Unit 1: Chapter 3

(Regular Expression (RE) and Language)

In previous lectures, we have described the languages in terms of machine like description-finite automata (DFA or NFA). Now we switch our attention to an algebraic description of languages, called regular expression.

Regular Expression are those algebraic expressions used for representing regular languages, the languages accepted by finite automaton. Regular expressions offer a declarative way to express the strings we want to accept. This is what the regular expression offer that the automata do not. Many system uses regular expression as input language. Some of them are:

- Search commands such as UNIX grep.
- Lexical analyzer generator such as LEX or FLEX. Lexical analyzer is a component of compiler that breaks the source program into logical unit called tokens.

Defining Regular expressions

A regular expression is built up out of simpler regular expression using a set of defining rules. Each regular expression 'r' denotes a language L(r). The defining rules specify how L(r) is formed by combining in various ways the languages denoted by the sub expressions of 'r'.

Here is the method:

Let Σ be an alphabet, the regular expression over the alphabet Σ are defined inductively as follows;

Basic steps:

-Φ is a regular expression representing empty language.
-Є is a regular expression representing the language of empty strings. i.e.{Є}
- if 'a' is a symbol in Σ, then 'a' is a regular expression representing the language {a}.

Now the following operations over basic regular expression define the complex regular expression as:

-if 'r' and 's' are the regular expressions representing the language L(r) and L(s) then

- r U s is a regular expression denoting the language L(r) U L(s).
- r.s is a regular expression denoting the language L(r).L(s).
- r* is a regular expression denoting the language (L(r))*.
- (r) is a regular expression denoting the language (L(r)). (this denote the same language as the regular expression 'r' denotes.

Note: any expression obtained from Φ, Є, a using above operation and parenthesis where required is a regular expression.

Regular operator:

Basically, there are three operators that are used to generate the languages that are regular,

Union (U / | /+): If L_1 and L_2 are any two regular languages then

$L_1 \cup L_2 = \{s \mid s \in L_1, \text{ or } s \in L_2\}$

For Example:
$L_1 = \{00, 11\}$, $L_2 = (Є, 10\}$
$L_1 \cup L_2 = \{Є, 00, 11, 10\}$

Concatenation (.): If L_1 and L_2 are any two regular languages then,
$L_1.L_2 = \{l_1.l_2 | l_1 \in L_1 \text{ and } l_2 \in L_2\}$

For examples: L1 = {00, 11} and L2 = {Є, 10}
L1.L2={00,11,0010,1110}
L2.L1={1000,1011,00,11}
So L1.L2 !=L2.L1

Kleen Closure (*):

If L is any regular Language then,
$L^* = L_i = L_0 \cup L_1 \cup L_2 \cup \ldots\ldots\ldots\ldots$

Precedence of regular operator:

1. The star operator is of highest precedence. i.e it applies to its left well formed RE.
2. Next precedence is taken by concatenation operator.
3. Finally, unions are taken.

Examples: Write a RE for the set of string that consists of alternating 0's and 1's over {0,1}.

First part: we have to generate the language {01,0101,0101,……}
Second part we have to generate the language {10,1010,101010…….}

So lets start first part.
Here we start with the basic regular expressions 0 and 1 that represent the language {0} and {1} respectively.

Now if we concatenate these two RE, we get the RE 01 that represent the language {01}.
Then to generate the language of zero or more occurrence of 01, we take Kleen closure. i.e. the RE (01)* represent the language {01,0101,……..}
Similarly, the RE for second part is (10)*.
Now finally we take union of above two first part and second part to get the required RE. i.e. the RE (01)*+(10)* represent the given language.

Regular language:

Let Σ be an alphabet, the class of regular language over Σ is defined inductively as;
- Φ is a regular language representing empty language
- {Є} is a regular language representing language of empty strings.
- For each a ε Σ, {a} is a regular language.
- If $L_1, L_2 \ldots\ldots\ldots\ldots L_n$ is regular languages, then so is $L_1 U L_2 U \ldots\ldots\ldots UL_n$.
- If $L_1, L_2, L_3, \ldots\ldots\ldots\ldots L_n$ are regular languages, then so is $L_1.L_2.L_3\ldots\ldots\ldots L_n$
-If L is a regular language, then so is L*

Note: strictly speaking a regular expression E is just expression, not a language. We should use L(E) when we want to refer to the language that E denotes. However it is to common to refer to say E when we really mean L(E).

