UNIT-I: Introduction

Table of Content

Regular Expression

- The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language.
- The languages accepted by some regular expressions are referred to as Regular languages.
- A regular expression can also be described as a sequence of pattern that defines a string.
- Regular expressions are used to match character combinations in strings. String searching algorithm used this pattern to find the operations on a string.

For instance:

In a regular expression, x* means zero or more occurrence of x.

It can generate {e, x, xx, xxx, xxxx, ...}

In a regular expression, x^+ means one or more occurrence of x.

It can generate {x, xx, xxx, xxxx, ...}

Regular expression

- A regular expression is a sequence of characters that define a pattern.
- Notational shorthand's
  1. One or more occurrences: +
  2. Zero or more occurrences: *
  3. Alphabets: Σ

Regular expression

L = Zero or More Occurrences of a = a^*

*

ϵ
a
aa
aaa
aaaa
aaaaa.....

Infinite .....

Regular expression

L = One or More Occurrences of a = a^+

\+

a
aa
aaa
aaaa
aaaaa.....

Infinite .....

Operations on Regular Language

The various operations on regular language are:

Union: If L and M are two regular languages then their union L U M is also a union.

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

Intersection: If L and M are two regular languages then their intersection is also an intersection.

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

Kleen closure: If L is a regular language, then its Kleen closure L1* will also be a regular language.

L* = Zero or more occurrence of language L.

Example:

Write the regular expression for the language accepting all the string containing any number of a's and b's.

The regular expression will be:

R.E. = (a + b)*

This will give the set as L = {ε, a, aa, b, bb, ab, ba, aba, bab, ...},

any combination of a and b.

The (a + b)* shows any combination with a and b even a null string.

Regular expression examples

1. 0 or 1

   $Strings: 0, 1$ $\qquad$ $R.E. = 0 \mid 1$

2. 0 or 11 or 111

   $Strings: 0, 11, 111$ $\qquad$ $R.E. = 0 \mid 11 \mid 111$

3. String having zero or more a.

   $Strings: \epsilon, a, aa, aaa, aaaa \ldots$ $\qquad$ $R.E. = a^*$

4. String having one or more a.

   $Strings: a, aa, aaa, aaaa \ldots\ldots$ $\qquad$ $R.E. = a^+$

5. Regular expression over $\Sigma = \{a, b, c\}$ that represent all string of length 3.

   $Strings: abc, bca, bbb, cab, al$ $\qquad$ $R.E. = (a|b|c)\ (a|b|c)\ (a|b|c)$

6. All binary string.

   $Strings: 0, 11, 101, 10101, 11$ $\qquad$ $R.E. = (0 \mid 1)^+$

Regular expression examples

7. 0 or more occurrence of either a or b or both

   $Strings: \epsilon, a, aa, abab, bab \ldots$ $R.E. = (a \mid b)^*$

8. 1 or more occurrence of either a or b or both

   $Strings: a, aa, abab, bab, bbb$ $R.E. = (a \mid b)^+$

9. Binary no. ends with 0

   $Strings: 0, 10, 100, 1010, 111$ $R.E. = (0 \mid 1)^* 0$

10. Binary no. ends with 1

   $Strings: 1, 101, 1001, 10101, .$ $R.E. = (0 \mid 1)^* 1$

11. Binary no. starts and ends with 1

   $Strings: 11, 101, 1001, 10101$ $R.E. = 1 (0 \mid 1)^* 1$

12. String starts and ends with same character

   $Strings: 00, 101, aba, baab \ldots$ $R.E. = 1 (0 \mid 1)^* 1 \text{ or } 0 (0 \mid 1)^* 0$

   $a (a \mid b)^* a \text{ or } b (a \mid b)^* b$

Regular expression examples

13. All string of a and b starting with a

   $Strings: a, ab, aab, abb \ldots$ $R.E. = a(a \mid b)^*$

14. String of 0 and 1 ends with 00

   $Strings: 00, 100, 000, 1000, 1100 \ldots$ $R.E. = (0 \mid 1)^* 00$

15. String ends with abb

   $Strings: abb, babb, ababb \ldots$ $R.E. = (a \mid b)^* abb$

16. String starts with 1 and ends with 0

   $Strings: 10, 100, 110, 1000, 1100 \ldots$ $R.E. = 1(0 \mid 1)^* 0$

17. All binary string with at least 3 characters and 3rd character should be zero $Strings: 000, 100, 1100, 1001 \ldots$ $R.E. = (0|1)(0|1)0(0 \mid 1)*$

