**Chapter 5: REGULAR EXPRESSIONS**

**Topic – 1: Basic Concept**

**Introduction**

* **Regular expression:** Any language that is accepted by **finite automata**.
* These must be accepted by some given expression.

**Application**

* Regular expressions define a pattern found in string.
* This is exactly what’s used in **compiler designing**.

**Certain Annotations**

* **a\*** means **'a' might** occur or **even not occur**.
* **a+** means **'a'** occurs **once** or **more than once**.

**Topic – 2: Operations On Regular Languages**

**Union**

* **Union** of two regular languages is also a regular language.

**L U M = {s | s is in L or s is in M}**

**Intersection**

* **Intersection** of two regular languages is also a regular language.

**L ⋂ M = {st | s is in L and t is in M}**

**Keen Closure**

* **Keen closure:** The **L\*** annotation.

**L\* = Zero or more occurrence of language L**

**Topic – 3: Examples**

**Example**

**Ques: Regular expression accepting all combination of 'a' from Σ = {a}.**

**Ans:**

**R = a\***

**[Even null character is considered a’s appearance]**

**L = {φ, a, aa, aaa, …}**

**Example - II**

**Ques: Regular expression accepting all combination of 'a' from Σ = {a}, except null character.**

**Ans:**

**R = a+**

**L = {a, aa, aaa, …}**

**Example – III**

**Ques: Regular expression accepting any number of 'a' & 'b' from Σ = {a,b}.**

**Ans:**

**R = (a+b)\***

**Example – IV**

**Ques: Regular expression starting with 'a' & no consecutive appearance of 'b' from Σ = {a,b}.**

**Ans:**

**R = (a + ab)\***

**Example – V**

**Ques: Regular expression accepting even length of string from Σ = {0}.**

**Ans:**

**R = (00)\***

**Example – VI**

**Ques: Guess the language for the following regular expression:**

**R = {b\*(aaa)\*b\*}\***

**Ans: A string which is either null or is a string which may start with b, may end with b & a only appears in triplets.**

**Topic – 4: RE To FA Conversion**

**Method**

* We use ***subset method*** to make this conversion.

**Steps Involved**

* **Step 1:** Design a **Ԑ-NFA** for the given **regular expression**.
* **Step 2:** Convert this **Ԑ-NFA** to **NFA**.
* **Step 3:** Convert that **NFA** to **DFA**.

**Rules Applied (RE To FA)**

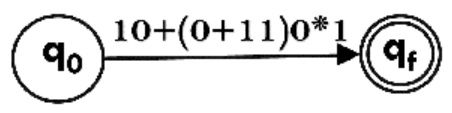
* **Rule 1:** **+** symbol in expression means **two separate lines** of transitions toward the **same** state. One having **left** side of **+** & another having **right**.
* **Rule 2:** Nothing between **'n'** characters of a string means **(n-1)** states will come **between** them.
* **Rule 3:** **\*** between two characters mean that the character preceding **\*** **self-transitions** to the **previous state**.
* **Rule 4:** If a part of expression is under **()\*** then it is **self-transitioned** to next state & previous state is **deleted**.
* **Rule 5:** If a **closed figure** is formed somewhere, then the final diagram must have that **loop** there too. Though some modification can be made, it **can’t** be separated.
* **Rule 6:** We can even separate the **closed figure final state** as **two final states** if we **don’t** want a closed figure in our automaton.

