Chapter 2 :Finite Automata

2.1 Finite Automata

Automata are computational devices to solve language recognition problems. Language recognition problem is to determine whether a word belongs to a language.
Automaton = abstract computing device, or "machine"
An automaton is an abstract model of a digital computer. Finite automata are good models of computers with a extremely limited amount of memory. Example of finite automata is Elevator problem, control unit of computer etc.

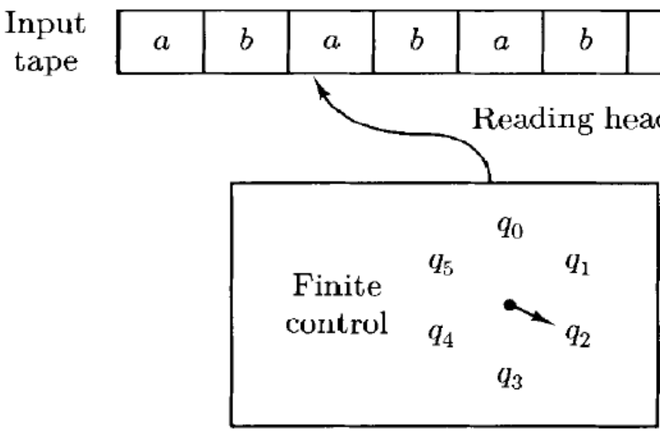


Fig: Finite automata

An automaton has a mechanism to read input ,which is string over a given alphabet. This input is actually written on an *input tape /file* **,**which can be read by automaton but cannot change it. Input file is divided into cells each of which can hold one symbol. Automaton has a control unit which is said to be in one of finite number of internal states. The automation can change states in a defined way.

- A state is the condition with respect to structure ,form, constitution and phase.
- Finite automaton is the simplest acceptor or rejecter for language specification

Uses of finite automata

- *Used in software for verifying all kinds of systems with a finite number of states, such as communication protocols*
- *Used in software for scanning text, to find certain patterns*
- *Used in "Lexical analyzers" of compilers (to turn program text into "tokens", e.g. identifiers, keywords, brackets, punctuation)*
- *Part of Turing machines and abacus machines*

Types of Finite Automaton

1. Deterministic Finite automata (DFA)
2. Non -deterministic finite automata (NDFA/NFA)

Deterministic Finite Automata(DFA)

Deterministic automaton is one in which each move (transition from one state to another) is uniquely determined by the current configuration. If the internal states , input and contents of the storage are known it is possible to predict the next (future) behavior of the automaton . This is said to be deterministic automaton otherwise it is non - deterministic automaton.

Formal Definition
A deterministic finite automaton is a quintuple M = (K,∑, δ, s, F) where
K is a finite set of states,
∑ is an alphabet,
s є K is the initial state,
F ⊆ K is the set of final states, and
δ, the transition function, is a function from K x ∑ → K.

If M is in state q є K and the symbol read from the input tape is a є ∑, then δ(q, a) є K is the uniquely determined state to which K passes.

The transition from one internal state to another are governed by transition function δ.
If δ (q_0, a) →q_1 then if the DFA is in state q_0 and the current input symbol is 'a' the DFA will go into state q_1 .

Configuration of DFA

A configuration is determined by the current state and the unread part of the string being processed.
Let M = (K, ∑, δ, s, F) be a DFA ,we say that a word in K x ∑* is a configuration of M. It represents the current states of M and remaining unread input of M. e.g (q_2, ababab) is one configuration

a	b	a	b	a	b

current scanning symbol

The binary relation $\vdash_M$ holds between two configurations of M if and only if the machine can pass from one to the other as a result of a single move. Thus if (q, w) and (q', w') are two configurations of M, then (q, w) $\vdash_M$ (q', w') if and only if w = aw' for some symbol a є ∑, and δ(q, a) = q'. For every configuration except those of the from (q, e) there is a uniquely determined next configuration.
A configuration of the form (q, e) signifies that M has consumed all its input, and hence its operation ceases at this point.

We denote the reflexive, transitive closure of $\vdash_M$ by $\vdash_M^*$; $(q, w) \vdash_M^* (q', w')$ is read, (q, w) yields (q', w') (after some number, possibly zero, of steps).

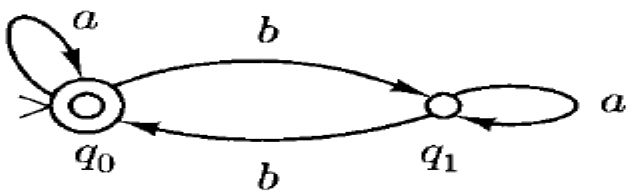
A string w є ∑ is said to be accepted by M if and only if there is a state q є F such that (s, w)* $\vdash_M^*$ *(q, e). Finally, the language accepted by M, L(M), is the set of all strings accepted by M.*

Example 1
*Design a DFA that accepts the language L which has set of strings in {a, b } * that have even number of b's .*

Solution
Let required DFA M= (K, ∑, δ, s, F)
where
K= {q_0, q_1 }
∑= { a ,b}
s=q_0
F= { q_0 } and δ is the function tabulated below

δ/∑	a	b
→q_0	q_0	q_1
q_1	q_1	q_0

State transition diagram is given as



e.g : If M is given the input *aabba*, its initial configuration is (qo, aabba). Then

$$\begin{aligned}(q_0, aabba) &\vdash_M (q_0, abba)\\ &\vdash_M (q_0, bba)\\ &\vdash_M (q_1, ba)\\ &\vdash_M (q_0, a)\\ &\vdash_M (q_0, e)\end{aligned}$$

Therefore $(q_0, aabba) \vdash_M^* (q_0, e)$, and so $aabba$ is accepted by M

Example 2:

*Design a deterministic finite automaton M that accepts the language L(M) = {w є {a, b} * : w does not contain three consecutive b's}.*

Solution

Let required DFA M= (K,∑, δ, s, F)

where

K= {q_0, q_1 , q_2, q_3 }

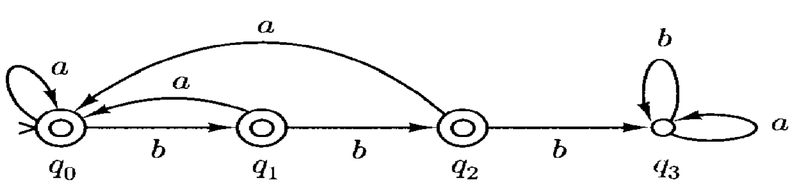
∑= { a ,b}

s=q_0

F= { q_0, q_1, q_2 } and δ is the function tabulated below

δ/∑	a	b
→q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_3
q_3	q_3	q_3

State diagram is given as

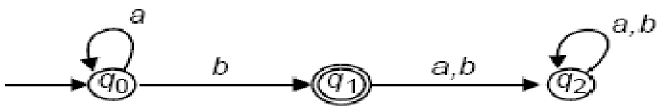


State q_3 is said to be a dead state, and if M reaches state q_3 it is said to be trapped, since no further input can enable it to escape from this state.

