SCS1204 - Theory of Computation

UNIT I

FINITE AUTOMATA AND REGULAR LANGUAGES

Finite automata and regular languages – Regular languages and regular expressions – Finite automata – Union, intersections, and complements of automata – Non-determinism and Kleene's theorem – Non-deterministic finite automata and NFA with Λ transition – Pumping lemma for Regular Languages

FINITE AUTOMATA AND REGULAR EXPRESSIONS

DEFINITION

A Regular language over an alphabet ∑ is one can be obtained from very simplest languages over ∑ , thise containing a single string of langth 0 ot 1, using only the operations of union(U), concatenation (C) and kleen*.

REGULAR LANGUAGE AND REGULAR EXPRESSION(RE)

A language is reguler if there exits a DFA for that language. The language accepted by DFA is RL.

A Mathematical notation used for described the regular language. This is formed by using 3 Symbols:

(i) . [dot operator] – for concatenation

(ii) + [Union operator] –atleast 1 occurence

(iii) * [Closure Operator] – Zero or more occurences

DEFINITION FOR REGULAR EXPRESSION

Let ∑ be an alphabet the regular expression over ∑ and the sets that they denote are defined recursively as follows:

i. Ø is a RE and denotes the empty set.

ii. ε is a RE and denotes the set { ε }

iii. For each a in ∑, a is a RE and denotes the set {a}

iv. If r and s are RE that denoting the languages R and S respectively, then,

(r+s) is a RE that denotes the set (RUS)

(rs) is a RE that denotes the set RS

(r)* is a RE that denotes the set R*

FINITE AUTOMATA

Study of representation and recognition of languages identifying whether i/p is accepted either accept or recogonized

DEFINITION: A finite Automaton or finite state machine is a 5 tuple

$M=(Q, \sum, q_0, \partial, A)$, where

Q- is a finite set of states

∑ - finite set of input symbols

$q_0 \varepsilon Q$ – Initial states

$A \subseteq Q \rightarrow set\ of\ Accepting\ states.$

$\partial - Transition\ Fuction$

$\delta : Q \times \sum \rightarrow Q$

EXTENDED TRANSITION FUNCTION OF FINITE AUTOMATA

The machine M=(Q, ∑, q_0, ∂, A) be a FA, the transition function ∂Let Q be a finite set and let Σbe a finite set of symbols. Also let δbe a function from Q $\times$ $\Sigma$to Q , let q_0 be a state in Q and let A be a subset of Q. We call the elements of Q a **state**, $\delta$the **transition function**, q_0 the **initial state** and A the set of**accepting states**. Then a **deterministic finite automaton** is a 5-tuple < Q , Σ, q_0 , δ, A >

Notes on the definition

1. The set Q in the above definition is simply a set with a finite number of elements. Its elements can, however, be interpreted as a state that the system (automaton) is in. Thus in the example of vending machine, for example, the states of the machine such as "waiting for a customer to put a coin in", "have received 5 cents" etc. are the elements of Q. "Waiting for a customer to put a coin in" can be considered the initial state of this automaton and the state in which the machine gives out a soda can can be considered the accepting state.
2. The transition function is also called a **next state function** meaning that the automaton moves into the state δ(q, a) if it receives the input symbol a while in state q. Thus in the example of vending machine, if q is the initial state and a nickel is put in, then δ(q, a) is equal to "have received 5 cents".
3. Note that $\delta$is a function. Thus for **each state** q of Q and for **each symbol** a of Σ, δ(q, a) must be specified.
4. The accepting states are used to distinguish sequences of inputs given to the finite automaton. If the finite automaton is in an accepting state when the input ceases to come, the sequence of input symbols given to the finite automaton is "accepted". Otherwise it is not accepted. For example, in the Example 1 below, the string a is accepted by the finite automaton. But any other strings such as aa, aaa, etc. are not accepted.
5. A deterministic finite automaton is also called simply a "finite automaton". Abbreviations such as **FA** and **DFA** are used to denote deterministic finite automaton.

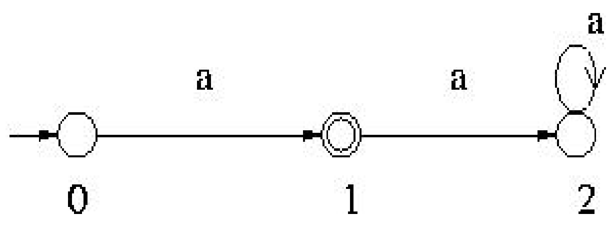
   DFAs are often represented by digraphs called **(state) transition diagram**. The vertices (denoted by single circles) of a transition diagram represent the states of the DFA and the arcs labeled with an input symbol correspond to the transitions. An arc (p , q) from vertex p to vertex q with label σ represents the transition δ(p, σ) = q . The accepting states are indicated by double circles. Transition functions can also be represented by tables as seen below. They are called **transition table**.

Examples of finite automaton

Example 1: Q = { 0, 1, 2 }, Σ = { a }, A = { 1 }, the initial state is 0 and δ is as shown in the following table.

State (q)	Input (a)	Next State (δ(q, a))
0	a	1
1	a	2
2	a	2

A state transition diagram for this DFA is given below.

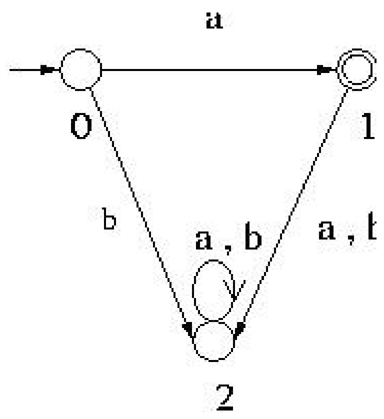


If the alphabet Σ of the Example 1 is changed to { a, b } in stead of { a }, then we need a DFA such as shown in the following examle to accept the same string a. It is a little more complex DFA.

