UNIT-II

Regular Expressions, Grammar and Languages

As discussed in Chomsky Hierarchy, Regular Languages are the most restricted types of languages and are accepted by finite automata.

Regular Expressions

Regular Expressions are used to denote regular languages. An expression is regular if:

- ϕ is a regular expression for regular language ϕ.
- ε is a regular expression for regular language {ε}.
- If a ∈ Σ (Σ represents the input alphabet), a is regular expression with language {a}.
- If a and b are regular expression, a + b is also a regular expression with language {a,b}.
- If a and b are regular expression, ab (concatenation of a and b) is also regular.
- If a is regular expression, a* (0 or more times a) is also regular.

Identities for regular expression –

There are many identities for the regular expression. Let p, q and r are regular expressions.

- ∅ + r = r
- ∅.r= r.∅ = ∅
- ∈.r = r.∈ =r
- ∈* = ∈ and ∅* = ∈
- r + r = r
- r*.r* = r*
- r.r* = r*.r = r^+.
- (r*)* = r*
- ∈ +r.r* = r* = ∈ + r.r*
- (p.q)*.p = p.(q.p)*
- (p + q)* = (p*.q*)* = (p* + q*)*
- (p+ q).r= p.r+ q.r and r.(p+q) = r.p + r.q

Regular Expressions

Regular Expressions are used to denote regular languages. An expression is regular if:

- ϕ is a regular expression for regular language ϕ.
- ε is a regular expression for regular language {ε}.
- If a ∈ Σ (Σ represents the input alphabet), a is regular expression with language {a}.
- If a and b are regular expression, a + b is also a regular expression with language {a,b}.
- If a and b are regular expression, ab (concatenation of a and b) is also regular.
- If a is regular expression, a* (0 or more times a) is also regular.

Closure Properties of Regular Languages

Union : If L1 and If L2 are two regular languages, their union L1 ∪ L2 will also be regular. For example, L1 = $\{a^n \mid n \geq 0\}$ and L2 = $\{b^n \mid n \geq 0\}$

L3 = L1 ∪ L2 = $\{a^n \cup b^n \mid n \geq 0\}$ is also regular.

Intersection : If L1 and If L2 are two regular languages, their intersection L1 ∩ L2 will also be regular. For example,

L1= $\{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ and L2= $\{a^m b^n \cup b^n a^m \mid n \geq 0 \text{ and } m \geq 0\}$

L3 = L1 ∩ L2 = $\{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ is also regular.

Concatenation : If L1 and If L2 are two regular languages, their concatenation L1.L2 will also be regular. For example,

L1 = $\{a^n \mid n \geq 0\}$ and L2 = $\{b^n \mid n \geq 0\}$

L3 = L1.L2 = $\{a^m . b^n \mid m \geq 0 \text{ and } n \geq 0\}$ is also regular.

Kleene Closure : If L1 is a regular language, its Kleene closure L1* will also be regular. For example,

L1 = (a ∪ b)

L1* = (a ∪ b)*

Complement : If L(G) is regular language, its complement L'(G) will also be regular. Complement of a language can be found by subtracting strings which are in L(G) from all possible strings. For example,

L(G) = $\{a^n \mid n > 3\}$

L'(G) = $\{a^n \mid n <= 3\}$

Note : Two regular expressions are equivalent if languages generated by them are same. For example, (a+b*)* and (a+b)* generate same language. Every string which is generated by (a+b*)* is also generated by (a+b)* and vice versa.

Pumping Lemma in Theory of Computation

There are two Pumping Lemmas, which are defined for

1. Regular Languages, and

2. Context – Free Languages

Pumping Lemma for Regular Languages

For any regular language L, there exists an integer n, such that for all $x \in L$ with $|x| \geq n$, there exists $u, v, w \in \Sigma*$, such that $x = uvw$, and
(1) $|uv| \leq n$
(2) $|v| \geq 1$
(3) for all $i \geq 0$: $uviw \in L$

In simple terms, this means that if a string v is ‘pumped’, i.e., if v is inserted any number of times, the resultant string still remains in L.