Example 3:
Design a DFA, the language recognized by the Automaton being $L = \{a^n b: n \geq 0\}$
Solution
Let required DFA $M= (K,\sum, \delta, s, F)$
where
$K= \{q_0, q_1, q_2\}$
$\sum= \{a, b\}$
$s=q_0$
$F= \{q_1\}$ and δ is the function tabulated below

δ/∑	**a**	**b**
$\rightarrow q_0$	q_0	q_1
$*q_1$	q_2	q_2
q_2	q_2	q_2

State diagram is given as

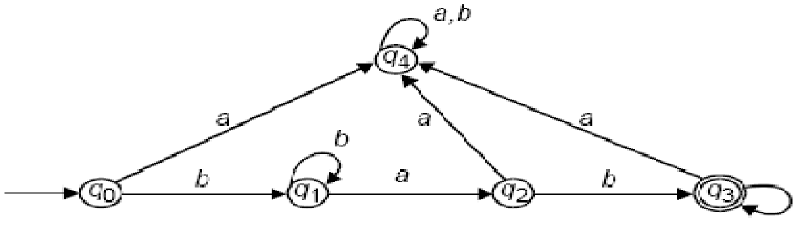


Example 4
Given $\sum = \{a, b\}$, construct a DFA that shall recognize the language $L = \{b^m ab^n : m, n > 0\}$.

Solution
Let required DFA $M= (K,\sum, \delta, s, F)$
where
$K= \{q_0, q_1, q_2, q_3, q_4\}$
$\sum= \{a, b\}$
$s=q_0$
$F= \{q_3\}$ and δ is the function tabulated below

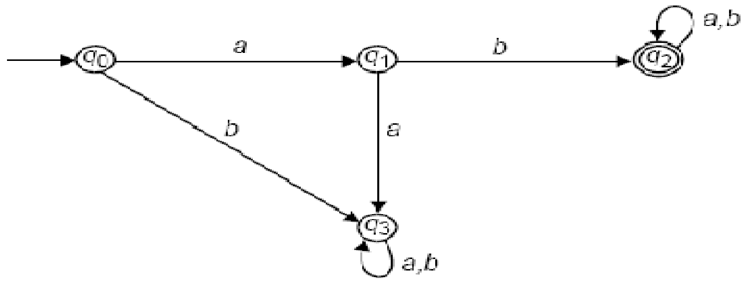
δ/∑	**a**	**b**
$\rightarrow q_0$	q_4	q_1
q_1	q_2	q_1
q_2	q_4	q_3
$*q_3$	q_4	q_3
q_4	q_4	q_4

State diagram is given as



Example 5:
Construct a DFA which recognizes the set of all strings on ∑ = {a, b} starting with the prefix ‘ab’.

Solution



NONDETERMINISTIC FINITE AUTOMATA (NFA)

- NFA has Ability to change states in a way that is only partially determined by the current state and input symbol.
- Permit several possible "next states" for a given combination of current state and input symbol.
- The automaton, as it reads the input string, may choose at each step to go into anyone of these legal next states; the choice is not determined by anything in our model, and is therefore said to be nondeterministic.
- Nondeterministic devices are not meant as realistic models of computers. They are simply a useful notational generalization of finite automata, as they can greatly simplify the description of these automata.
- Nondeterministic finite automaton can be a much more convenient device to design than a deterministic finite automaton,

Formal Definition : A nondeterministic finite automaton is a quintuple M = (K, ∑, Δ, s, F), where
K is a finite set of states,
∑ is an alphabet,
s ϵ K is the initial state,
F ⊆ K is the set of final states, and
Δ, the transition relation, is a subset of K x (∑ ∪ {e}) x K.

NFA is a variant of finite automaton with two capabilities

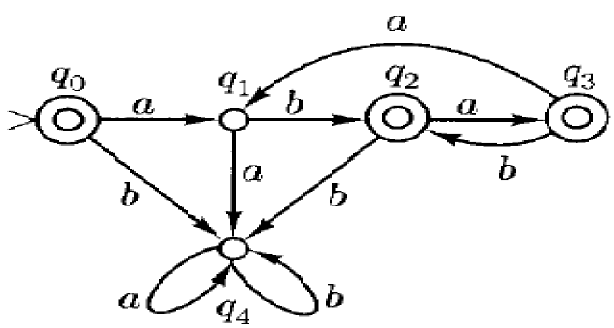
a. e or λ transition : state transition can made without reading a symbol
b. non-determinism : zero or more than one possible value may exist for state transition

For each p ϵ K and each a ϵ ∑{ e} , δ (s, a) =R means
"Upon reading an 'a ' the automaton M may transition from state s to any state in R."
For e in particular, δ (s, e) =R means,
"Without reading the symbol the automaton M may transition from s to any state in R"
Configuration of NFA:

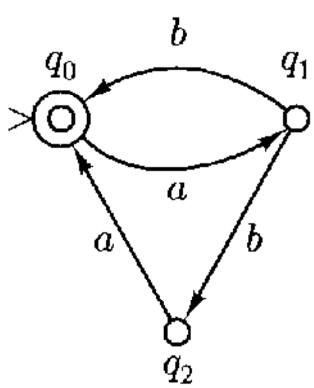
The formal definitions of computations by nondeterministic finite automata are very similar to those for deterministic finite automata. A **configuration** of M is, once again, an element of $K \times \Sigma^*$. The relation $\vdash_M$ between configurations (**yields in one step**) is defined as follows: $(q, w) \vdash_M (q', w')$ if and only if there is a $u \in \Sigma \cup \{e\}$ such that $w = uw'$ and $(q, u, q') \in \Delta$. Note that $\vdash_M$ need not be a function; for some configurations (q, w), there may be several pairs (q', w') —or none at all— such that $(q, w) \vdash_M (q', w')$. As before, $\vdash_M^*$ is the reflexive, transitive closure of $\vdash_M$ and a string $w \in \Sigma^*$ is **accepted** by M if and only if there is a state $q \in F$ such that $(s, w) \vdash_M^* (q, e)$. Finally $L(M)$, the language accepted by M, is the set of all strings accepted by M.

Examples

To see that a nondeterministic finite automaton can be a much more convenient device to design than a deterministic finite automaton, consider the language L = (ab U aba)*, which is accepted by the deterministic finite automaton illustrated in below



L is accepted by the simple nondeterministic device shown in fig below



Fog: NFA for L = (ab U aba)*

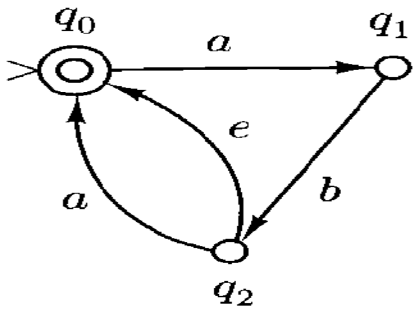


Fig: NFA for L = (ab U aba)* with e-transition

Example 2: Design a nondeterministic finite automata (NFA) that accept the set of all strings containing an occurrence of the pattern bb or of the pattern bab.