Example 2: Q = { 0, 1, 2 }, Σ = { a, b }, A = { 1 }, the initial state is 0 and δ is as shown in the following table.

State (q)	Input (a)	Next State (δ(q, a))
0	a	1
0	b	2
1	a	2
1	b	2
2	a	2
2	b	2

Note that for each state there are two rows in the table for δ corresponding to the symbols a and b, while in the Example 1 there is only one row for each state.
A state transition diagram for this DFA is given below.

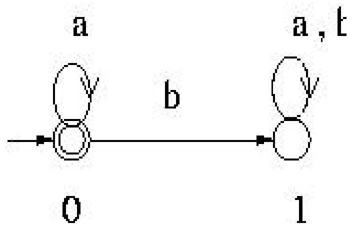


A DFA that accepts all strings consisting of only symbol a over the alphabet { a, b } is the next example.

Example 3: Q = { 0, 1 }, Σ = { a, b }, A = { 0 }, the initial state is 0 and δ is as shown in the following table.

State (q)	Input (a)	Next State (δ(q, a))
0	a	0
0	b	1
1	a	1
1	b	1

A state transition diagram for this DFA is given below.



UNION, INTERSECTION AND COMPLEMENTS OF AUTOMATA

THEOREM

Suppose that M1=(Q1, Σ,q1,A1, δ1) and M2=(Q2, Σ,q2,A2, δ2) accept languages L1 and L2 respectively.Let M be a finite automata defined as M=(Q, Σ,q,A, δ) where,
Q=Q1xQ2, q0=(q1,q2) and δ=((p,q),1)=(δ1(p,a), δ2(q,a)),for any p$\in$Q1,q$\in$q2 and a$\in$ Σ.,then

1.If A={{(p,q)|p$\in$A1 or q$\in$aA2},M aceepts L1$\cup$L2.}

2. If A={{(p,q)|p∈A1 or q∈aA2},M aceepts L1∩L2.}
If A={{(p)|p∈A1 or q∉A2},M aceepts L1-L2.}

DEFINITION OF NONDETERMINISTIC FINITE AUTOMATON

Let Q be a finite set and let Σ be a finite set of symbols. Also let δ be a function from Q $\times$ Σ to 2^Q ,let q_0 be a state in Q and let A be a subset of Q. We call the elements of Q a **state**, δ the **transition function**, q_0 the **initial state** and A the set of **accepting states**.
Then a **nondeterministic finite automaton** is a 5-tuple < Q , Σ, q_0 , δ, A >

Notes on the definition

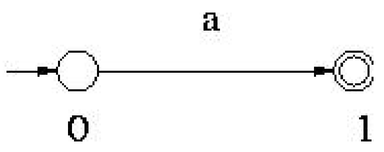
1. As in the case of DFA the set Q in the above definition is simply a set with a finite number of elements. Its elements can be interpreted as a state that the system (automaton) is in.
2. The transition function is also called a **next state function** . Unlike DFAs an NFA moves into one of the states given by δ(q, a) if it receives the input symbol a while in state q. Which one of the states in δ(q, a) to select is determined nondeterministically.
3. Note that δ is a function. Thus for **each state** q of Q and for **each symbol** a of Σ δ(q, a) must be specified. But it can be the empty set, in which case the NFA aborts its operation.
4. As in the case of DFA the accepting states are used to distinguish sequences of inputs given to the finite automaton. If the finite automaton is in an accepting state when the input ends i.e. ceases to come, the sequence of input symbols given to the finite automaton is "accepted". Otherwise it is not accepted.
5. Note that any DFA is also a NFA.

Examples of NFA

Example 1: Q = { 0, 1 }, Σ = { a }, A = { 1 }, the initial state is 0 and δ is as shown in the following table.

State (q)	Input (a)	Next State (δ(q, a))
0	a	{ 1 }
1	a	$\emptyset$

A state transition diagram for this finite automaton is given below.

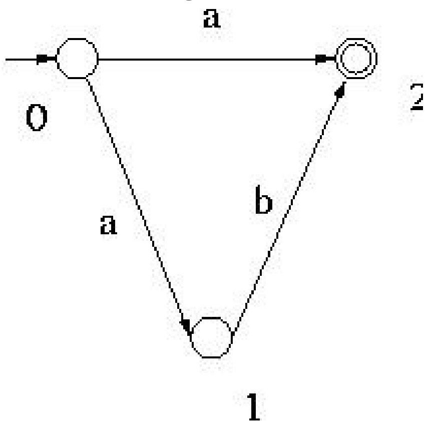


If the alphabet Σ is changed to { a, b } in stead of { a }, this is still an NFA that accepts { a } .

Example 2: Q = { 0, 1, 2 }, Σ = { a, b }, A = { 2 }, the initial state is 0 and δ is as shown in the following table.

State (q)	Input (a)	Next State (δ(q, a))
0	a	{ 1 , 2 }
0	b	$\emptyset$
1	a	$\emptyset$
1	b	{ 2 }
2	a	$\emptyset$
2	b	$\emptyset$

Note that for each state there are two rows in the table for δ corresponding to the symbols a and b, whilein the Example 1 there is only one row for each state.
A state transition diagram for this finite automaton is given below.



OPERATION OF NFA

Let us see how an automaton operates when some inputs are applied to it. As an example let us consider the automaton of Example 2 above.
Initially it is in state 0. When it reads the symbol a, it moves to either state 1 or state 2. Since the state 2 is the accepting state, if it moves to state 2 and no more inputs are given, then it stays in the accepting state. We say that this automaton accepts the string a. If on the other hand it moves to state 1 after reading a, if the next input is b and if no more inputs are given, then it goes to state 2 and remains there. Thus the string ab is also accepted by this NFA. If any other strings are given to this NFA, it does not accept any of them.