Pumping Lemma is used as a proof for irregularity of a language. Thus, if a language is regular, it always satisfies pumping lemma. If there exists at least one string made from pumping which is not in L, then L is surely not regular.
The opposite of this may not always be true. That is, if Pumping Lemma holds, it does not mean that the language is regular.
There are two Pumping Lemmas, which are defined for
1. Regular Languages, and
2. Context – Free Languages

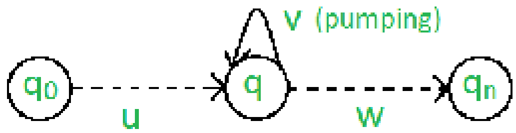
Pumping Lemma for Regular Languages

For any regular language L, there exists an integer n, such that for all $x \in L$ with $|x| \geq n$, there exists $u, v, w \in \Sigma*$, such that $x = uvw$, and
(1) $|uv| \leq n$
(2) $|v| \geq 1$
(3) for all $i \geq 0$: $uviw \in L$

In simple terms, this means that if a string v is ‘pumped’, i.e., if v is inserted any number of times, the resultant string still remains in L.

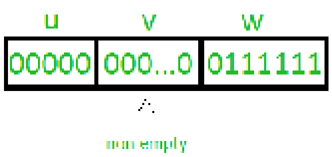
Pumping Lemma is used as a proof for irregularity of a language. Thus, if a language is regular, it always satisfies pumping lemma. If there exists at least one string made from pumping which is not in L, then L is surely not regular.
The opposite of this may not always be true. That is, if Pumping Lemma

holds, it does not mean that the language is regular.



For example, let us prove L01 = {0n1n | n ≥ 0} is irregular.
Let us assume that L is regular, then by Pumping Lemma the above given rules follow.
Now, let x ∈ L and |x| ≥ n. So, by Pumping Lemma, there exists u, v, w such that (1) – (3) hold.

We show that for all u, v, w, (1) – (3) does not hold.
If (1) and (2) hold then x = 0n1n = uvw with |uv| ≤ n and |v| ≥ 1.
So, u = 0a, v = 0b, w = 0c1n where : a + b ≤ n, b ≥ 1, c ≥ 0, a + b + c = n
But, then (3) fails for i = 0
uv0w = uw = 0a0c1n = 0a + c1n ∉ L, since a + c ≠ n.



Minimization of DFA

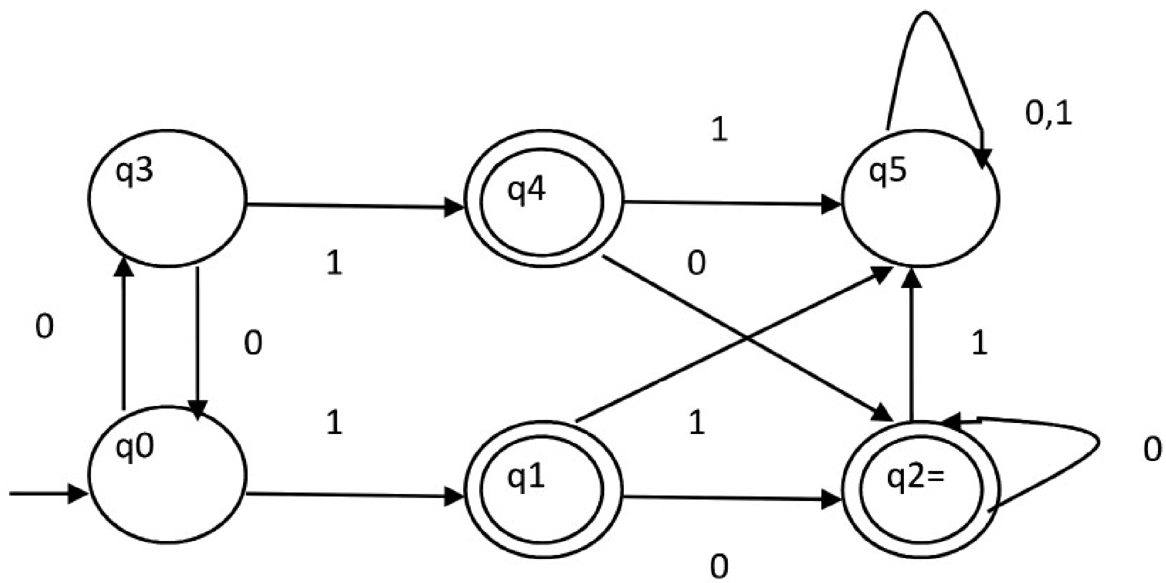
DFA minimization stands for converting a given DFA to its equivalent DFA with minimum number of states. DFA minimization is also called as Optimization of DFA and uses partitioning algorithm.