Solution:

Let required NFA M= (K, ∑, Δ, s, F), where

K= { q_O , q_1, q_2, q_3, q_4 }

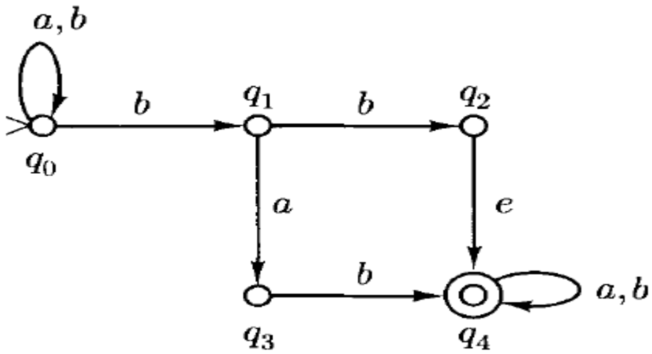
∑= {a, b }

s= q_O

F= { q_4 } and

$$\Delta = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, b, q_1), (q_1, b, q_2), (q_1, a, q_3), (q_2, e, q_4), (q_3, b, q_4), (q_4, a, q_4), (q_4, b, q_4)\}.$$

and state diagram is given by



When M is given the string *bababab* as input, several different sequences of moves may ensue. For example, M may wind up in the non final state q_o in case the only transitions used are (q_O, a, q_0) and (q_O, b, q_0):

$$\begin{aligned}(q_0, bababab) &\vdash_M (q_0, ababab)\\ &\vdash_M (q_0, babab)\\ &\vdash_M (q_0, abab)\\ &\vdots\\ &\vdash_M (q_0, e)\end{aligned}$$

The same input string may drive M from state q_0 to the final state q_4, and indeed may do so in three different ways. One of these ways is the following.

$$\begin{aligned}(q_0, bababab) &\vdash_M (q_1, ababab)\\ &\vdash_M (q_3, babab)\\ &\vdash_M (q_4, abab)\\ &\vdash_M (q_4, bab)\\ &\vdash_M (q_4, ab)\\ &\vdash_M (q_4, b)\\ &\vdash_M (q_4, e)\end{aligned}$$

Since a string is accepted by a nondeterministic finite automaton if and only if there is at least one sequence of moves leading to a final state, it follows that *bababab* $\in$ L(M)

E-closure

Epsilon closure (ε- closure) of a state q is the set that contains the state q itself and all other states that can be reached from state q by following ε transitions.

We use the term, ECLOSE(q0) to denote the Epsilon closure of state q0.

Suppose that the given NFA M= (Q,∑, δ , q_0,F) . For any set s $\subseteq$ Q of states of M, we define the e-closure of S, denoted by E(S) to be the set of states reachable from s via zero or more e transition in a row.

Formally for I ≥ 0 ,we define $E_i(S)$ inductively to be the set of states reachable from s via exactly i many e-transitions, so that $E_0(s)$=s and for I ≥ 0

$E_{i+1}(s) = \{ r \in Q \mid E_q \in E_i(s) : r \in \delta(q, e) \}$

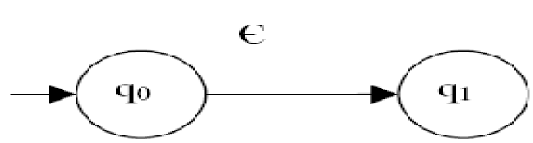
$\quad = \bigcup_{q \in Ei(s)} \delta(q, e)$

$E(s) \quad = \bigcup_{i \geq = 0} Ei(s)$

Namely, the set of all states reachable from s via zero or more e-transitions.

e-closure(q0) denotes a set of all vertices p such that there is a path from q0 to p labeled .

Example



C-closure(q0)={q0,q1}

Equivalence of DFA and NFA

A DFA is just a special type of NFA . In a DFA it so happens that the transition relation Δ is in fact a function from K x $\sum$ to K i.e. NFA (K,$\sum$, δ, s, F) is deterministic if and only if there are no transition of the form (q, e, p) in δ and for each q ∈ K and a ∈ $\sum$ there is a exactly one p ∈ K such that (q, a, p) ∈ Δ

It is there fore evident that the class of languages accepted by DFA is a subset of the class of languages accepted by NFA. Rather surprisingly these classes are in fact equal.

Despite the power and generality enjoyed by NFA, they are no more powerful than the NFA in terms of the language they accept.

NFA can always be converted into an equivalent deterministic one.

Formally, we say that the two finite automata M_1 and M_2 (NFA and DFA) are equivalent if and only if $L(M_1) = L(M_2)$. Thus two automata are considered to be equivalent if they accept the same language, even though they may use different methods to do so.

Theorem : For each NFA , there is equivalent DFA.

Proof:

Let M=(K, $\sum$,Δ, s, F) be a NFA we shall construct a DFA **M'=**(K' , $\sum$, δ', s', F') equivalent to M. View a NFA as a occupying at any moment not a single state but a set of states, namely all the states that can be reached from the initial state by means of the input consumed thus far.

- The set of states of M' will be $K' = 2^K$, the power set of states of M
- The set of final state of M' will consist of all those subset of K that contains at least one final state of M. i.e. $F'=\{ Q \subseteq K : Q \cap F \neq \emptyset \}$
- The definition of transition function of M' is slightly complicated . The basic idea is that a move of M' on reading an input symbol a , possibly followed by any number of e-moves of M.

  For every $Q \subseteq K$ and $a \in \sum$

  $\delta'(Q, a) = E\left(\bigcup_{q \in Q} \delta(q, a) \right)$

  To formalize this idea we need a special definition

  For any state q ϵ K , let E(q) be the set of all states of M that are reachable from state q without reading any input (e-moves) i.e.

  $$E(q) = \{ p \in K : (q , e) \vdash^*_M (p , e) \}$$

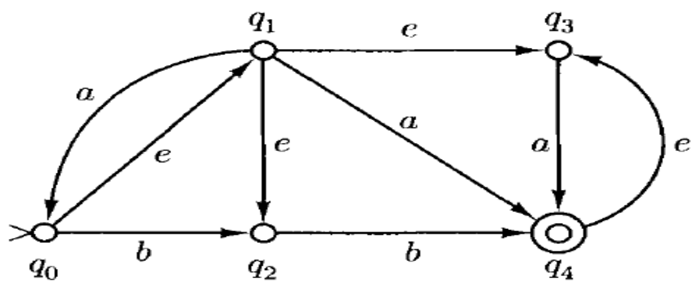
  To put it otherwise, E(q) is the closure of the set {q} under the relation
  { (p, r) : there is a transition (p, e, r) ϵ Δ}.

  Thus, E(q) can be computed by the following algorithm:

  Initially set E(q) := {q};
  while there is a transition (p, e, r) ϵ Δ. with p ϵ E(q)

and $r \notin E(q)$ do: $E(q) := E(q) \cup \{r\}$.