EQUIVALENCE OF DFA AND NFA

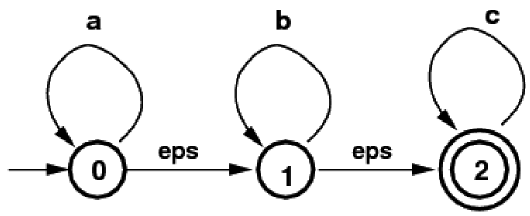
As every DFA is an NFA, the class of languages accepted by NFA's includes the class languages accepted by DFA's. For every NFA we can construct an equivalent DFA.

Theorem: Let L be a language accepted by a NFA. Then there exists a DFA that accepts L.

Proof: Let M=(Q,∑, δ,q_o, F) be an NFA accepting L. Define a DFA M_1=(Q_1, $∑_1$, $δ_1$,q_{o1}, F_1) as follows. The states of M^1 are all the possible non-empty subsets of the set of states for M. If M has n states, then M^1 has 2n-1 states which permit it to simulate all the states in which M could be at any particular point in time. The states symbols in Q^1 will be denoted by [q_o][q_1].....[q_{n-1}],[q_0,q_1],[q_0,q_2]......[q_0,q_{n-1}], [q_1,q_2],.......[q_1,q_{n-1}]...[q_{n-2},q_{n-1}] .....[q_0, q1.....q_{n-1}]. Also $δ^1$(q_o^1, x) ε F^1 exactly when δ (q_o, x)Ω≠ Ø. Both machines have the same alphabet and q_o^1=[q_0].
Define $∂^1$ as follows $δ^1$([q_1, q_2.....,q_i], a)=[p_1,p_2,...p_j]
If and only if δ ([q_1, q_2.....,q_i], a)=[p_1,p_2,...p_j]
That is, if we apply ∂ to each of q_1, q_2.....,q_i and taking the union, we get some new set of states p_1,p_2,...,p_j. This new set of states has representative [p_1,p_2,...p_j] ε Q^1. By construction it is clear that M^1 is deterministic. We must now prove that T(M^1)=T(M). Let us take an input string x that $δ^1$(q_o^1, x)=[p_1,p_2,..,p_j] **=**δ (q_o, x)= [p_1,p_2,...p_j].
Basic Step: If w is of length zero, i.e w in an empty stringe ε $δ^1$([q_o], ε =[q_o]) and δ (q_o, w) iff δ (q_o, ε)= q_o
Induction Step: Suppose that the result is true for inputs of length n or less. Now we prove the results fpr string of length n+1. Let w=xa be a string with x having length n and a € ∑. Then by induction assumption.
$δ^1$(q_o^1, x)= $δ^1$($δ^1$(q_o^1, x), a) = $δ^1$([p_1,p_2,..p_j], a)= [r_1,r_2,...r_k]
If and only if δ ([p_1,p_2,..p_j], a)= [r_1,r_2,...r_k]
Thus, $δ^1$(q_o^1, xa)= [r_1,r_2,...r_k] If and only if δ (q_o, xa)= [r_1,r_2,...r_k]
Hence the result is true for a string of length n+1. By induction hypothesis the result is true for all n. Thus M and M^1both accept the same string w iff ($δ^1$(q_o^1, x) or δ (q_o, x) respectively contains a state in F^1. Therefore T(M)=T(M^1).
Note: As DFA and NFA accept the same sets, we shall not differentiate between them.

NFA'S WITH E MOVES

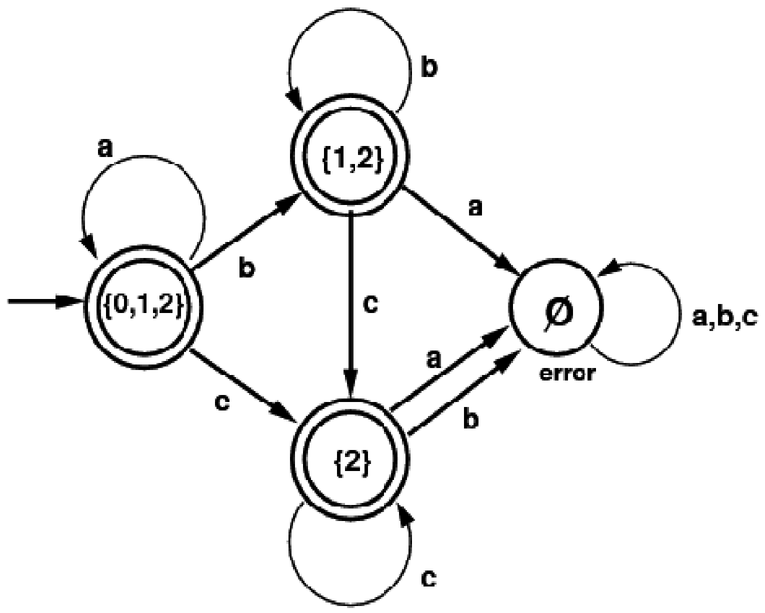
Here one allows transitions labeled with **ε**, the empty string. This means that you can follow such an arrow without consuming any input symbols. The **NFA** on the left shows how useful epsilon moves are in recognizing regular expressions with the example **a*b*c***: which is "zero of more **a**s, followed by zero or more **b**s, followed by zero of more **c**s". (The diagram uses "eps" for **ε**.)
On the right is a **DFA** that recognizes the same language. Notice some new things about this **DFA**: three states are terminal ones, including the start state. There is an "error" state, with label **err**, and it is not terminal. If you transition to that state, you can't get out and you can't accept. This state could also have been labeled with the symbol for the empty set: **Ø**
A refinement or expansion of the Subset Algorithm shows how to construct this **DFA.**



NFA for a*b*c*

Subset Algorithm (Tables)

DFA, a*b*c*			
States	**a**	**b**	**c**
{0,1,2} (start,term)	**{0,1,2}**	**{1,2}**	**{2}**
{1,2} (term)	**Ø**	**{1,2}**	**{2}**
{2} (term)	**Ø**	**Ø**	**{2}**
Ø	**Ø**	**Ø**	**Ø**



DFA for a*b*c*

ε-closure: The key to constructing a **DFA** from an **NFA** with **ε**-moves is the **ε-closure** algorithm. The starts with a set of states and find all additional states that can be reached from the given states using **ε**-moves.

```
Algorithm:                                         ε-closure
Input: an NFA with ε-moves N, a set P of states.
Output: ε-closure(P), which is P along with all states
  reachable from P by a sequence of ε-moves.
Method: use stack S to carry out a depth-first search

// black means in ε-closure(P)
color all states white
color all states of P black
push all states of P onto S
while (S not empty) {
  u = pop(S)
  for (each edge (u,v) labeled ε) {
    if (v has color white) {
```

The subset algorithm for an **NFA** with **ε**-moves is the same as the one for ordinary **NFA**s, except that each time a subset is constructed, it must have the **ε**-closure algorithm (at the left) applied to it.