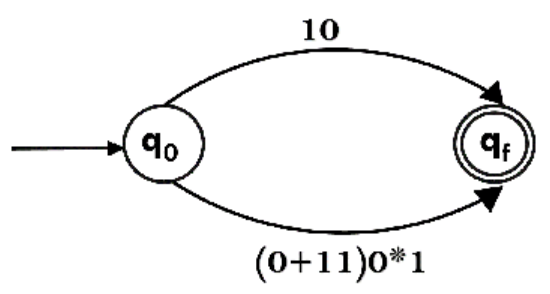
**Example**

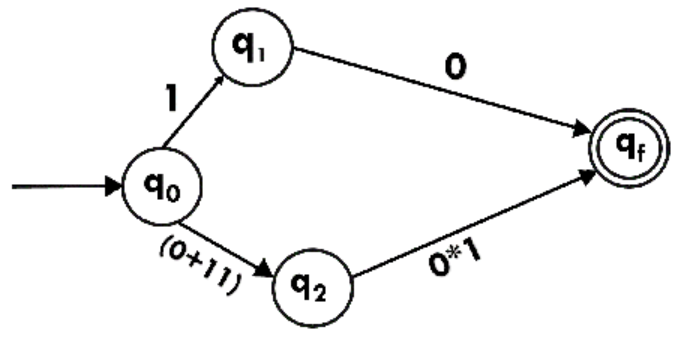
**Ques:**

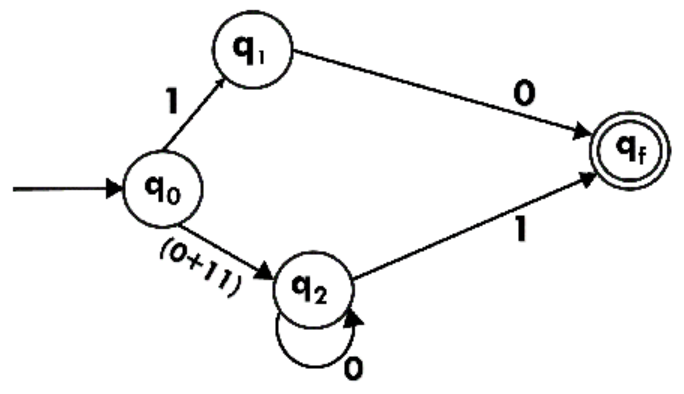
**RE = 10+(0+11)0\*1**

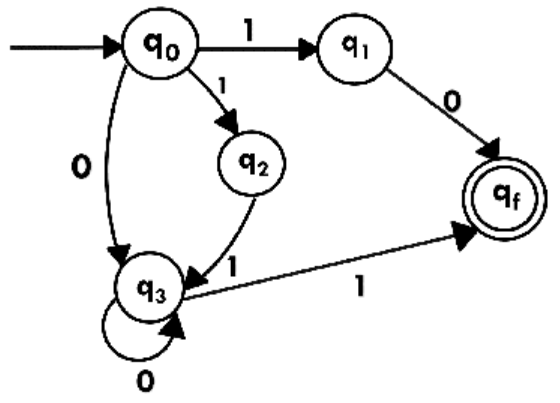
**Ans:**

****

****

****

****

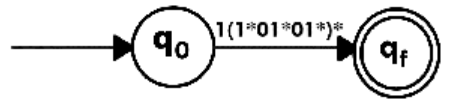
****

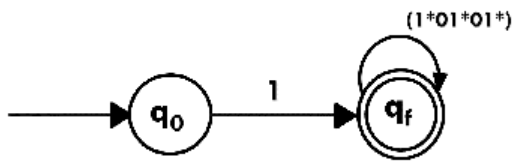
**Example - II**

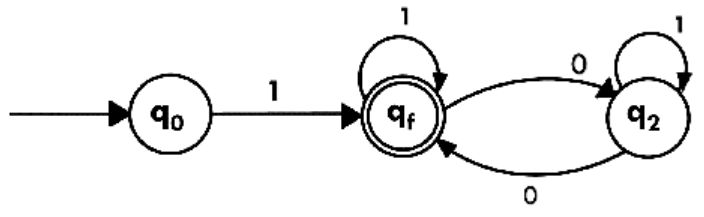
**Ques:**

**RE = 1(1\*01\*01\*)\***

**Ans:**

****

****

****

**Topic – 5: Arden’s Theorem**

**Introduction**

* **Arden’s theorem** is used in checking if two **RE** are **equal**.
* It is also used to convert a **DFA to RE**.