Example: Convert the e-NFA given below to its corresponding DFA



Solution:
Here
Given NFA M= $(K,\sum,\delta, s, F)$
$E(q_0)= \{ q_0, q_1, q_2, q_3\}$
$E(q_1) = \{q_1,q_2,q_3\}$, and
$E(q_2) = \{q_2\}$

We are now ready to define formally the deterministic automaton $M'= (K',\sum,\delta', s', F')$ that is equivalent to M, where

- $K' =2^K$
- $S'=E(s)$
- $F'=\{ Q\subseteq K : Q \cap F \neq \emptyset \}$ and
  for each $Q\subseteq K$ and each symbol $a \in \sum$,define

$\delta'(Q, a) = \bigcup\{E(p) : p \in K \text{ and } (q, a, p) \in \Delta \text{for some } q \in Q \}$
i. e. $\delta'(Q, a)$ is taken to be the set of all states of M to which M can go by reading input a and possibly followed several e-moves.

$E(q0) = \{ q0, q1,q2,q3 \}$

$\delta'(s', a)= E(q0) \cup E(q4) = \{ q0, q1,q2,q3 , q4\}$
Similarly,

$\delta'(s', b)= E(q2) \cup E(q4)= \{ q2,q3 , q4\}$

Again for newly introduced states

$\delta'(\{ q0, q1,q2,q3 , q4\} , a)= \{ q0, q1,q2,q3 , q4\}$

$\delta'(\{ q0, q1,q2,q3 , q4\} , b)= \{q2,q3 , q4\}$

Again for newly introduced states $\{q2,q3 , q4\}$

$\delta'(\{q2, q3 , q4\} , a)= \{q3 , q4\}$

δ'({q2,q3 , q4} , b)= {q3 , q4}

Again,

δ'({q3 , q4} , a)= {q3 , q4}

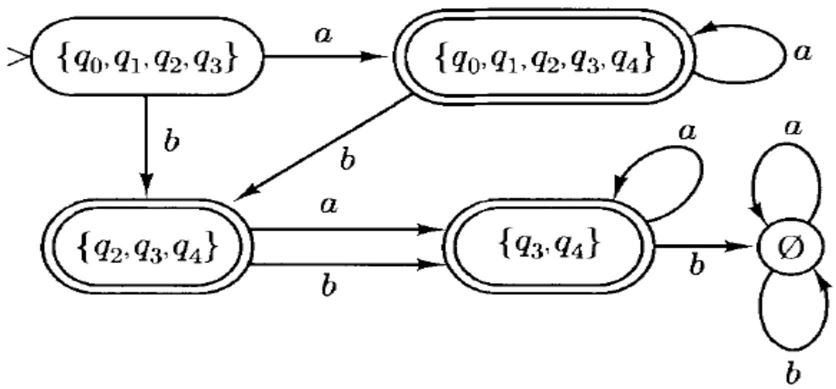
δ'({q3 , q4} , b)= { Ø }

and finally

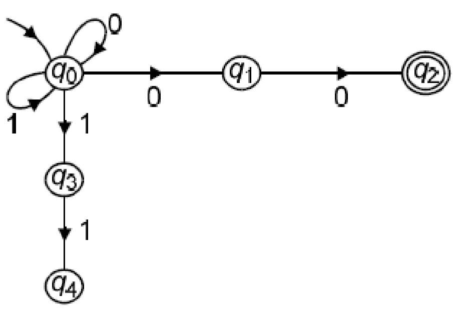
δ'({Ø , a)= { Ø }

δ'({Ø , b)= { Ø }

F' , the set of final states contains each set of states of which q4 is member , since q4 is the sole final member of F.



Example : Given the NFA as shown in fig. below, determine the equivalent DFA.



Solution

The given NFA has q_2 and q_4 as final states. It accepts strings ending in 00 or 11. The state table is shown below.

	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	$\{q_2\}$	$\varnothing$
q_2	$\varnothing$	$\varnothing$
q_3	$\varnothing$	$\{q_4\}$
q_4	$\varnothing$	$\varnothing$

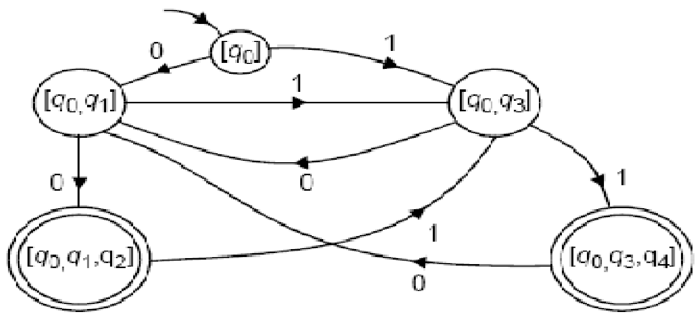
The conversion of NFA to DFA is done through the subset construction.

δ' is given by the following state table.

	0	1
$\rightarrow [q_0]$	$[q_0, q_1]$	$[q_0, q_3]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_3]$
$[q_0, q_3]$	$[q_0, q_1]$	$[q_0, q_3, q_4]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_3]$
$[q_0, q_3, q_4]$	$[q_0, q_1]$	$[q_0, q_3, q_4]$

Any state containing q_2 or q_4 will be a final state.

The DFA is shown below.



Properties of Regular Languages

1. **Closure Properties**
2. **Decision Properties**

Closure Properties of Regular Languages

If a set of regular languages are combined using an operator, then the resulting language is also regular.

Closure property is a statement that a certain operation on languages when applied to languages in a class , produces a result is also in that class.

Theorem: The class of languages accepted by finite automata is closed under

- union;
- concatenation;
- Kleene star;
- complementation;
- intersection.
- reverse
- set difference

We shall show that the set of regular languages is closed under each of the operations defined above.

The general approach is as follows:

1. Build automata (DFA or NFA) for each of the languages involved.
2. Show how to combine the automata in order to form a new automata which recognizes the desired language
3. Since the language is represented by NFA/DFA , we shall conclude that the language is regular.

Proof : In each case we show how to construct an automaton M that accepts the appropriate language, given two automata M_l and M_2 (only M_l in the cases of Kleene star and complementation).

Union

The set of regular languages is closed under the union operation, i.e., if L1 and L2 are regular languages over the same alphabet ∑, then L1 ∪ L2 is also a regular language.