18. Language which consist of exactly two b's over the set $\Sigma = \{a, b\}$

   $Strings: bb, bab, aabb, abba \ldots$ $R.E. = a^* b a^* b a^*$

Regular expression examples

19. The language with $\Sigma = \{a, b\}$ such that 3rd character from right end of the string is always

    $\textit{Strings}: aaa, aba, aaba, abb...$ $R.E. = (a \mid b) * a(a|b)(a|b)$

20. Any no. of a followed by any no. of b followed by any no. of c

    $\textit{Strings}: \epsilon, abc, aabbcc, aabc, abb...$ $R.E. = a^* b^* c^*$

21. String should contain at least three 1

    $\textit{Strings}: 111, 01101, 0101110....$ $R.E. = (0|1)^*1\ (0|1)^*1\ (0|1)^*1\ (0|1)^*$

22. String should contain exactly two 1

    $\textit{Strings}: 11, 0101, 1100, 010010, 100100....$ $R.E. = 0^*10^*10^*$

23. Length of string should be at least 1 and at most 3

    $\textit{Strings}: 0, 1, 11, 01, 111, 010, 100....$ $R.E. = (0|1) \mid (0|1)(0|1) \mid (0|1)(0|1)(0|1)$

24. No. of zero should be multiple of 3

    $\textit{Strings}: 000, 010101, 110100, 000000, 100010010....$ $R.E. = (1^*01^*01^*01^*)^*$

Regular expression examples

25. The language with $\Sigma = \{a, b, c\}$ where a should be multiple of 3

    $Strings: aaa, baaa, bacaba, a$ $R.E. = ((b|c)^* a(b|c)^* a(b|c)^* a(b|c)^*)^*$

26. Even no. of 0

    $Strings: 00, 0101, 0000, 100100....$ $R.E. = (1^*01^*01^*)^*$

27. String should have odd length

    $Strings: 0, 010, 110, 000, 10010....$ $R.E. = (0|1)\ ((0|1)(0|1))^*$

28. String should have even length

    $Strings: 00, 0101, 0000, 100100....$ $R.E. = ((0|1)(0|1))^*$

29. String start with 0 and has odd length

    $Strings: 0, 010, 010, 000, 00010....$ $R.E. = (0)\ ((0|1)(0|1))^*$

30. String start with 1 and has even length

    $Strings: 10, 1100, 1000, 100100....$ $R.E. = 1(0|1)((0|1)(0|1))^*$

Regular expression examples

31. All string begins or ends with 00 or 11

    $Strings: 00101, 10100, 110, ($ $R.E. = (00|11)(0\mid 1)^* \mid (0|1)^*(00|11)$

32. Language of all string containing both 11 and 00 as substring

    $Strings: 0011, 1100, 100110,$

    $R.E. = ((0|1)^*00(0|1)^*11(0|1)^*) \mid ((0|1)^*11(0|1)^*00(0|1)^*)$

33. String ending with 1 and not contain 00

    $Strings: 011, 1101, 1011 ....$ $R.E. = (1|01)^+$

34. Language of C identifier

    $Strings: area, i, redious, gra$ $R.E. = (\_ + L)(\_ + L + D)^*$

    $where\ L\ is\ Letter\ \&\ D\ is\ digit$

Example:

Write the regular expression for the language accepting all the string which are starting with 1 and ending with 0, over ∑ = {0, 1}.

Solution:

In a regular expression, the first symbol should be 1, and the last symbol should be 0. The R.E. is as follows:

R = 1 (0+1)* 0

Example :

Write the regular expression for the language starting with a but not having consecutive b's.

Solution: The regular expression has to be built for the language:

L = {a, aba, aab, aba, aaa, abab, ...}

The regular expression for the above language is:

R = {a + ab}*

Conversion of RE to FA

To convert the RE to FA, we are going to use a method called the subset method. This method is used to obtain FA from the given regular expression. This method is given below:

Step 1: Design a transition diagram for given regular expression, using NFA with ε moves.