In the above case, the start state of the **NFA** is **0**, so the start state of the **DFA** is not **{0}** but **ε-closure({0}) = {0,1,2}.**

Transitions on **b** from **0, 1,** or **2** only go to **1**. Then **ε-closure({1}) = {1,2}.**.

There are no transitions on **a** from **1** or **2**, so the set in this case is **Ø**, the empty set. The empty set is a subset that serves as an err

```
        color **v black**
        **push v**
      }
   }
}
**ε-closure(P) = black** states
```

EQUIVALENCE OF NFA's WITH AND WITHOUT ε-MOVES

Theorem

If L is accepted by a NFA with έ-transitions, then L is accepted by an NFA without ε-transitions**.**

Proof: Let M=(Q, ∑,δ, q_o, F) be an NFA with έ-transitions. Let M=(Q^1, $\sum^1$, δ^1,q_o^1, F^1) where

$$F^1 = \begin{cases} F \cup \{q_0\} & \text{if } \varepsilon\text{-CLOSURE } (q_0) \text{ contains a state in } F \\ F & \text{otherwise.} \end{cases}$$

$$\delta^1(q, a) = \hat{\delta}(q, a),\ q \in \alpha,\ a \in \Sigma.$$

Then M^1 has no ε-transitions. Note that δ^1 and are δ same but δ and δ are not same.

We show that induction that $\delta^1\{q_0,w\}= \delta(q_0,w)$ for any string w. note that for w= ε, $\delta^1(q_0, \varepsilon)=\{q_0\}$ and $\delta\{q_0, \varepsilon\}=$ ε-CLOSURE$\{q_0\}$. So $\delta(q_0, \varepsilon)\neq \delta\{q_0, \varepsilon\}$. So the induction is not true for the string having length 0. We start our induction for the string having length 1.

Basic step: Let a be the string of length one. That is , a Є∑. $\delta^1\{q_0, a\}= \delta(q_0, a)$ by definition of $\delta^{1.}$

Induction: Assume that $\delta^1(q_0, x)= \delta(q_0, x)$ for any string x of length ≤n. Assume w is a string of length n+1. Let w=xa for some symbol a Є∑. Then

$$\begin{aligned} \delta^1(q_0, w) &= \delta^1 (q_0, xa) \\ &= \delta^1 (\delta^1(q_0, x),a) \\ &= \delta^1 (\delta^1(q_0, x),a) \\ &= \delta^1 (p, a) \end{aligned}$$

Where p= $\delta^1(q_0, x)$

We must show that $\delta^1(p, a)= \delta(q_0, xa)$ but

$$\delta^1(P, a) = \bigcup_{q \in P} \delta'(q,a) = \bigcup_{q \in P} \hat{\delta}(q,a)$$

$$\begin{aligned} &= \delta^1 (\delta^1(q_0, x),a) \\ &= \delta^1(q_0, xa) \\ &= \delta^1(q_0, w) \end{aligned}$$

Hence $\delta^1(q_0, w)= \delta^1(q_0, w)$

Lastly, we show that $\delta^1(q_0, w)$ contains a state of F^1 if $\delta^1(q_0, w)$ and only if contains a state of F. Let $\delta^1(q_0, w)$ contains a state of F^1. If w= ε, then

$\delta^1(q_0, \varepsilon)= \{q_0\}= \delta^1(q_0, \varepsilon) =$ ε-CLOSURE$\{q_0\}$

Note that ε-CLOSURE$\{q_0\}$ contains a state (possibly q_0) in F. Hence $\delta^1(q_0, \varepsilon)$ contains a state in F. Converse is clear from the definition of F^1.

If $w \neq \varepsilon$, then $w=xa$ for some symbol a. if $\delta^1(q_0, w)$ contains a state of F,then surely $\delta^1(q_0, w)$ contains the same state in F^1. Conversely if $\delta^1(q_0, w)$ contains the same state in F^1 other than q_0 then $\delta^1(q_0, w)$ contains a state in F. If $\delta^1(q_0, w)$ contains q_0, $q_0 \in F$. Then ε-CLOSURE $\delta^1(\delta^1(q_0, x),a)$ the state in ε-CLOSURE(q_0) and in F must be in.

Conversion of regular expression to FSA)

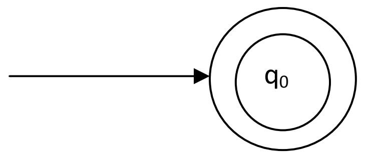
Theorem 1 (Part 1 of Kleene's theorem): Any regular language is accepted by a finite automaton.

For every regular expression r there exists a NFA with ε-transitions that accepts L(r).

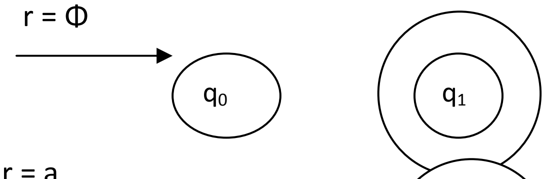
Proof: We prove by induction on the number of operators in the regular expression r that there is an NFA M with ε-transitions, having one final state and no transitions out of this final state such that T(M)=L(r).