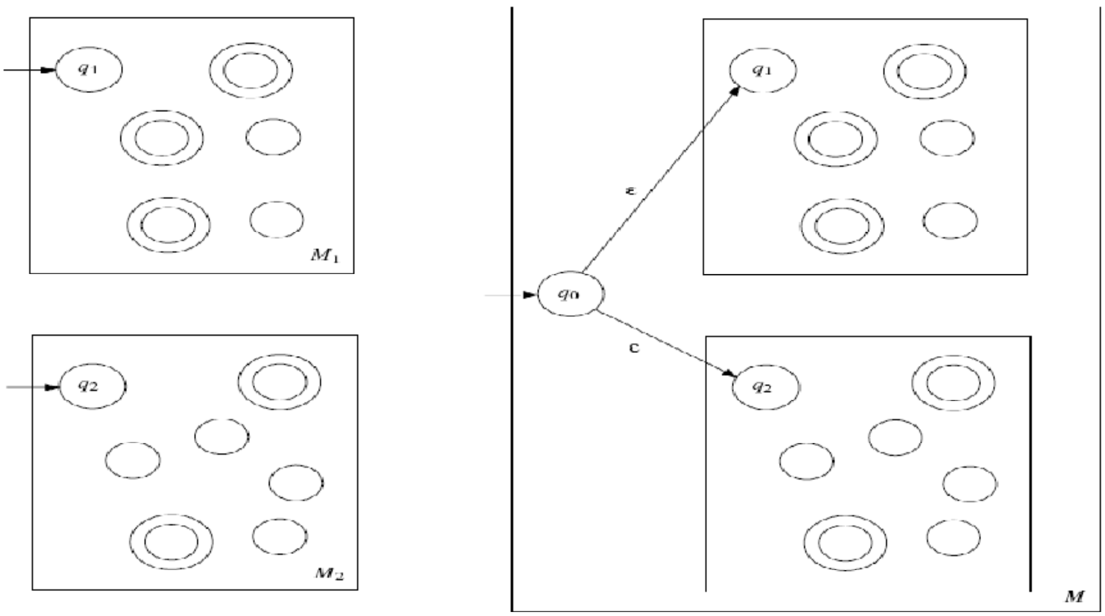


Figure 2.1: *The NFA M accepts $L(M_1) \cup L(M_2)$.*

Proof.

Since L_1 is regular, there is, an NFA $M_1 = (K_1, \sum, \delta_1, q_1, F_1)$, such that $L_1 = L(M_1)$. Similarly, there is an NFA M2 = $(K_2, \sum, \delta_2, q_2, F_2)$, such that L2 = L(M2).

We may assume that $K_1 \cap K_2 = \emptyset$, (K_1 and K_2 are disjoint sets) because otherwise, we can give new “names” to the states of K1 and K_2.

From these two NFAs, we will construct an NFA M = (K, ∑, δ, q0, F), such that L(M) = L1 ∪ L2. The construction is illustrated in Figure 2.1. The

NFA M is defined as follows:

1. $K = K_1 \cup K_2 \cup \{q_0\}$, where q_0 is a new state.
2. q_0 is the start state of M.
3. $F = F_1 \cup F_2$.
4. $\delta : \delta 1 \cup \delta 2 \cup \{ (q_0, e, q_1), (q_0, e, q_2)\}$ is defined as follows:

For any r ∈ K and for any a ∈ Σ

$\delta(r, a) = \delta 1(r, a)$ if $r \in K1$
$= \delta 2(r, a)$ if $r \in K2$
$= \{q_1, q_2\}$ if $r = q_0$ and $a = e$
$= \emptyset$ if $r = q_0$ and $a \neq e$

Steps in Union of L1 and L2

- Create a new start state
- Make a e-transition from the new start state to each of the original start states.

Concatenation

The set of regular languages is closed under the concatenation operation, i.e., if L1 and L2 are regular languages over the same alphabet , then L1L2 is also a regular language.

Proof. Since L_1 is regular, there is, an NFA $M_1 = (K_1, \sum, \delta_1, q_1, F_1)$, such that $L_1 = L(M_1)$.
Similarly, there is an NFA $M2 = (K_2, \sum, \delta_2, q_2, F_2)$, such that $L2 = L(M2)$.
We may assume that $K_1 \cap K_2 = \emptyset$, (K_1 and K_2 are disjoint sets) because otherwise, we can give new "names" to the states of K1 and K_2.
From these two NFAs, we will construct an NFA $M = (K, \sum, \delta, q0, F)$, such that $L(M) = L1 . L2$.
The construction is illustrated in Figure 2.2. The
NFA M is defined as follows:

1. $K = K_1 \cup K_2$,
2. $q_0 = q_1$ is the start state of M.
3. $F = F_2$.
4. δ : is defined as follows:

For any $r \in K$ and for any $a \in \Sigma$

$\delta(r, a) = \delta 1(r, a)$ if $r \in K1$ and $r \notin F1$
$= \delta 1(r, a)$ if $r \in F1$ and $a \neq e$
$= \delta 1(r, a) \cup \{q_2\}$ if $r \in F1$ and $a = e$
$= \delta 2(r, a)$ if $r \in K2$

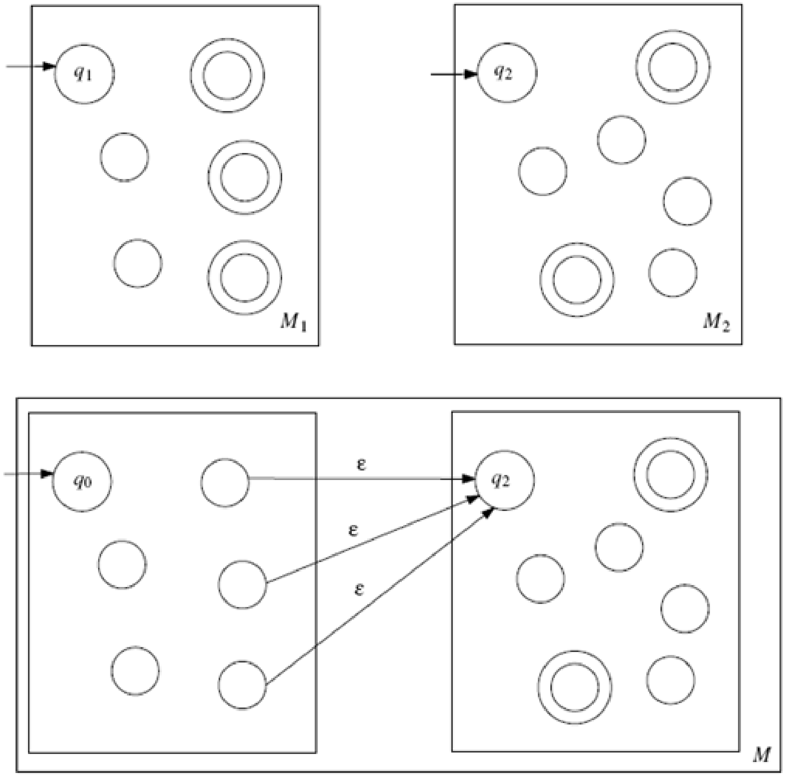


Fig 2.2 The NFA M accepts L(M1).L(M2)

Steps in Concatenation of L_1 and L_2

- Put a e-transition from each final state of L1 to the initial state of L2.
- Make the original final states of L1 non final.

Kleene Star

If L is a regular language , then L* is also a regular language.