Step 2: Convert this NFA with ε to NFA without ε.

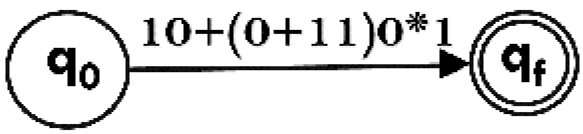
Step 3: Convert the obtained NFA to equivalent DFA.

Example 1:

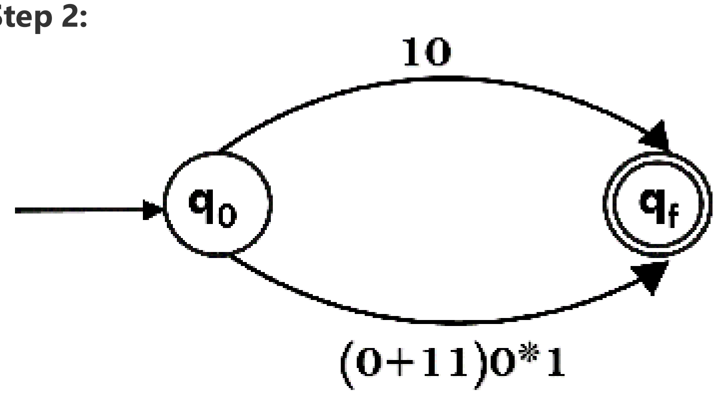
Design a FA from given regular expression 10 + (0 + 11)0* 1.

Solution: First we will construct the transition diagram for a given regular expression.

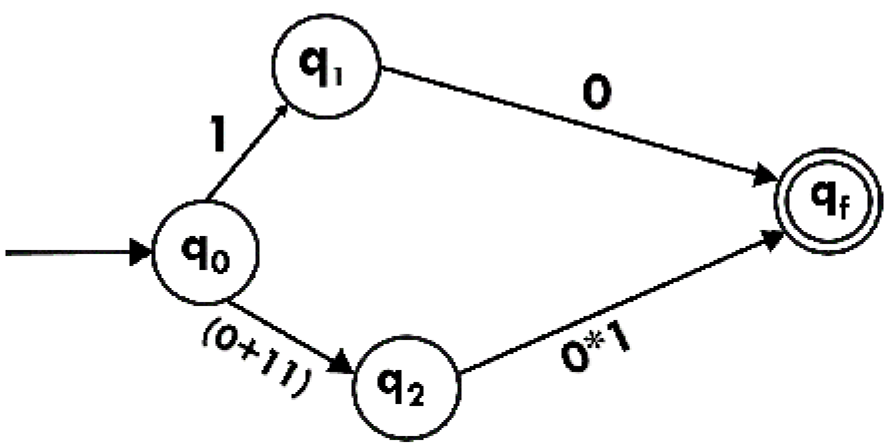
Step 1:



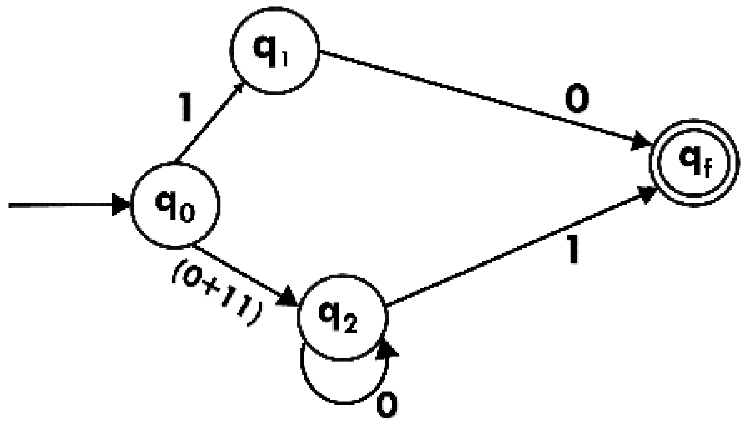
Step 2:



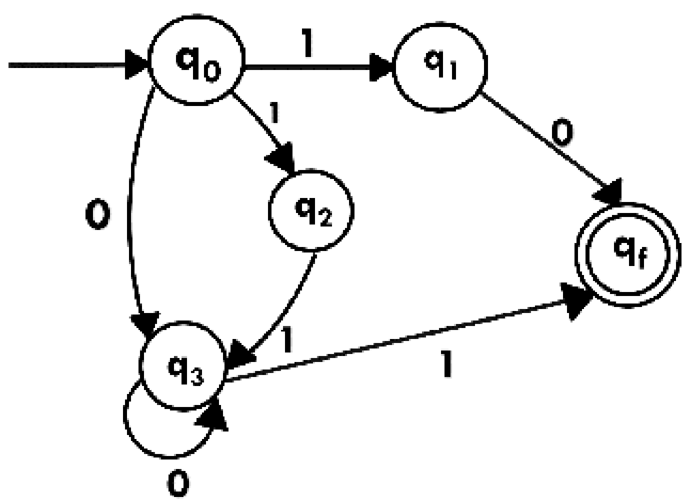
Step 3:



Step 4:



Step 5:



Now we have got NFA without ε. Now we will convert it into required DFA for that, we will first write a transition table for this NFA.

State	0	1
→q0	q3	{q1, q2}
q1	qf	ϕ
q2	ϕ	q3
q3	q3	qf
*qf	ϕ	ϕ

The equivalent DFA will be:

State	0	1
→[q0]	[q3]	[q1, q2]
[q1]	[qf]	ϕ
[q2]	ϕ	[q3]
[q3]	[q3]	[qf]
[q1, q2]	[qf]	[qf]
*[qf]	ϕ	Φ

Applications of FA

- Lexical analysis phase of a compiler.
- Design of digital circuit.
- String matching.
- Communication Protocol for information exchange.

Arden's Theorem

The Arden's Theorem is useful for checking the equivalence of two regular expressions as well as in the conversion of DFA to a regular expression.

Let us see its use in the conversion of DFA to a regular expression.

Following algorithm is used to build the regular expression form given DFA.

1. Let q_1 be the initial state.

2. There are q_2, q_3, q_4 ....q_n number of states. The final state may be some q_j where j <= n.
3. Let α_{ji} represents the transition from q_j to q_i.
4. Calculate q_i such that

$$q_i = \alpha_{ji} * q_j$$

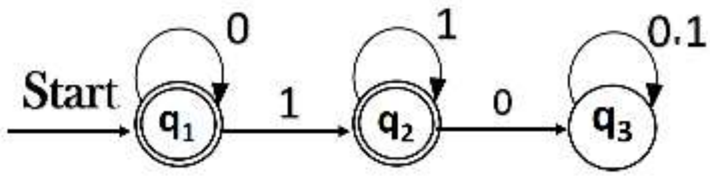
If q_j is a start state then we have:

$$q_i = \alpha_{ji} * q_j + \varepsilon$$

5. Similarly, compute the final state which ultimately gives the regular expression 'r'.

Example:

Construct the regular expression for the given DFA



$q1 = q1\ 0 + \varepsilon$

Since q1 is the start state, so ε will be added, and the input 0 is coming to q1 from q1 hence we write
State = source state of input × input coming to it

Similarly,

$q2 = q1\ 1 + q2\ 1$
$q3 = q2\ 0 + q3\ (0+1)$

Since the final states are q1 and q2, we are interested in solving q1 and q2 only. Let us see q1 first

$q1 = q1\ 0 + \varepsilon$

We can re-write it as

$q1 = \varepsilon + q1\ 0$

Which is similar to R = Q + RP, and gets reduced to R = OP*.

Assuming R = q1, Q = ε, P = 0

We get

$q1 = \varepsilon.(0)^*$
$q1 = 0^* \quad (\varepsilon.R^* = R^*)$

Substituting the value into q2, we will get

$q2 = 0^*\ 1 + q2\ 1$
$q2 = 0^*\ 1\ (1)^* \quad (R = Q + RP \rightarrow Q\ P^*)$

The regular expression is given by

$r = q1 + q2$
$= 0^* + 0^*\ 1.1^*$
$r = 0^* + 0^*\ 1^+ \quad (1.1^* = 1^+)$

Pumping lemma for Regular languages

- It gives a method for pumping (generating) many substrings from a given string.
- In other words, we say it provides means to break a given long input string into several substrings.
- Lt gives necessary condition(s) to prove a set of strings is not regular.