**Theorem**

* Let **initial state** be **q1** where **'n'** states exist in an automaton.
* And let **final state** be **qj** where **j ≤ n**.
* **αij** represents transition from state **qi** to final state **qj**.

**If final state qj is the first state:**

**qj = αij \* qi**

**Else:**

**qj = αij \* qi + Ԑ**

**Steps Involved**

* **Step 1:** We write the **final states** in form of **equations** we just discussed.
* **Step 2:** Then we **apply laws** on them for **conversion**.
* **Step 3:** After it is done on **each** final state equation, we **add** them.
* **Step 4:** Finally, we **simplify** it.

**Laws Followed**

**Law 1:**

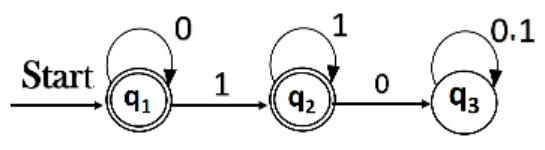
**R = Q + RP = QP\***

**Law 2:**

**Ԑ.R\* = R\***

**Example**

**Ques:**

****

**Ans:**

**Final state equations:**

**q1 = q1 \* 0 + Ԑ**

**q2 = q1 \* 1 + q2 \* 1**

**Now applying laws:**

**q1 = q1 \* 0 + Ԑ**

**= Ԑ + q1 \* 0**

**= Ԑ.(0)\* [Law 1]**

**= 0\* [Law 2]**

**q2 = q1 \* 1 + q2 \* 1**

**= 0\*1 + q2 \* 1 [Substitution]**

**= 0\*11\* [Law 1]**

**= 0\*1+**

**R = q1 + q2+**

**= 0\* + 0\*1+**

**Topic – 6: Moore’s Machine**

**Introduction**

* In **Moore’s machine**, **next state** is decided by **current state** & **current input symbol**.
* The **current state** is written in **each state**.
* **Arrows** denote the **input** & **current state** denotes **output**.

**Moore’s Tuple**

* **Moore’s machine** is defined by the following tuple.