Application of regular languages:

Validation: Determining that a string complies with a set of formatting constraints. Like email address validation, password validation etc.

Search and Selection: Identifying a subset of items from a larger set on the basis of a pattern match.

Tokenization: Converting a sequence of characters into words, tokens (like keywords, identifiers) for later interpretation.

Algebraic Rules/laws for regular expression

1. ***Commutativity:*** *Commutative of oerator means we can switch the order of its operands and get the same result. The union of regular expression is commutative but concatenation of regular expression is not commutative. i.e. if r and s are regular expressions representing like languages L(r) and L(s) then, r+s =s+r i.e.r U s = s U r but r.s ≠s.r.*

2. *Associativity: The unions as well as concatenation of regular expressions are associative. i.e. if t, r, s are regular expressions representing regular languages L(t),L(r) and L(s) then,*
   t+(r+s) = (t+r)+s
   And t.(r.s) = (t.r).s

3. ***Distributive law**: For any regular expression r,s,t representing regular language L(r), L(s) and L(t) then,*
   r(s+t) = rs+rt ------ left distribution.
   (s+t)r = sr+tr ------ right distribution.

4. ***Identity law**: Φ is identity for union. i.e. for any regular expression r representing regular expression L(r).*

   r + Φ = Φ + r = r i.e. ΦUr=r.
   Ɛ is identity for concatenation. i.e. Ɛ.r = r = r. Ɛ

5. ***Annihilator**: An annihilator for an operator is a value such that when the operator is applied to the annihilator and some other value, the result is annihilator. Φ is annihilator for concatenation.*
   i.e. Φ.r = r. Φ = Φ
6. ***Idempotent law of union**: For any regular expression r representing the regular language L(r), r + r = r. This is the idempotent law of union.*
7. ***Law of closure**: for any regular expression r, representing the regular language L(r), then*
   -(r)*=r**
   -Closure of Φ = Φ = Ɛ*
   -Closure of Ɛ = Ɛ = Ɛ*
   *-Positive closure of r, r_+ = rr**.

Examples

Consider Σ = {0, 1}, then some regular expressions over Σ are ;

- 0*10* is RE that represents language {w|w contains a single 1}
- $\Sigma^*1\Sigma^*$ is RE for language{w|w contains at least single 1}
- Σ*001 Σ* = {w|w contains the string 001 as substring}
- (Σ Σ)* or ((0+1)*.(0+1)*) is RE for {w|w is string of even length}
- 1*(01*01*)* is RE for {w|w is string containing even number of zeros}
- 0*10*10*10* is RE for {w|w is a string with exactly three 1's}

- For string that have substring either 001 or 100, the regular expression is (1+0)*.001.(1+0)*+(1+0)*.(100).(1+0)*
- For strings that have at most two 0's with in it, the regular expression is 1*.(0+Є).1*.(0+Є).1*
- For the strings ending with 11, the regular expression is $(1+0)^*.(11)^+$
- Regular expression that denotes the C identifiers:

  (Alphabet + _)(Alphabet + digit + _)*

Theorem 1
If L, M and N are any languages, then L(M U N) = LM U LN.

Proof:
Let w = xy be a string, now to prove the theorem it is sufficient to show that 'w' ε LM U LN.

Now first consider "if part":

Let w ε LM U LN This implies that, w ε L(M) or w ε L(N) (by union rule)
i.e. xy ε LM or xy ε LN

Also,
xy ε LM implies x ε L and y ε M (by concatenation rule)

And,
xy ε LN implies x ε L and y ε N (by concatenation rule)

This implies that
x εL and y ε (M U N) then xy ε L(M U N) (concatenating above)

This implies that w ε L(M U N)

Now consider "only if" part:
Let w ε L(M U N) => xy ε L(M U N)

Now,
xy ε L(M U N) => x ε L and y ε (M U N) (by concatenation)
y ε (M U N) => y ε M or y ε N (by union rule)