Minimization of DFA
Suppose there is a DFA D < Q, Σ, q0, δ, F > which recognizes a language L. Then the minimized DFA D < Q', Σ, q0, δ', F' > can be constructed for language L as:
Step 1: We will divide Q (set of states) into two sets. One set will contain all final states and other set will contain non-final states. This partition is called P0.
Step 2: Initialize k = 1
Step 3: Find Pk by partitioning the different sets of Pk-1. In each set of Pk-1, we will take all possible pair of states. If two states of a set are distinguishable, we will split the sets into different sets in Pk.
Step 4: Stop when Pk = Pk-1 (No change in partition)
Step 5: All states of one set are merged into one. No. of states in minimized DFA will be equal to no. of sets in Pk.

How to find whether two states in partition Pk are distinguishable ?
Two states (qi, qj) are distinguishable in partition Pk if for any input symbol a, δ (qi, a) and δ (qj, a) are in different sets in partition Pk-1.

Example

Consider the following DFA shown in figure.



Step 1. P0 will have two sets of states. One set will contain q1, q2, q4 which are final states of DFA and another set will contain remaining states. So P0 = { { q1, q2, q4 }, { q0, q3, q5 } }.
Step 2. To calculate P1, we will check whether sets of partition P0 can be partitioned or not:
i) For set { q1, q2, q4 } :
δ (q1, 0) = δ (q2, 0) = q2 and δ (q1, 1) = δ (q2, 1) = q5, So q1 and q2 are not distinguishable.
Similarly, δ (q1, 0) = δ (q4, 0) = q2 and δ (q1, 1) = δ (q4, 1) = q5, So q1 and q4 are not distinguishable.
Since, q1 and q2 are not distinguishable and q1 and q4 are also not distinguishable, So q2 and q4 are not distinguishable. So, { q1, q2, q4 } set will not be partitioned in P1.
ii) For set { q0, q3, q5 } :
δ (q0, 0) = q3 and δ (q3, 0) = q0
δ (q0, 1) = q1 and δ(q3, 1) = q4
Moves of q0 and q3 on input symbol 0 are q3 and q0 respectively which are in same set in partition P0. Similarly, Moves of q0 and q3 on input symbol 1 are q1 and q4 which are in same set in partition P0. So, q0 and q3 are not distinguishable.
δ (q0, 0) = q3 and δ (q5, 0) = q5 and δ (q0, 1) = q1 and δ (q5, 1) = q5
Moves of q0 and q5 on input symbol 1 are q1 and q5 respectively which are

in different set in partition P0. So, q0 and q5 are distinguishable. So, set { q0, q3, q5 } will be partitioned into { q0, q3 } and { q5 }. So, P1 = { { q1, q2, q4 }, { q0, q3}, { q5 } }

To calculate P2, we will check whether sets of partition P1 can be partitioned or not:

iii)For set { q1, q2, q4 } :

δ (q1, 0) = δ (q2, 0) = q2 and δ (q1, 1) = δ (q2, 1) = q5, So q1 and q2 are not distinguishable.

Similarly, δ (q1, 0) = δ (q4, 0) = q2 and δ (q1, 1) = δ (q4, 1) = q5, So q1 and q4 are not distinguishable.

Since, q1 and q2 are not distinguishable and q1 and q4 are also not distinguishable, So q2 and q4 are not distinguishable. So, { q1, q2, q4 } set will not be partitioned in P2.

iv)For set { q0, q3 } :

