate the non regularity of a language.

Consider the Language $L = \{0^n 1^n \mid n \geq 0\}$, which consist of strings of the form "0^n1^n" where the number of 0's and 1's are the same. use the pumping Lemma for Regular Language to prove that the Language L is not regular

Consider The pumping Lemma for Regular languages is a powerful tool used to prove the non negative regularity of a certain language it states that for every regular language L there exists a pumping length p (a positive integer) such that any string s in L with length of least p can be divided into three parts $s = xyz$, satisfying the following conditions:

1. for any non negative integer i, the string $xy^i z$ is also in L
2. the length of the string y must be greater than 0.
3. The combined length of xy must be less than or equal to p

Languag L: $\{0^n 1^n \mid n \geq 0\}$

Assumption: Assume that the language L is regular
let p be the pumping length for L as per the pumping Lemma consider the string $s = 0^p 1^p$ this string is in L and has a length greater than or equal to p

write $s = xyz$

x : 0s
y : 0's
z : 1's

let pump y the resulting string would by $xy^2 z = xyyz$ since y consists only 0's, pumping y will result in more 0's than is in the resulting string. However this new string is no longer in L. as the number of 0's is now greater than the number of 1's.

This contradicts the definition of language L, which requires an equal number of 0's and 1's. Thus, our assumption that L is regular must be false

As we have reached a contradiction, we conclude that the language $L = \{0^n 1^n \mid n \geq 0\}$ is not regular.

6. what is a Regular Grammar, and how does it contribute to the generation of Regular languages? Discuss the relation b/w regular granmer and finite automata, highlighting how regular grammer capture the essence of regular languages.

Consider the regular language $L = \{w / w$ is a binary string containing an even number of 0's$\}$ Design a regular Grammar that generates the Language L.

A Regular Grammar is a type of formal grammar used to generate Regular grammar Languages. it is characterized by having production rules of the form $A \rightarrow xB$ or $A \rightarrow \lambda$, where A and B are non terminal symbols and x is a string consisting of terminal symbols. Regular grammar are closely related to finite automata and capture the essence of regular languages by defining a set of rules that can generate strings in a straight forward and limited way.

Regular Grammar vs finite Automata.

- Regular Grammar:
  - → Consists of a set of non terminal symbol, a set of terminal symbols, a start symbol and a set of production rules

→ Production rules are the form of $A \to xB$ or $A \to x$, where A and B are non-terminals and x is a string of terminals or empty string.

→ Regular grammars are capable of generating strings in a left to right maner with a one to one correspondence b/w productions and states.

Finite Automata

→ Consists of a set of states, an alphabet, a transition function a start state and a set of accept states.

→ the transition function defines how the automation transition from one state to another based on input symbols.

→ Finite automata recognize languages by transition through states while reading input symbol.

$L = \{ w/w$ is a binary string containing an even number of 0's$\}$ start symbol s

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

Production rules : $S \to 0A \mid 1S \mid \varepsilon$

$A \to 1A \mid 0's$

Explanation of the production rules.

- $S \to 0A$: start with 0 followed by even number of 0's
- $S \to 1's$: start with 1 continue generating more symbols
- $S \to \epsilon$: empty string.
- $A \to 1A$: start with 1 and followed by an even numbers of 0's
- $A \to 0's$: start with 0's and continue generating more symbols.

The grammar generates strings in a way that ensures there's an even number of 0's in the resulting string

7 Consider the Regular Expression $abba(aba)^* bb$. Design the Right linear grammar for the Regular expression. use thompson's construction technique to construct a NFA for this Regular Expression.

The right linear grammar for the regular expression $abba(aba)^*$ bb is:

$$S \rightarrow abba\ A$$
$$A \rightarrow aba\ A \mid bb$$

The start symbol is S and the production rules are

$$S \rightarrow abba\ A$$
$$A - aba\ A + bb$$
$$A \rightarrow bb$$

$$S \rightarrow abba\ A$$
$$A \rightarrow aba\ A$$

The following is a diagram of the NFA for the regular expression $(aba)^* bb$