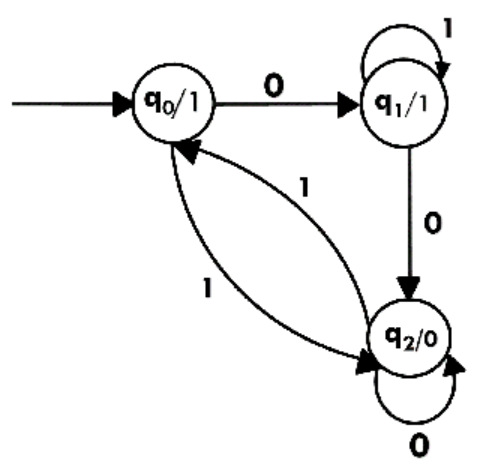
**{Q, q0, Σ, O, δ, λ}**

**O = Output alphabet**

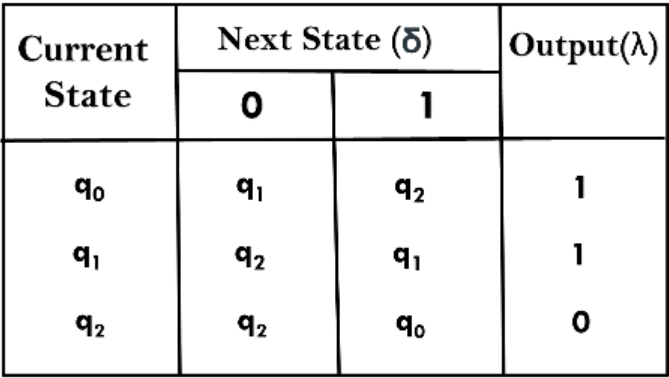
**λ = Output function (Q 🡪 O)**

**Example**

**Ques: Design a table for the given Moore’s machine.**

****

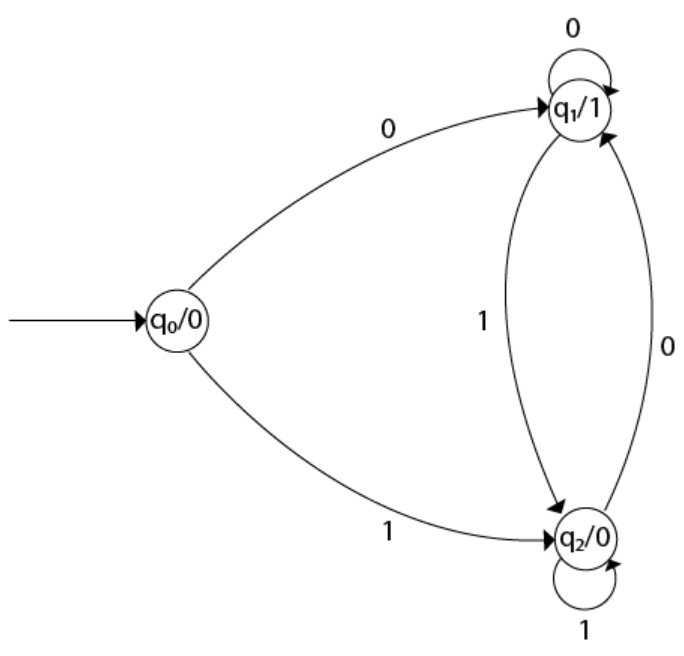
**Ans:**

****

**Example – II**

**Ques: Design a Moore’s machine to generate 1’s complement of a given binary number.**

**Ans:**

****

**At starting, everything remains on the initial state with current value 0. Then the transitions occur as per the input string.**

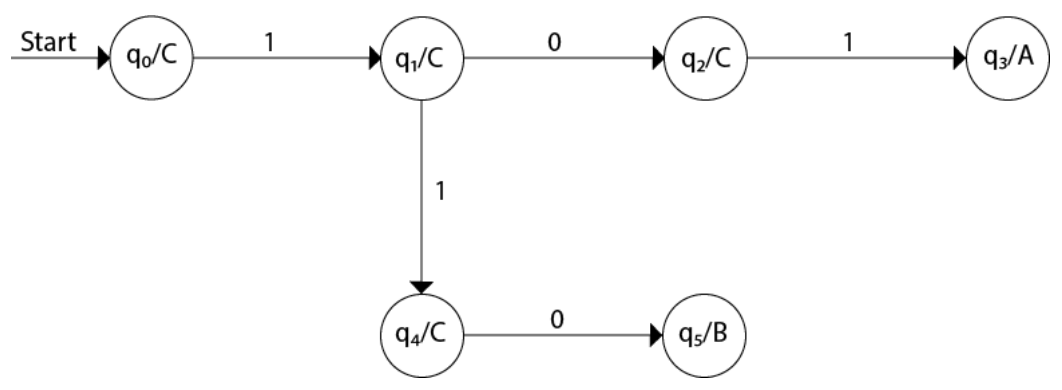
**Example – III**

**Ques: Design a Moore’s machine such that if substring 101 is found, then output is A. If 110 is found, then output is B. Else, the output is C.**