Now, we have x ε L
Here if y ε M then xy ε L(M) (by concatenation)
And if y ε N then xy ε L(N) (by concatenation)

Thus, if xy ε L(M) => xy ε (L(M) U L(N)) (by union rule)
xy ε L(N) i.e. w ε (L(M) U L(N)

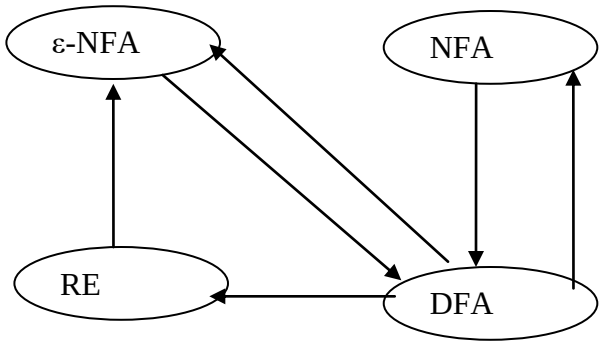
Thus,
We have, L (M U N) = L(M) U L(N)

Finite Automata and Regular expression

The regular expression approach for describing language is fundamentally different from the finite automaton approach. However, these two notations turn out to represent exactly the same set of languages, which we call regular languages. In order to show that the RE define the same class of language as Finite automata, we must show that:

1)Any language define by one of these finite automata is also defined by RE.
2)Every language defined by RE is also defined by any of these finite automata.

We can proceed as:



1. RE to NFA conversion

   We can show that every language L(R) for some RE R, is also a language L(E) for some epsilon NFA. This say that both RE and epsilon-NFA are equivalent in terms of language representation.

Theorem 1
Every language defined by a regular expression is also defined by a finite automaton. [For any regular expression r, there is an Є-NFA that accepts the same language represented by r].

Proof:

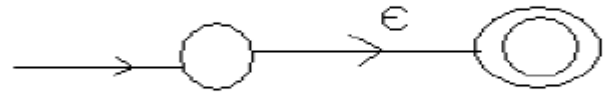
Let L =L(r) be the language for regular expression r, now we have to show there is an Є-NFA E such that L (E) =L.

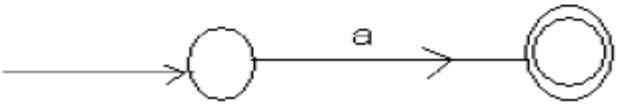
The proof can be done through structural induction on r, following the recursive definition of regular expressions.

For this we know Φ, Є, 'a' are the regular expressions representing languages {Φ}; an empty language, { Є };language for empty strings and {a} respectively.

The Є-NFA accepting these languages can be constructed as;

1.  for r = Φ

2.  for r = Є

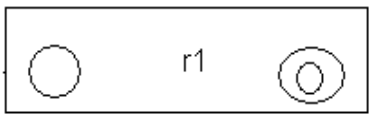
3.  for r= a

This forms the basis steps.

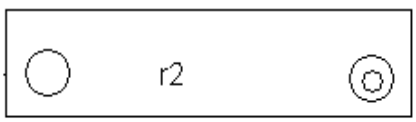
Now the induction parts are shown below

Let r be a regular expression representing language L(r) and r1,r2 be regular expressions for languages L(r1) and L(r2),

1. For union '+': From basis step we can construct Є-NFA's for r1 and r2. Let the Є-NFA's be M1 and M2 respectively

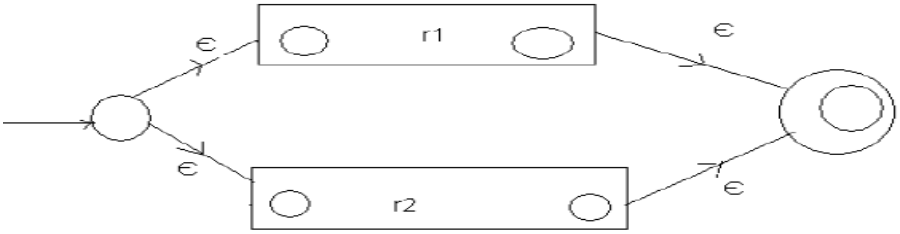


M1



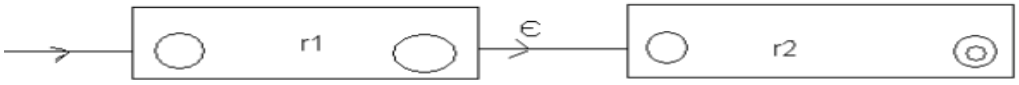
M2

Then, $r=r_1+r_2$ can be constructed as:



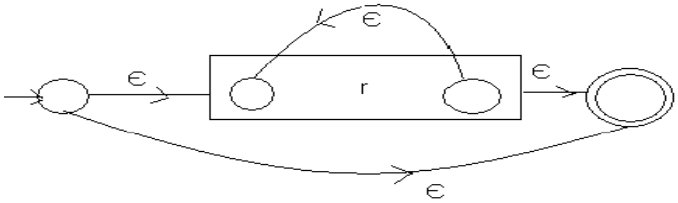
The language of this automaton is L(r1) U L(r2) which is also the language represented by expression r1+r2.

2. For concatenation '.': Now, $r = r_1.r_2$ can be constructed as;



Here, the path from starting to accepting state go first through the automaton for r1, where it must follow a path labeled by a string in L(r1), and then through the automaton for r2, where it follows a path labeled by a string in L(r2). Thus, the language accepted by above automaton is L(r1).L(r2).

3. For *(Kleen closure)
Now, r* Can be constructed as;



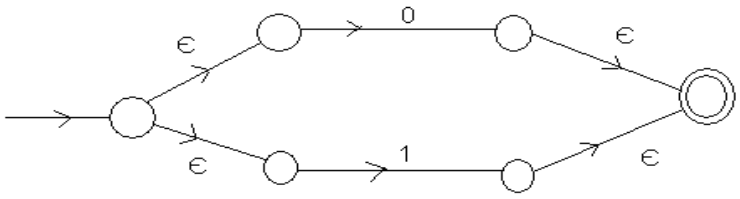
Clearly language of this Є-NFA is L(r*) as it can also just Є as well as string in L(r), L(r)L(r), L(r)L(r)L(r) and so on. Thus covering all strings in L(r*).

Finally, for regular expression (r), the automaton for r also serves as the automaton for (r), since the parentheses do not change the language defined by the expression.

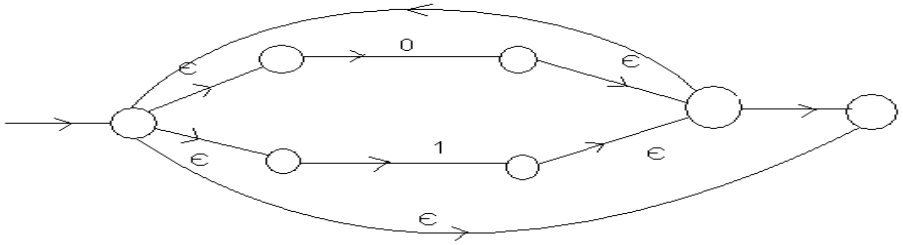
This completes the proof.

Examples (Conversion from RE to Ɛ-NFA)

1. For regular expression (1+0) the Ɛ-NFA is:

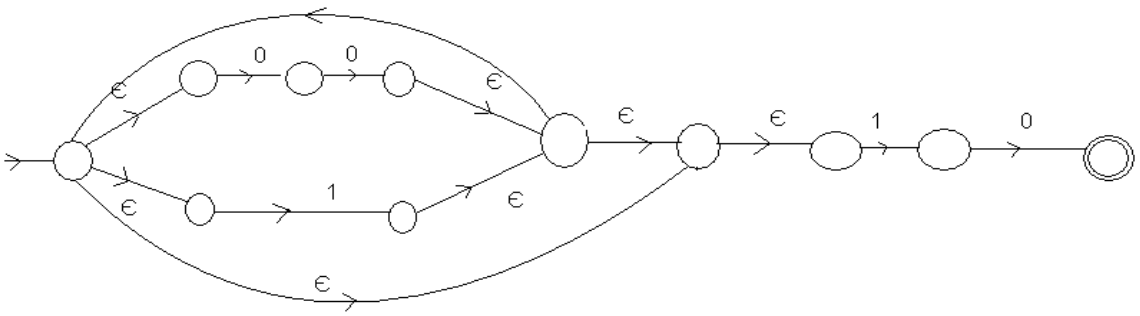


2. for (0+1)*, the Ɛ-NFA is:



3. Now, Ɛ-NFA for whole regular expression (0+1)*1(0+1)

4. For regular expression (00+1)*10 the Ɛ-NFA is as:



Conversion from DFA to Regular Expression(DFA to RE)

Arden's Theorem

Let p and q be the regular expressions over the alphabet Σ, if p does not contain any empty string then r = q + rp has a unique solution r = qp*.

Proof:

Here, r = q + rp …… (i)
Let us put the value of r = q + rp on the right hand side of the relation (i), so;
r = q + (q + rp)p
$r = q + qp + rp^2$……(ii)

Again putting value of r = q + rp in relation (ii), we get;
$r = q + qp + (q + rp)\, p^2$
$r = q + qp + qp^2 + qp^3$……

Continuing in the same way, we will get as;
$r = q + qp + qp^2 + qp^3$……..
$r = q(Є + p + p^2 + p^3 +$……..

Thus r = qp* Proved.

Use of Arden's rule to find the regular expression for DFA:

To convert the given DFA into a regular expression, here are some of the assumptions regarding the transition system:

- The transition diagram should not have the Є-transitions.
-There must be only one initial state.
-The vertices or the states in the DFA are as;

$q_1, q_2, \ldots\ldots\ldots\ldots q_n$ (Any qi is final state)

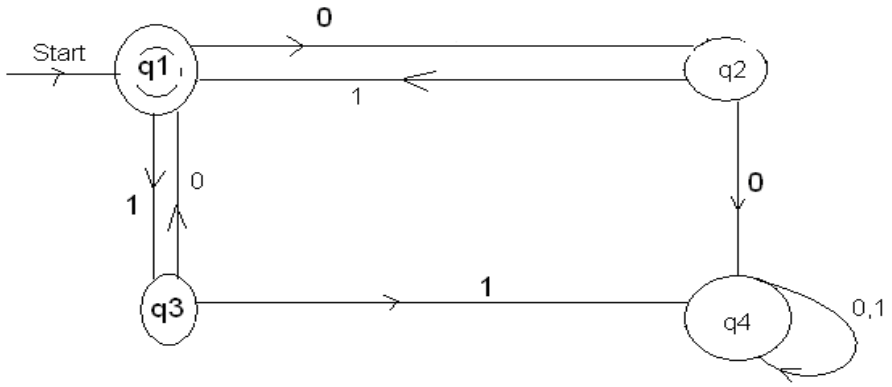
-W_{ij} denotes the regular expression representing the set of labels of the edjes from q_i to q_j.

Thus we can write expressions as;

$q_1 = q_1w_{11} + q_2w_{12} + q_3w_{31} + \ldots\ldots\ldots\ldots\ldots q_nw_{n1} + Є$
$q_2 = q_1w_{12} + q_2w_{22} + q_3w_{32} + \ldots\ldots\ldots\ldots\ldots + q_nw_{n2}$
$q_3 = q_1w_{13} + q_2w_{23} + q_3w_{33} + \ldots\ldots\ldots\ldots\ldots + q_nw_{n3}$
……
……
……
$q_n = q_1w_{1n} + q_2w_{n2} + q_3w_{n3} + \ldots\ldots\ldots\ldots\ldots q_nw_{nn}$

Solving these equations for q_i in terms of w_{ij} gives the regular expression eqivalent to given DFA.

Examples: Convert the following DFA into regular expression.