Theorem

For any regular language L, there exists an integer P, such that for all w in L

|w|>=P

We can break w into three strings, w=xyz such that.

(1)lxyl < P

(2)lyl > 1

(3)for all k>= 0: the string xy^kz is also in L

Application of pumping lemma

Pumping lemma is to be applied to show that certain languages are not regular.

It should never be used to show a language is regular.

- If L is regular, it satisfies the Pumping lemma.
- If L does not satisfy the Pumping Lemma, it is not regular.

Steps to prove that a language is not regular by using PLare as follows−

- step 1 − We have to assume that L is regular
- step 2 − So, the pumping lemma should hold for L.

- step 3 – It has to have a pumping length (say P).
- step 4 – All strings longer that P can be pumped |w|>=p.
- step 5 – Now find a string 'w' in L such that |w|>=P
- step 6 – Divide w into xyz.
- step 7 – Show that $xy^iz \notin L$ for some i.
- step 8 – Then consider all ways that w can be divided into xyz.
- step 9 – Show that none of these can satisfy all the 3 pumping conditions at same time.
- step 10 – w cannot be pumped = CONTRADICTION.

Example of Proof Idea of the Pumping Lemma[3] [top]

The integer p associated with the pumping lemma is just the number of states a DFA that recognizes the regular language in question.

```
             a          b          c
---> [q0] ---> [q1] ---> [q2] ---> [[q3]]
                ^                    |
                |                    |
                +--------------------+
                         a

p = 4

So we need a string of length at least 4

w = abcabc is accepted and has length 6 >= 4.

The first state that is repeated when w is input to this DFA
is  q1.

x = string up to the first occurrence of repeated state q1
y = string after x up to the second occurence of q1

So x = a and y = bca which means z = bc

w = |a|bca|bc|
     x  y   z
```

Pumping lemma says these string are also in the language

$w_0 = xy^0z = xz = a\ bc$
$w_1 = xy^1z = xyz = a\ \underline{bca}\ bc$
$w_2 = xy^2z = xyyz = a\ \underline{bca}\ \underline{bca}\ bc$

Example 1 Using the Pumping Lemma[4] [top]

```
L = { a^i b a^j | 0 <= i < j }
```

Proof is by contradiction, using the pumping lemma to get the contradiction.

Assume L is regular and let p be the constant given by the pumping lemma.

The string $w = a^p b a^{p+1}$ is in L and has length > p.

By the pumping lemma w = xyz with |xy| <= p and |y| > 0.

But this means the prefix xy must come before the 'b' and consist only of a's.

So this means y consists of 1 or more a's (x might be empty).

By the pumping lemma, $w_2 = xy^2z$ must also be in L, but the number of a's before the b in w_2 must be at least p + 1, while the number of a's after b is still p + 1.

But this contradicts the condition for w_2 being in L and so the assumption that L is regular is false.

Example 2 Using the Pumping Lemma[5] [top]

```
L = { a^i b a^j | i >  j >= 0 }
```

Proof is by contradiction again, using the pumping lemma to get the contradiction, but has to work slightly differently.

Adding more y's will not work because now the condition is i > j., the leading a's are greater in number than the ones after the b.

Assume L is regular and let p be the constant given by the pumping lemma.

The string $w = a^{p+1}ba^p$ is in L and has length > p.

By the pumping lemma w = xyz with |xy| <= p and |y| > 0.

But this again means the prefix xy must come before the 'b' and consist only of a's.

So this means y consists of 1 or more a's (x might be empty).

By the pumping lemma, w_0 = xz must also be in L, but the number of a's before the b in w_0 must be no more than p since we have removed at least 1 a from w to get w_0. But the number of a's after b in w_0 is still p.

But this contradicts the condition for w_0 being in L and so the assumption that L is regular is false.