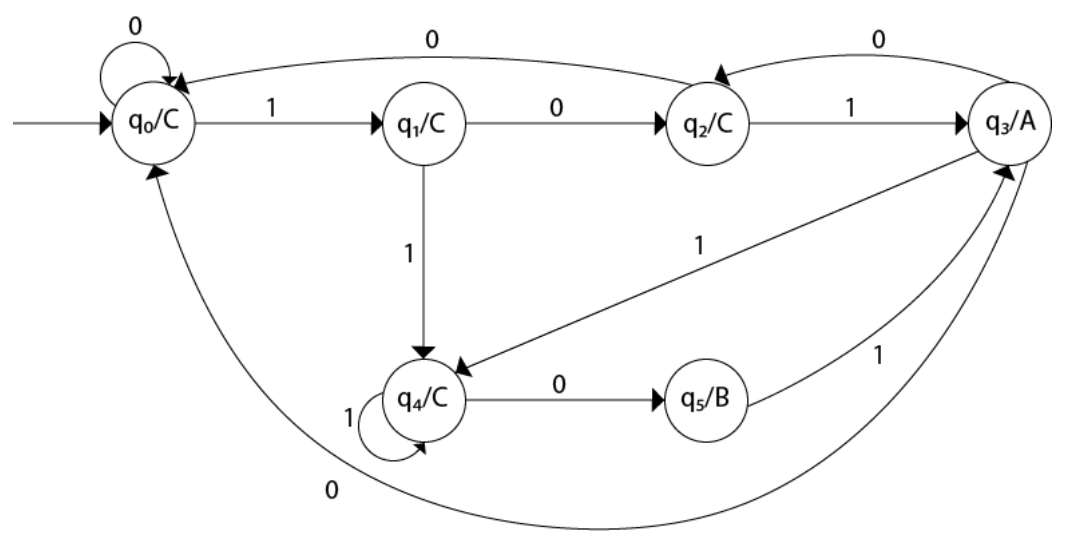
**Ans:**

**There is no preference given for output. Means the machine doesn’t stop when it receives 101 or 110. It will continue working & output as per the latest substring found. Output can change anytime from any current state.**

**First, we draw this much for ease:**

****

**Then we add remaining output lines:**

****

**Topic – 7: Mealy Machine**

**Introduction**

* The **output symbol** depends on **present input symbol** & **present state**.
* **Output** is represented on **arrows** where **input symbols** are separated by **/**.
* This is basically **same** as **Moore’s machine** & outputs **aren’t** written inside the state but on the **preceding arrows**, before reaching the state.

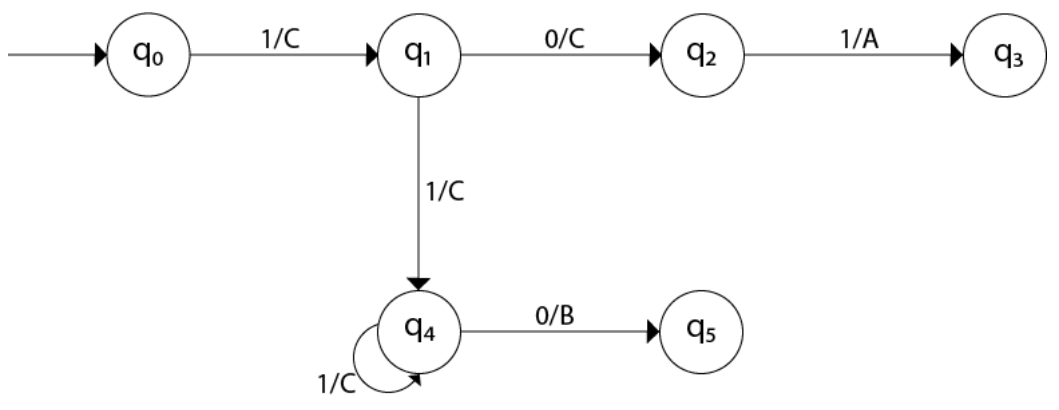
**Mealy Tuple**

* **Mealy machine** is defined by the following tuple.

**{Q, q0, Σ, O, δ, λ’}**

**λ’ = Output function (Q × Σ 🡪 O)**

**Example**

****

**In q0 to q1 transition, the input at state q1 is C. So, we write 1/C on its preceding arrow where 1 is input & C is its corresponding output.**