Let Σ be the alphabet of L1 and let N = (K_1,Σ, δ1, q_1, F_1) be an NFA, such that L1 = L(N). We will construct an NFA M = (K,Σ, δ, q_0, F), such that L(M) = L_1^{*}. The construction is illustrated in Figure 2.3. The NFA M is defined as follows:

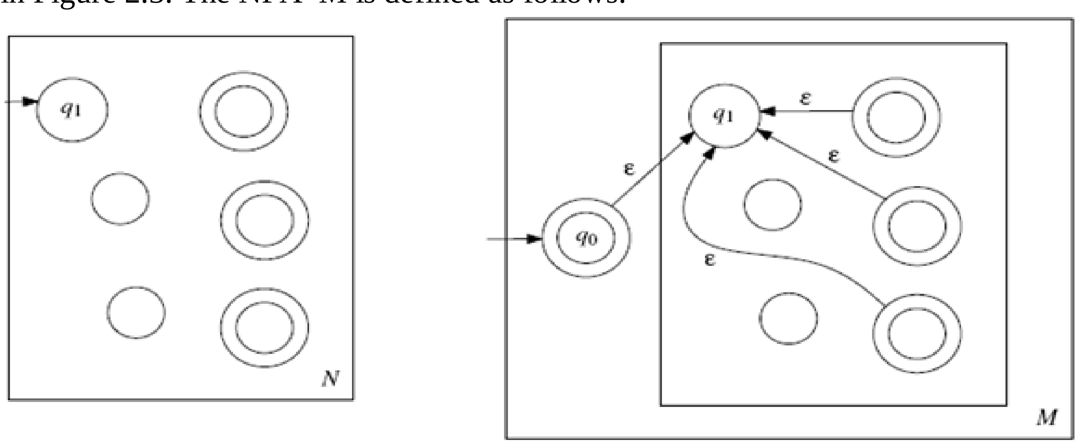


Fig: 2.3 The NFA M accepts L(N)*

Where

$K = K1 \cup \{q_0\}$, where q_0 is a new state

q_0 is the start state of m

$F = q_0 \cup F_1$, since $e \in L1^*$, q_0 has to be an accepting state (final)

δ : for any $r \in K$ and $a \in \Sigma$

$\delta(r, a) = \delta1(r, a)$ if $r \in k1$ and $r \notin F1$

$= \delta1(r, a)$ if $r \in F1$ and $a \neq e$

$= \delta1(r, a) \cup \{q_0\}$ if $r \in k1$ and $r =$ F1

$= \{q1\}$ if $r=q0$ and $a= e$

$= \emptyset$ if $r=q0$ and $a \neq e$

Steps in Kleene star of L1

- Make a new start state; connect it to the original start state with a e-transition.
- Make a new final state; connect the original final state (which becomes non final) to it with e-transitions.
- Connect the new start state and new final state with a pair of e-transitions.

Complementation

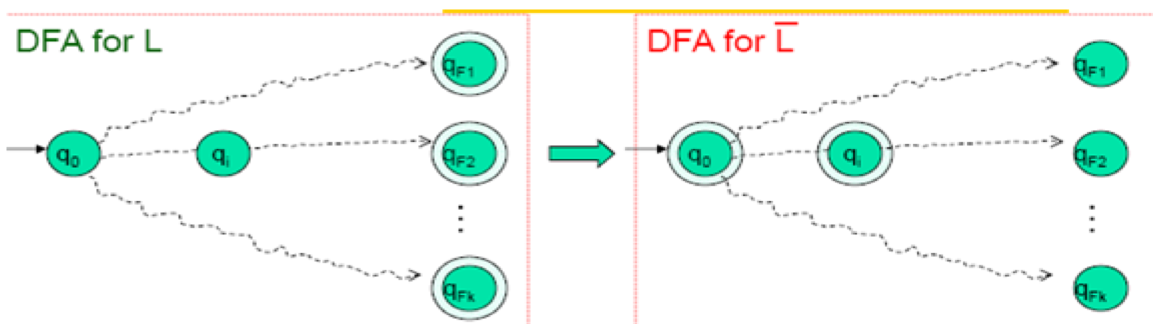
The set of regular languages is closed under the complement operations:

If L1 is a regular language over the alphabet Σ, then the complement $L1 = \{w \in \Sigma^* : w \notin L1\}$ is also a regular language.

If L is an Regular over Σ , then Complement of $L = \Sigma^* - L$

Proof

To show Complement of L is also regular , Convert every final state into non final and every non final state to final state.



Let $M = (K,\Sigma, \delta, s, F)$ be a deterministic finite automaton.

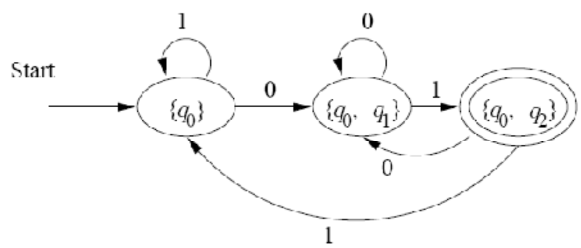
Then the complementary language $L = \Sigma^* - L(M)$ is accepted by the deterministic finite automaton

$M1 = (K,\Sigma, \delta, s, K - F)$. That is, M1 is identical to M except that final and non final states are interchanged.

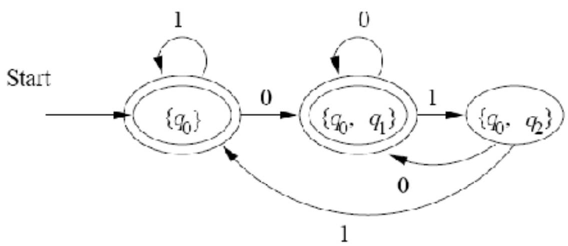
Steps Complementation of L_1

- Start with a complete DFA, not with an NFA
- Make every final state non final and every non final state final.

Let L be recognized by the DFA below



Then $\overline{L}$ is recognized by



Steps in Reverse of L1

- Start with an automaton with just one final state.
- Make the initial state final and final state initial.
- Reverse the direction of every arc

Steps for Intersection and Set Difference

Just as with the other operations, it can be proved that regular languages are closed under intersection and set difference by starting with automata for the initial languages, and constructing a new automaton that represents the operation applied to the initial languages.

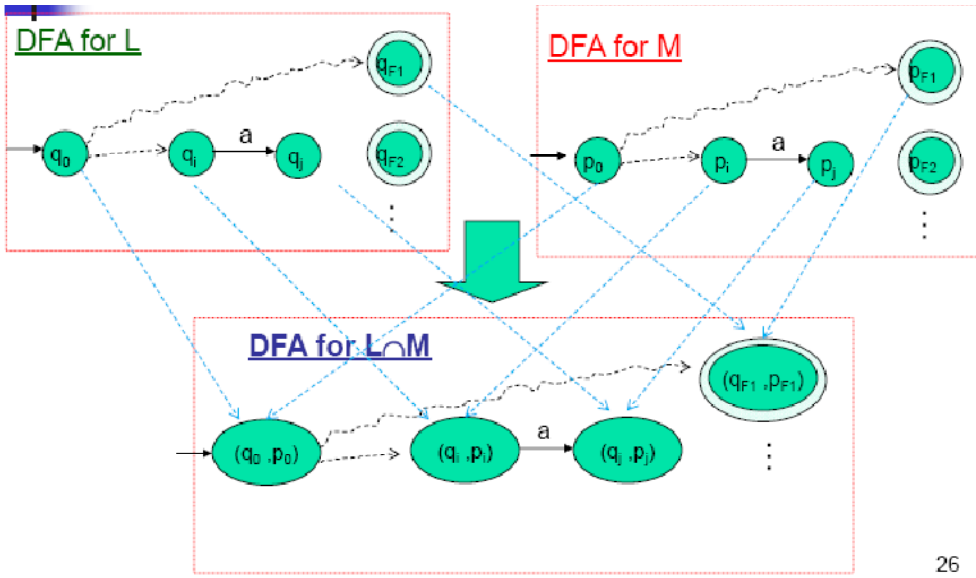
In this construction, a completely new machine is formed, whose states are labelled with an ordered pair of state names: the first element of each pair is a state from L1 and the second element of each pair is a state from L2.