Basis Step: Suppose r is ε, Φ or a for some a£∑. Then the following NFA's clearly satisfy the conditions.

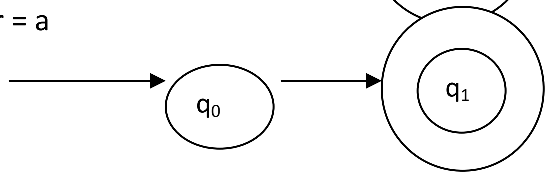
r = ε



r = Φ



r = a



Induction Step: Assume the theorem is true for r having fewer than i operators i ≥ 1. Let have i operators. We discuss three cases depending on the form of r.

Case 1: Let $r=r_1+r_2$. Both $r=r_1$ and r_2 must have fewer than i operators. Thus there are NFA's $M_1=(Q_1, \sum_1, \delta_1, q_1, \{F_1\})$ and $M_2=(Q_2, \sum_2, \delta_2, q_2, \{F_2\})$ with $T(M_1)=L(r_1)$ and $T(M_2)=L(r_2)$. Assume Q_1 and Q_2 are disjoint. Let q_0 be a new initial state and f_0 a new final state construct.

$M=(Q_1 \cup Q_2 \cup \{ q_0, f_0\}, \sum_1 \cup \sum_2, \delta, q_0 ,\{ f_0\})$

Where δ is defined by

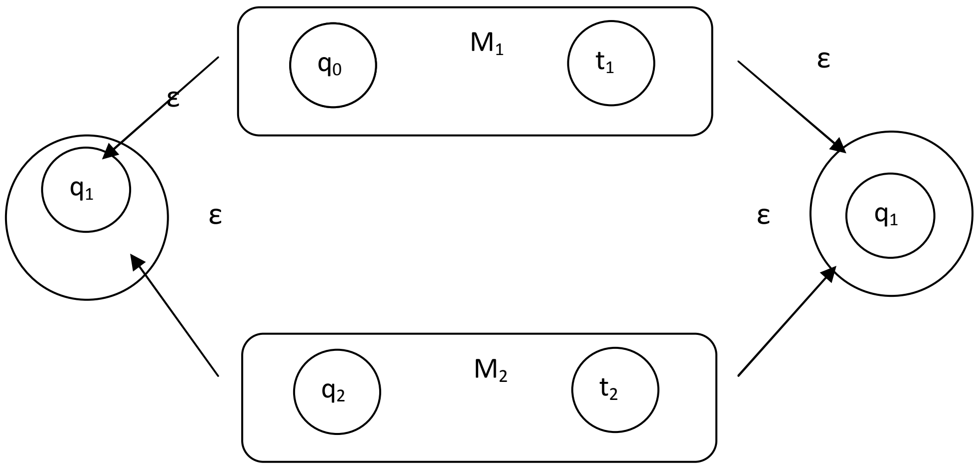
(i) $\delta\{ q_0, \varepsilon\}=\{q_1,q_2\}$

(ii) $\delta\{ q, a\}= \delta_1\{ q, a\}$ if $q \in Q_1-\{f_1\}$, $a \in \sum_1 \cup \{ \varepsilon\}$

$= \delta_2\{ q, a\}$ if $q \in Q_2-\{f_2\}$, $a \in \sum_2 \cup \{ \varepsilon\}$

(iii) $\delta_1(f_1, \varepsilon) = \delta_2(f_2, \varepsilon)=\{ f_0\}$

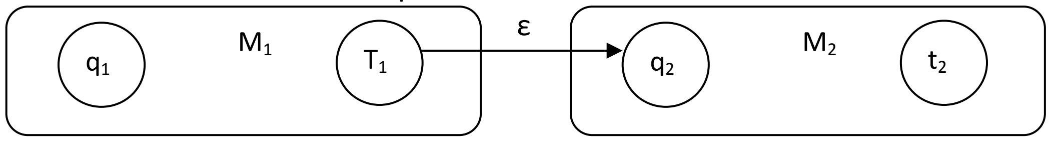
All the moves of M_1 and M_2 are present in M. The construction of M is depicted below



Any path in the transition diagram of M from q_0 to f_0 must be going to either q_1
Or q_2 on ε and follow any path in M_1 or in M_2 corresponding by and reaches f along the path ε.
These are the only paths from q_0 to f_0. There is a path labeled x in M from q_0 to f_0 if and only if there is a path labeled x in M_1 from q_1 to f_1 or a path in M_2 from q_2 to f_2 . Hence T(M)=T(M_1)UT(M_2).

Case2:r=r_1 r_2.Let M_1 and M_2 be as in case 1. Construct M=(Q_1 U Q_2 , $\sum_1$ U $\sum_2$, δ, {q_1} ,{ f_2})
Where δ is given by

(i) δ{ q, a}= δ_1{ q, a} if q ∈ Q_1 –{f_1} , a ∈ $\sum_1$ U { ε}
(ii) δ_1(f_1, ε) = {q_2}
(iii) δ{ q, a}= δ_2{ q, a} if q ∈ Q2 , a ∈ $\sum_2$ U { ε}
The construction of M is depicted as below