Let the equations are:
q1=q21+q30+ Є…….(i)
q2=q10……(ii)
q3=q11……..(iii)
q4=q20+q31+q40+ q41……(iv)

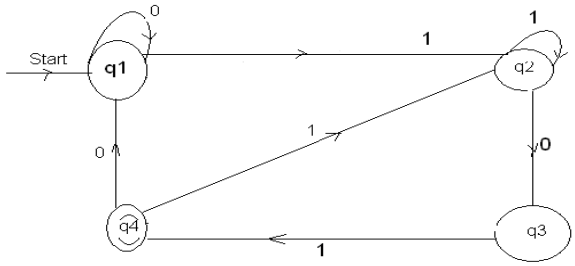
Putting the values of q2 and q3 in (i)

q1=q101+q110+ Є
i.e.q1=q1(01+10)+ Є
i.e.q1= Є+q1(01+10) (since r = q+rp)
i.e. q1= Є(01+10)* (using Arden's rule)

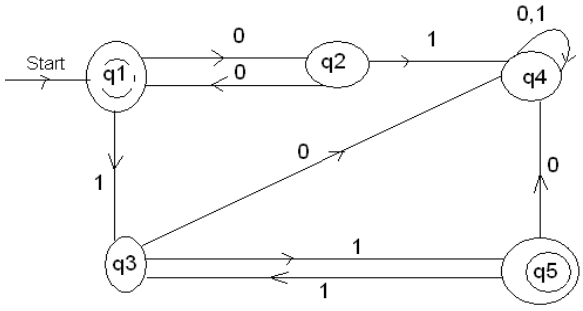
Since, q1 is final state, the final regular expression for the DFA is
Є(01+10)* = (01+10)*

Exercise: Try some Question from text book as well as from Adesh Kumar Pandey's book.Here I have maintion some of them.

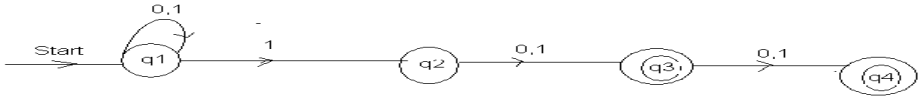
1. Convert this DFA into RE.



2.Convert this DFA into RE



3.Convert this NFA into RE



Proving Langauge not to be Regular

It is shown that the class of language known as regular language has at least four different descriptions. They are the language accepted by DFA's, by NFA's, by Є-NFA, and defined by RE.
Not every language is Regular. To show that a langauge is not regular, the powerfull technique used is known as Pumping Lemma.

Pumping Lemma

Statement: Let L be a regular language. Then, there exists an integer constant n so that for any x ε L with $|x| \geq n$, there are strings u, v, w such that x = uvw, $|uv| \leq n$, $|v| > 0$.
Then $uv^k w \varepsilon L$ for all $k \geq 0$.

Note: Here y is the string that can be pumped i.e repeating y any number of times or deleting it, keeps the resulting string in the language.

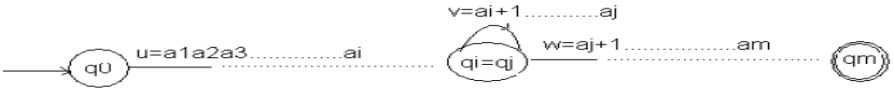
Proof:

Suppose L is a regular language, then L is accepted by some DFA M. Let M has n states. Also L is infinite so M accepts some string x of length n or greater. Let length of x, |x| =m where $m \geq n$.

Now suppose;
$X = a_1a_2a_3\ldots\ldots\ldots\ldots a_m$ where each $a_i \varepsilon \Sigma$ be an input symbol to M. Now, consider for j = 1,……n, qj be states of M
Then,

$\hat{\delta}(q_0,x) = \hat{\delta}(q_0,a_1a_2\ldots\ldots\ldots a_m)$ [q0 being start state of M]
$= \hat{\delta}(q_1,a_2\ldots\ldots\ldots a_m)$
=……
=……
=……
$= \hat{\delta}(q_m,Є)$ [qm being final state]

Since $m \geq n$, and DFA M has only n states, so by pigeonhole principle, there exists some i and j; $0 \leq i < j \leq m$ such that $q_i = q_j$.



Now we can break x=uvw as
u = a1a2……..ai
v =ai+1……aj
w =aj+1……am

i.e. string ai+1 ……aj takes M from state qi back to itself since qi = qj. So we can say M accepts $a_1a_2\ldots\ldots\ldots a_i(a_{i+1}\ldots\ldots\ldots a_j)^K a_{j+1}\ldots\ldots\ldots\ldots a_m$ for all k≥0.