- Begin by creating a start state whose label is (start state of L1, start state of L2).
- Repeat the following until no new arcs can be added:

(1) Find a state (A, B) that lacks a transition for some x in S.

(2) Add a transition on x from state (A, B) to state (δ (A, x), δ (B, x)). (If this state does not already exist, create it).

The same construction is used for both intersection and set difference. The distinction is in how the final states are selected.



If L1 and L1 are regular languages, then so is $L_1 \cap L_2$.

$$L_1 \cap L_2 = \Sigma^* - ((\Sigma^* - L_1) \cup (\Sigma^* - L_2))$$

Indirect way to proof is

Proof: Observe that $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$. Since regular languages are closed under union and complementation, we have

- $\overline{L_1}$ and $\overline{L_2}$ are regular
- $\overline{L_1} \cup \overline{L_2}$ is regular
- Hence, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ is regular.

Direct Way to proof

Proof: Let A and B be DFA's whose languages are L1 and L2, respectively.

- Construct C, the product automaton of A and B.
- Make the final states of C be the pairs consisting of final states of both A and B.

Since L_1 is regular, there is, an DFA $M_1 = (K_1, \Sigma, \delta_1, q_1, F_1)$, such that $L_1 = L(M_1)$. Similarly, there is an DFA $M2 = (K_2, \Sigma, \delta_2, q_2, F_2)$, such that $L2 = L(M2)$. We can construct a DFA $M_1 \cap M2 = M = (K, \Sigma, \delta, q_0, F)$

Where

$K = K_1 x K_2$,

$\Sigma = \Sigma$,

$q_0 = (q_1, q_2)$,

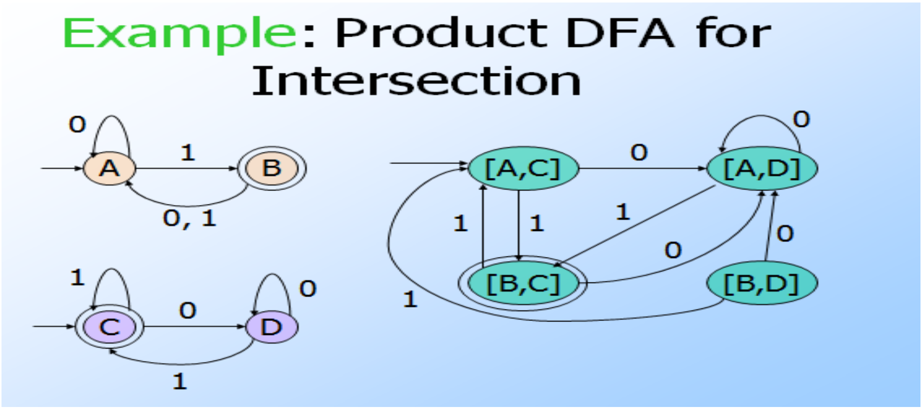
$F = F_1 \text{ x } F_2$

and δ is defined Such a way that ,

$\delta((p, q),a) = (\delta_1(p, a) , \delta_2(q, a))$, where p in K_1 and q in K_2.
This construction ensures that a string w will be accepted if and only if w reaches an accepting state in both input DFAs.

Steps Intersection

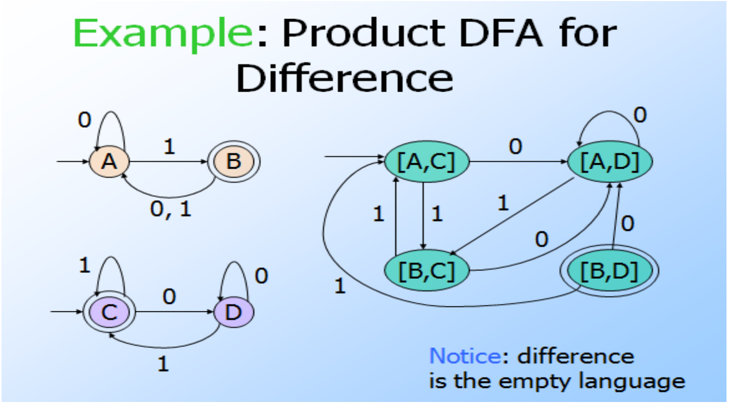
Make a state (A, B) as final if both
(i) A is a final state in L1 and
(ii) B is a final state in L2



Set Difference

Proof:
Let A and B be DFA’s whose languages are L and M, respectively.
Construct C, the product automaton of A and B.
Make the final states of C be the pairs where A-state is final but B-state is not.

Mark a state (A, B) as final if A is a final state in L1, but B is not a final state in L2.



State Minimization

Minimization of DFA

For a given language, many DFA may exist that accept it. The DFA we produce from a NFA may contain many dead states, inaccessible states and indistinguishable states. All these unnecessary states can be eliminated from the DFA through a process called minimization.
For practical applications, it is desirable that number of states in the DFA is minimum.
The Algorithm for minimizing a DFA as follows:

1. Step 1: Eliminate any state that cannot be reached from the start state.
2. Step 2: partition the remaining states into blocks so that all the states in the same block are equivalent and no pair of states from different blocks are equivalent.

Example

Minimise the following DFA

Current state	input symbol	
	a	b
→q0	q5	q1
q1	q2	q6
*q2	q2	q0
q4	q5	q7
q5	q6	q2
q6	q4	q6
q7	q2	q6
q3	q6	q2

Step 1: Eliminate any state that can't be reached from the start state

In above, the state q3 can't be reached. So remove the corresponding to q3 from the transition table. Now the new transition table is

Current state	input symbol	
	a	b
→q0	q5	q1
q1	q2	q6
*q2	q2	q0
q4	q5	q7
q5	q6	q2
q6	q4	q6
q7	q2	q6

Step 2: Divided the rows of the table into 2 sets as

1. one set containing only rows which starts from non final states

Set 1

q0	q5	q1
q1	q2	q6
q4	q5	q7
q5	q6	q2
q6	q4	q6
q7	q2	q6

2. another set containing those rows which start from final states

* q2	q2	q0

Step 3a: Consider the set 1

q0	q5	q1	Row1
q1	q2	q6	Row2
q4	q5	q7	Row3

q5	q6	q2	Row4
q6	q4	q6	Row5
q7	q2	q6	Row6

Row 2 and Row 6 are similar since q1 and q7 transit to same states on inputs a and b so remove one of them (for instance q7) and replace q7 with q1 in rest we get