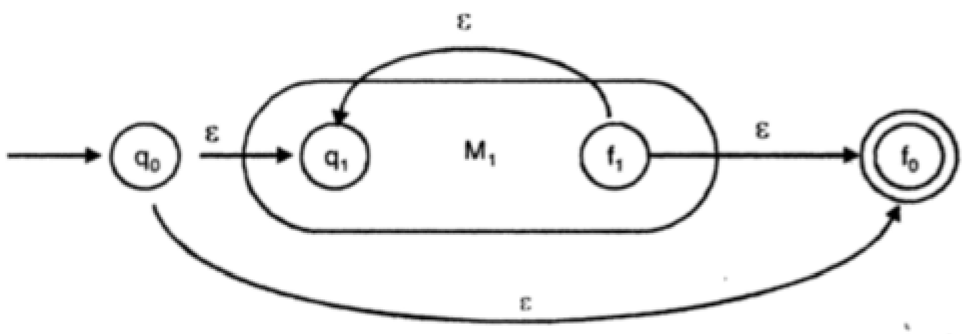


Every path in M from q_1 to f_2 is a path labeled by some string x from q_1 to f_1 followed by the edge from f_1 to q_2 labeled ε ,followed by a path labeled by some string y from q_2 to f_2. Thus
T(M)={xy :x∈ T(M_1), y∈ T(M_2)}
And hence
T(M)=T(M_1)T(M_2)
Case 3: r=(r_1) Let M_1=(Q_1, $\sum_1$, δ_1,q_1, {F_1}) and T(M_1)=r_1.onstruct
M=(Q_1 U { q_0, f_0},∑, δ,q_0, {f_0})
Where δ is given by

(i) δ{ q_0, ε }= δ(f_1, ε)={ q_1,f_0}

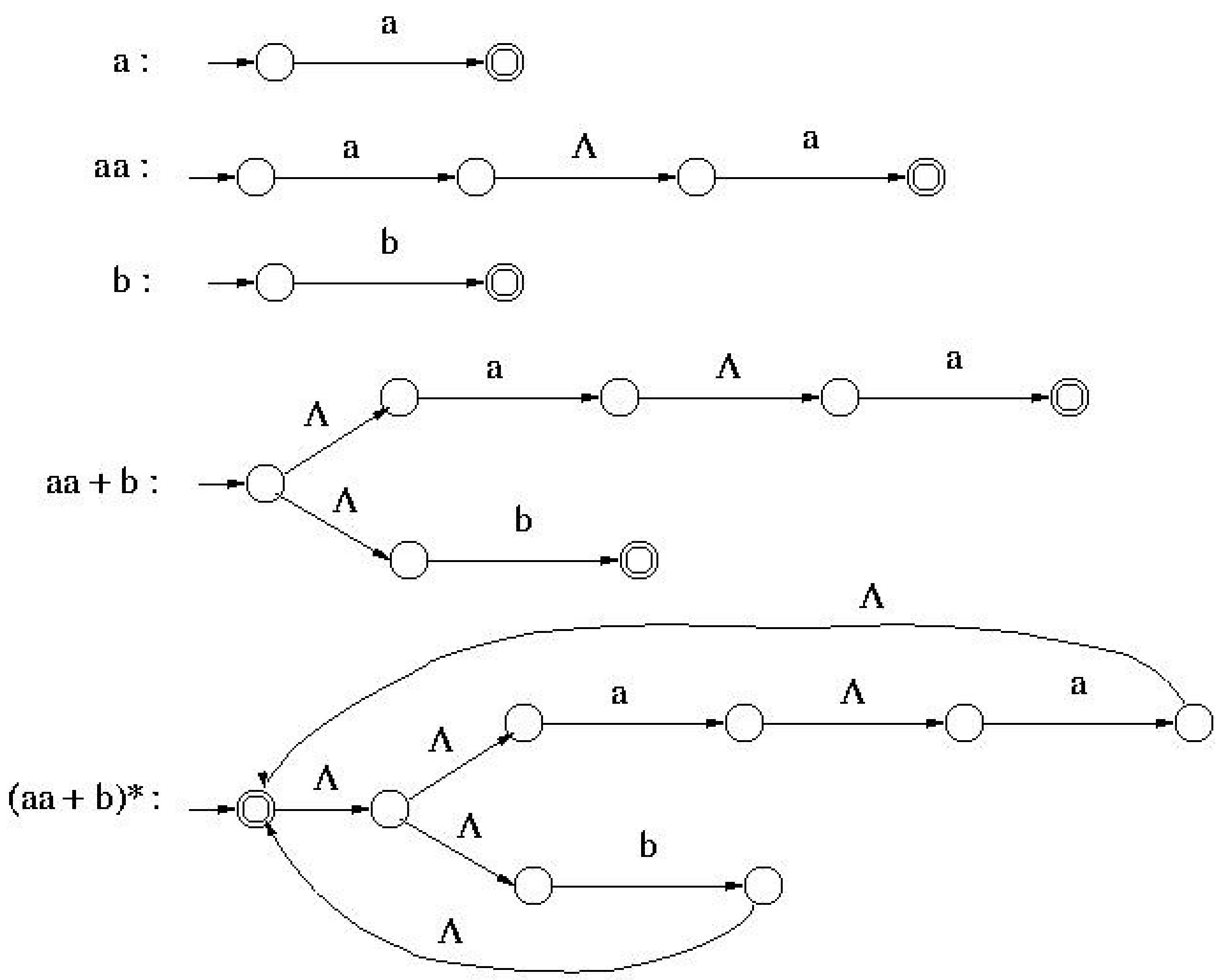
(ii) δ{ q, a}= δ_1{ q, a} for q ∈ Q1-{f_1} , a ∈ $\sum_1$ U { ε}