δ (q0, 0) = q3 and δ (q3, 0) = q0

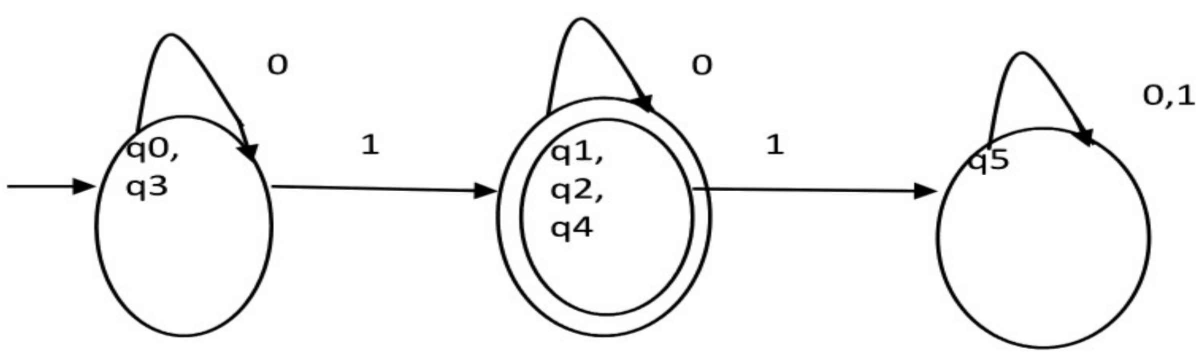
δ (q0, 1) = q1 and δ (q3, 1) = q4

Moves of q0 and q3 on input symbol 0 are q3 and q0 respectively which are in same set in partition P1. Similarly, Moves of q0 and q3 on input symbol 1 are q1 and q4 which are in same set in partition P1. So, q0 and q3 are not distinguishable.

v) For set { q5 }:

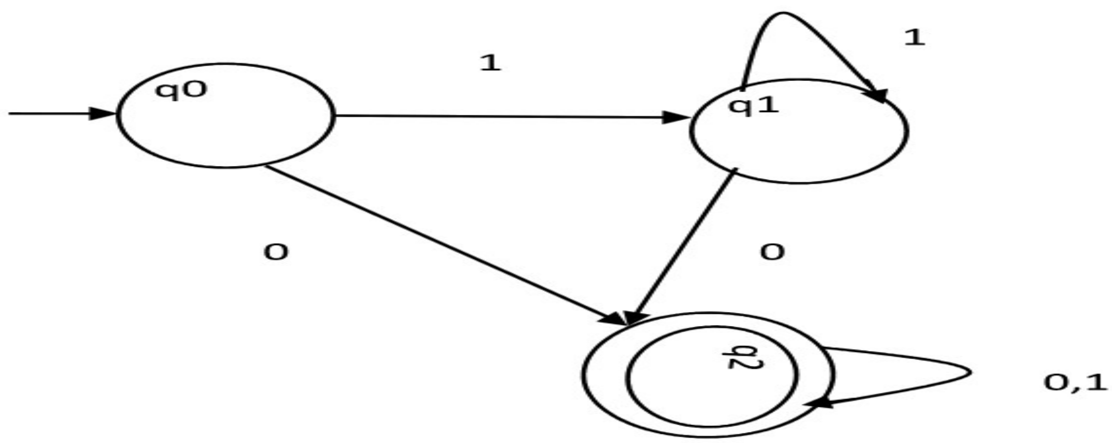
Since we have only one state in this set, it can’t be further partitioned. So, P2 = { { q1, q2, q4 }, { q0, q3 }, { q5 } }

Since, P1=P2. So, this is the final partition. Partition P2 means that q1, q2 and q4 states are merged into one. Similarly, q0 and q3 are merged into one. Minimized DFA corresponding to DFA of Figure 1 is shown in Figure 2 as:



Question : Consider the given DFA. Which of the following is false?

1. Complement of L(A) is context-free.
2. L(A) = L ((11 * 0 + 0) (0 + 1)* 0* 1*)
3. For the language accepted by A, A is the minimal DFA.
4. A accepts all strings over { 0, 1 } of length atleast two.



A. 1 and 3 only
B. 2 and 4 only
C. 2 and 3 only
D. 3 and 4 only

Solution : Statement 4 says, it will accept all strings of length atleast 2. But it accepts 0 which is of length 1. So, 4 is false.
Statement 3 says that the DFA is minimal. We will check using the algorithm discussed above.
P0 = { { q2 }, { q0, q1 } }
P1 = { q2 }, { q0, q1 } }. Since, P0 = P1, P1 is the final DFA. q0 and q1 can be merged. So minimal DFA will have two states. Therefore, statement 3 is also false.
So correct option is (D).

This article has been contributed by Sonal Tuteja.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.