Hence, uvkw ε L for all k≥0.

Application of Pumping Lemma:

To prove any language is not a regular language.

For example:

1. Show that language, L={0_r 1_r|n ≥0} is not a regular language.

Solution:

Let L is a regular language. Then by pumping lemma, there are strings u, v, w with v≥1 such that uv^kw ε L for k≥0.

Case I:

Let v contain 0's only. Then, suppose $u = 0^p$, $v = 0^q$,$w = 0^r1^s$;
Then we must have p+q+r = s (as we have 0^r1^r) and q>0

Now, $uv^kw = 0^p(0^q)^k0^r1^s=0^{p+qk+r}1^s$
Only these strings in $0^{p+qk+r}1^s$ belongs to L for k=1 otherwise not.
Hence we conclude that the language is not regular.

Case II

Let v contains 1's only. Then $u= 0^p1^q$, $v= 1^r$, $w=1^s$
Then p= q+r+s and r>0

Now, $0^p1^q(1^r)^k1^s=0^p1^{q+rk+s}$
Only those strings in 0^p1^{q+rk+s} belongs to L fpr k =1 otherwise not.
Hence the language is not regular.

Case III

V contains 0's and 1's both. Then, suppose,
$u = 0^p$, $v = 0^q1^r$, $w = 1^s$;
p+q = r+s and q+r>0

Now, $uv^kw = 0^p(0^q1^r)^k1^s= 0^{p+qk}1^{rk+s}$
Only those strings in $0^{p+qk}1^{rk+s}$ belongs to L for k=1, otherwise not. (As it contains 0 after 1 for k>1 in the string.)
Hence the language is not regular.

Minimization of DFA

Given a DFA M, that accepts a language L (M). Now, configure a DFA M '. During the course of minimization, it involves identifying the equivalent states and distinguishable states.

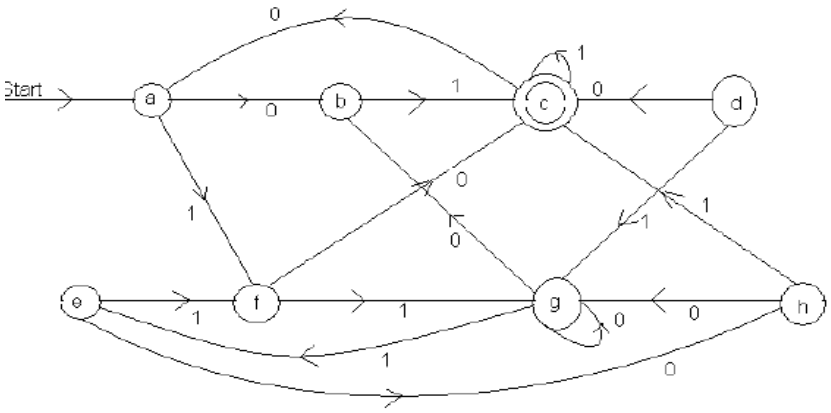
Equivalent States: Two states p & q are called equivalent states, denoted by p ≡ q if and only if for each input string x, $\hat{\delta}$ (p, x) is a final state if and only if $\hat{\delta}$(q, x) is a final state.

Distinguishable state: Two states p & q are said to be distinguishable states if (for any) there exists a string x, such that $\hat{\delta}$(p, x) is a final state $\hat{\delta}$(q, x) is not a final state.

For minimization, the table filling algorithm is used. The steps of the algorithm are;
For identifying the pairs (p, q) with p ≠ q;

- List all the pairs of states for which p ≠ q.
- Make a sequence of passes through each pairs.
- On first pass, mark the pair for which exactly one element is final (F).
- On each sequence of pass, mark the pair (r, s) if for any a ε Σ, δ(r, a) = p and δ(s, a) = q and (p, q) is already marked.
- After a pass in which no new pairs are to be marked, stop
- Then marked pairs (p, q) are those for which p q and unmarked pairs are those for which p ≡ q.