The construction of M is depicted below



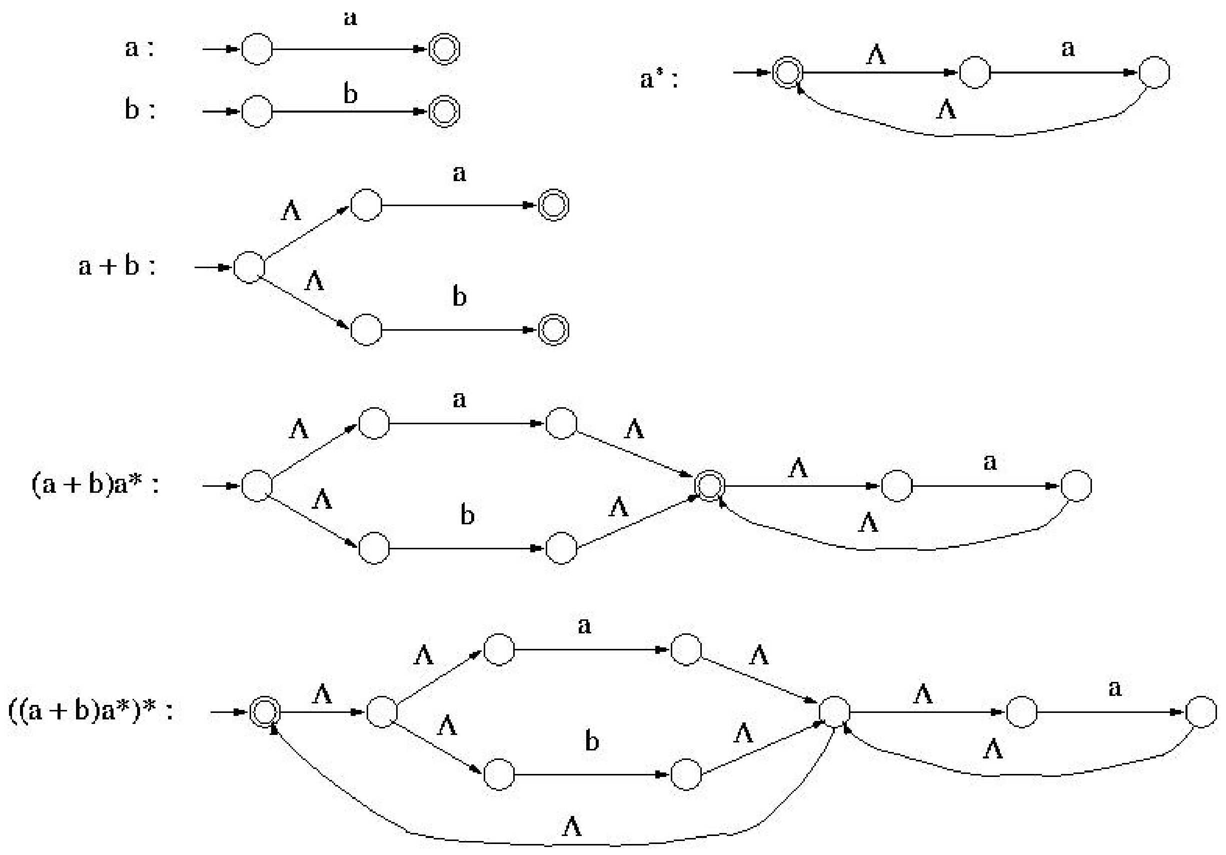
Any path from q_0 to f_0 consists either of a path from q_0 to f_0 on ε or a path from q_0 to q_1 on ε followed by some number of paths from q_1 to f_1, then back to q_1 on ε. There is a path in M from q_0 to f_0 labeled x if and only if we write $x=x_1x_2....x_j$ for some $j\geq 0$ such that each $x_i \in T(M_1)$. Hence $T(M)=T(M_1)$

Example 1: An NFA- Λ that accepts the language represented by the regular expression $(aa + b)^*$ can be constructed as follows using the operations given above.



Construction of $(aa + b)^*$

Example 2: An NFA- Λ that accepts the language represented by the regular expression $((a + b)a^*)^*$ can be constructed as follows using the operations given above.



Construction of ((a + b)a*)*

REGULAR EXPRESSIONS FROM DFA

Theorem 2 (Part 2 of Kleene's Theorem): Any language accepted by a finite automaton is regular. Theorem

If L is accepted by a DFA then L is denoted by a regular expression .

Proof: Let L be the set accepted by the DFA M=(Q,∑, δ,q_1,F), where Q={q_1,q_2,.....q_n}. Let R_{ij}^{k} be the set of strings tha takes M from state q_i to state q_j without entering and leaving through any state numbered higher than k. That is if x∈R_{ij}^{k} then

δ($q_{i,}$x) = q_j and if δ($q_{j,}$y) = q_l

for any y which is of the form x=y x_i....x_n than l≤k (or) y = ε and l=i (or) x=y and j=L

Note that R_{ij}^{n} denoted all strings that take q_i to q_j as there is no state numbered greater than n.

We can define R_{ij}^{k} recursively as follows. When the input string is given, M moves from q_i to q_j without passing through a state higher than q_k are either

(i) In R_{ij}^{k-1} (that is they never pass through a state as high as q_k

(ii) composed of a string in R_{ij}^{k-1} (which takes M to q_k first time) followed by zero or more strings in R_{ij}^{k-1} (which make M from q_k back to q_k) without passing through q_k or a higher numbered state)followed by a string in R_{ij}^{k-1}(which takes M from state q_k back to q_j) .That is

$$R_{ij}^{k-1} = R_{ij}^{k} \cup R_{ik2}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1}$$

We showfor each i,j,k there exists a regular expression R_{ij}^{k} denoting the language R_{ij}^{k} . We proceed by induction on K.

Basis step: When k=0, R_{ij}^{0} = {a} or ε. Thus R_{ij}^{0} can be written as $a_1+a_2..+a_p$ if i≠ j or $a_1+a_2..+a_p+ \varepsilon$ if i=j where { $a_1+a_2..+a_p$} is the set of symbols 'a' such that $\delta(q_{i,}a) = q_{j.}$if there are no such 'a' then ø or (ε in the case i=j) is R_{ij}^{0}.

Induction step: Assume by induction that each l and m, there exists a regulat expression R_{ij}^{k-1} such that $L(F_{lm}^{k-1}= R_{ij}^{k-1})$Hence therev exists a regular expression R_{ij}^{k} such that

$$r_{ij}^{k}= r_{ik}^{k-1} (r_{kk}^{k-1})^* r_{kj}^{k-1}+ r_{ij}^{k-1}$$

This completes yhe induction as the recursive formula for R_{ij}^{k} involves only the regular expression operators union, concatenation and closure.

To complete the proof we observe that

$$T(M)=\cup_{q_i \in F} R_{ij}^{n}$$

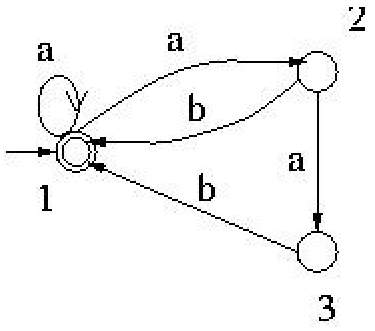
Since R_{ij}^{n} denotes the set of all strings that take q_1 to q_j .T(M) is denoted by the regular expression

$$r_{ij}^{n}{}_1 + r_{ij}^{n}{}_{2} + r_{ij}^{n}{}_p$$

where F={q_{j1}, q_{j2},..., q_{jp}}

Hence the proof.

Example : Let us find the language accepted by the following finite automaton using the lemmas.



NFA

Let us denote by r(p, q, k) the regular expression for the set of strings L(p, q, k).

Then the language accepted by this NFA is r(1, 1, 3).

By Lemma 1, $r(1, 1, 3) = r(1, 1, 2) + r(1, 3, 2)r(3, 3, 2)^*r(3, 1, 2)$.

r(1, 1, 2):

$r(1, 1, 2) = r(1, 1, 1) + r(1, 2, 1)r(2, 2, 1)^*r(2, 1, 1)$

$r(1, 1, 1) = r(1,1,0) + r(1,1,0)r(1,1,0)^*r(1,1,0) = a^*$, since $r(1,1,0) = a + \Lambda$.

$r(1, 2, 1) = r(1,2,0) + r(1,1,0)r(1,1,0)^*r(1,2,0) = a^+$, since $r(1,2,0) = a$.

$r(2, 2, 1) = r(2,2,0) + r(2,1,0)r(1,1,0)^*r(1,2,0) = ba^+ + \Lambda$, since $r(2,2,0) = \Lambda$ and $r(2,1,0) = b$.

$r(2, 1, 1) = r(2,1,0) + r(2,1,0)r(1,1,0)^*r(1,1,0) = ba^*$.

Hence $r(1, 1, 2) = a^{*} + a^{+}(b\,a^{+})^{*}b\,a^{*}$.

r(1, 3, 2):

$r(1, 3, 2) = r(1, 3, 1) + r(1, 2, 1)r(2, 2, 1)^{*}r(2, 3, 1)$

$r(1, 3, 1) = \emptyset$

$r(2, 3, 1) = a$

Hence $r(1, 3, 2) = a^{+}(b\,a^{+} + \Lambda)^{*}a = a^{+}(b\,a^{+})^{*}a$.

r(3, 3, 2):

$r(3, 3, 2) = r(3, 3, 1) + r(3, 2, 1)r(2, 2, 1)^{*}r(2, 3, 1)$

$r(3, 3, 1) = \Lambda$

$r(3, 2, 1) = r(3,2,0) + r(3,1,0)r(1,1,0)^{*}r(1,2,0) = ba^{+}$, since $r(3,2,0) = \emptyset$ and $r(3,1,0) = b$.

Hence $r(3, 3, 2) = \Lambda + ba^{+}(ba^{+} + \Lambda)^{*}a = \Lambda + (ba^{+})^{+}a$

r(3, 1, 2):

$r(3, 1, 2) = r(3, 1, 1) + r(3, 2, 1)r(2, 2, 1)^{*}r(2, 1, 1)$

r(3, 1, 1) = r(3,1,0) + r(3,1,0)r(1,1,0)<sup*< sup="">r(1,1,0) = ba*

Hence $r(3, 1, 2) = ba^{*} + ba^{+}(ba^{+} + \Lambda)^{*}ba^{*} = (ba^{+})^{*}ba^{*}$.

Hence $r(1, 1, 3) = a^{*} + a^{+}(b\,a^{+})^{*}ba^{*} + (a^{+}(ba^{+})^{*}a)(\Lambda + (ba^{+})^{+}a)^{*}(ba^{+})^{*}ba^{*}$.

This can be further simplified to $(a + ab + abb)^{*}$, then to $(a + ab)^{*}$.

In this example there is only one accepting state. If there are more accepting states, then r(p, q, n) must be found for each accepting state q, where p is the initial state and n is the number of states in the given finite automaton, and all the r(p, q, n)'s must be added together to get the regular expression for the language accepted by the automaton.

LIMITATIONS OF FINITE AUTOMATA

1.A finite automata has finite number of stsates and cannot remember long information.

2.It has trouble recognizing various types of languages involving counting,calculating,storing the string

PUMPING LEMMA FOR REGULAR LANGUAGES

- Used to prove that a language is not regular.
- Based on pigeon hole principle.

ADVANTAGES OF PUMPING LEMMA

- Can easily prove that certain languages are regular and finite or infinite.

Lemma : The pumping lemma states that for a regular language L, there exists a constant n such that every string w in L (such that length of w (|w|) >= n) we can break w into three substrings, w = xyz, such that

- y is not null
- |xy| <= n
- For all i >= 0, the string xy^iz is also in L

Note that |x| and |z| can be 0 but |y| has to be greater than 1.

The lemma states that for a regular language every string can be broken into three parts x,y and z such that if we repeat y i times between x and z then the resulting string is also present in L.

Example : Prove that the language of palindromes over {0, 1} is not regular.

Solution:
Let L be a regular language and n be the integer in the statement of the pumping lemma.

To prove
The language is not regular

Proof
choose a string w of length n.
Let w =O^n 1 O^n.
Let y = O^j where j > 0.
By pumping lemma $O^{(n-j)}$ $(O^j)^i$1 O^n ∈L for all values of i >= 0.
But for i = 0,w= $O^{(n-j)}$1 O^n ∉ L .This contradicts the pumping lemma for L
Hence L is not a regular language.