Example



Now to solve this problem first we should determine weather the pair is distinguishable or not.

	a	b	c	d	e	f	g
b	x						
c	x	x					
d	x	x	x				
e		x	x	x			
f	x	x	x		x		
g	x	x	x	x	x	x	
h	x		x	x	x	x	x

For pair (b, a)
$(\delta(b, 0), \delta(a, 0)) = (g, h)$ – unmarked
$(\delta(b, 1), \delta(a, 1)) = (c, f)$ – marked

For pair (d, a)
$(\delta(d, 0), \delta(a, 0)) = (c, b)$ – marked
Therefore (d, a) is distinguishable.

For pair (e, a)
$(\delta(e, 0), \delta(a, 0)) = (h, h)$ – unmarked.
$(\delta(e, 1), \delta(a, 1)) = (f, f)$ –unmarked.
[(e, a) is not distinguishable)]

For pair (g, a)
$(\delta(g, 0), \delta(a, 0)) = (a, g)$ – unmarked.
$(\delta(g, 1), \delta(a, 1)) = (e, f)$ – unmarked

For pair (h, a)
$(\delta(h, 0), \delta(a, 0)) = (g, h)$ –unmarked
$(\delta(h, 1), \delta(a\ 1) = (c, f)$ – marked
Therefore (h, a) is distinguishable.

For pair (d, b)
$(\delta(d, 0), \delta(b,0)) = (c, g)$ – marked
Therefore (d, b) is distinguishable.

For pair (e, b)
$(\delta(e, 0), \delta(b,0)) = (h, g)$ –unmarked
$(\delta(e, 1), \delta(b,1) = (f, c)$ – marked.

For pair (f, b)
$(\delta(f, 0), \delta(b,0)) = (c, g)$ – marked

For pair (g, b)
$(\delta(g, 0), \delta(b, 0)) = (g, g)$ – unmarked
$(\delta(h, 1), \delta(b, 1)) = (e, c)$ – marked

For pair (h, b)
$(\delta(h, 0), \delta(b, 0)) = (g, g)$ – unmarked
$(\delta(h,1), \delta(b,1)) = (c, c)$ - unmarked.

For pair (e, d)
$(\delta(e, 0), \delta(d, 0)) = (h, c)$ – marked
(e, d) is distinguishable.

For pair (f, d)
(δ(f, 0), δ(d, 0)) = (c, c) – unmarked
(δ(f,1), δ(f,1)) = (g, g) - unmarked.

For pair (g, d)
(δ(g, 0), δ(d, 0)) = (g, c) – marked

For pair (h, d)
(δ(h, 0), δ(d, 0)) = (g, c) – marked

For pair (f, e)
(δ(f, 0), δ(e, 0)) = (c, h) – marked

For pair (g, e)
(δ(g, 0), δ(e, 0)) = (g, h) – unmarked
(δ(g,1), δ(e,1)) = (e, f) -marked.

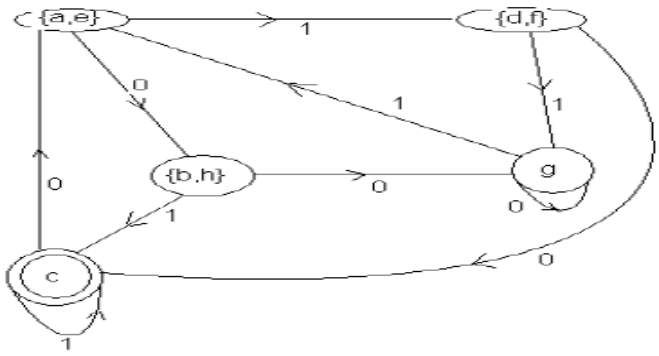
For pair (h, e)
(δ(h, 0), δ(e, 0)) = (g, h) – unmarked
(δ(h,1), δ(e,1)) = (c, f) -marked.

For pair (g, f)
(δ(g, 0), δ(f, 0)) = (g, c) – marked

For pair (h, f)
(δ(h, 0), δ(f, 0)) = (g, c) – marked

For pair (h, g)
(δ(h, 0), δ(g, 0)) = (g, g) – unmarked
(δ(h,1), δ(g,1)) = (c, e) -marked.
Thus (a, e), (b, h) and (d, f) are equivalent pairs of states.

Hence the minimized DFA is



Another simple approaches from Asesh Kumar Pandey's Book is