Set 1

q0	q5	q1	Row1
q1	q2	q6	Row2
q4	q5	q1	Row3
q5	q6	q2	Row4
q6	q4	q6	Row5

Now Row 1 and Row 3 are similar. So remove one of them (for instance q4) and replace q4 with q0 in the rest

we get

Set 1

q0	q5	q1	Row1
q1	q2	q6	Row2
q5	q6	q2	Row3
q6	q0	q6	Row4

Now there are no more similar rows

3b. Consider the set 2

Set 2

*q2	q2	qq0

Do the same process for set 2

But it contains only one row .It is already minimized

Step 4

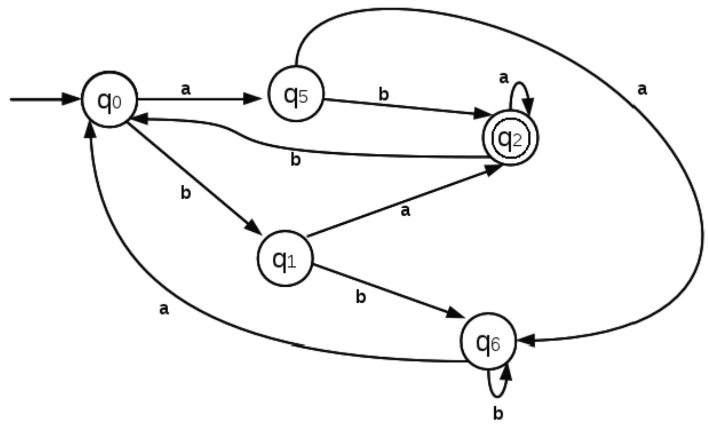
Combine set 1 and set 2

we get

Current state	input symbol	
	a	b
→q0	q5	q1
q1	q2	q6
q5	q6	q2
q6	q0	q6
*q2	q2	q0

Now this is minimized DFA

The transition diagram is



Example 2:

Minimise the following DFA,

Current State	Input symbol 0	1
$\longrightarrow q_0$	q_1	q_3
q_1	q_0	q_3
q_2	q_1	q_4
$*q_3$	q_5	q_5
q_4	q_3	q_3
$*q_5$	q_5	q_5

Step 1: Eliminate any state that cannot be reached from the start state.

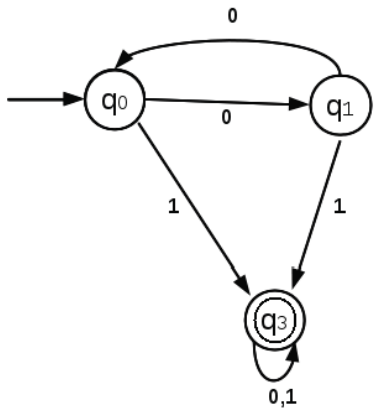
This is done by enumerating all the simple paths in the graph of the DFA beginning from the start state. Any state that is not part of some path is unreachable.

In the above, the states q_2 and q_4 cannot be reached. So remove the rows corresponding to q_2 and q_4 from the transition table

Similarly do yourself
final Minimised DFA diagram is

Now this is a minimised DFA.

Transition diagram is,



1. Minimise the following DFA represnted as transition table,

Current State	Input symbol	
	0	1
$\longrightarrow q_0$	q_1	q_2
q_1	q_2	q_3
q_2	q_2	q_4
$*q_3$	q_3	q_3
$*q_4$	q_4	q_4
q_5	q_5	q_4

Equivalence of Regular languages and Finite Automata

Finite Automata and Regular Expressions are equivalent. To show this:
– Show we can express a DFA as an equivalent RE
– Show we can express a RE as an e-NFA. Since the e-NFA can be converted to a DFA and the DFA to an NFA, then RE will be equivalent to all the automata we have described.

Theorem : A language is regular if and only if it is accepted by a finite automaton.

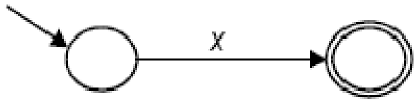
Only If Part

(a) Regular Expression to NFA Construction

Proof:

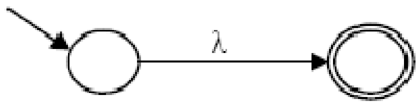
Recall that the class of regular languages is the smallest class of languages containing the empty set ø and the singletons *a*, where *a* is a symbol, and closed under union concatenation, and Kleene star. It is evident (see Figure below) that the empty set and all singletons are indeed accepted by finite automata; and by Theorem the finite automaton languages are closed under union, concatenation, and Kleene star. Hence every regular language is accepted by some finite automaton.

1. For any x in Σ, the regular expression denotes the language $\{x\}$. The NFA (with a single start state and a single final state) as shown below, represents exactly that language.



NFA for x

2. The regular expression l denotes the language $\{\lambda\}$ or $\{e\}$ that is the language containing only the empty string.

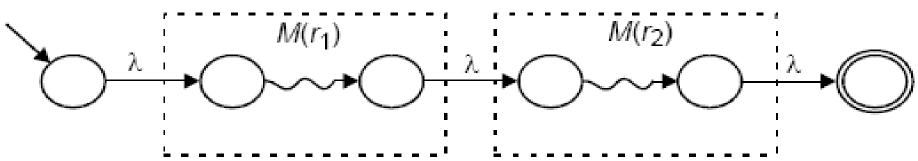


NFA for λ

3. The regular expression ø denotes the language ø; no strings belong to this language, not even the empty string.

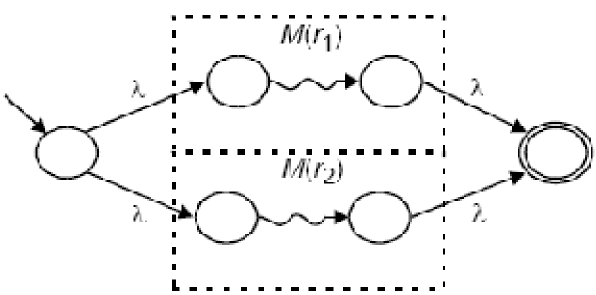
NFA for $\varnothing$

4. For juxtaposition/concatenation, strings in L(r1) followed by strings in L(r2), we chain the NFAs together as shown.



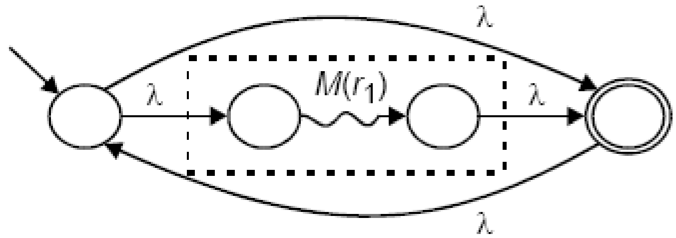
NFA for r_1r_2

5. The "+" denotes "or" in a regular expression, we would use an NFA with a choice of paths (Union)



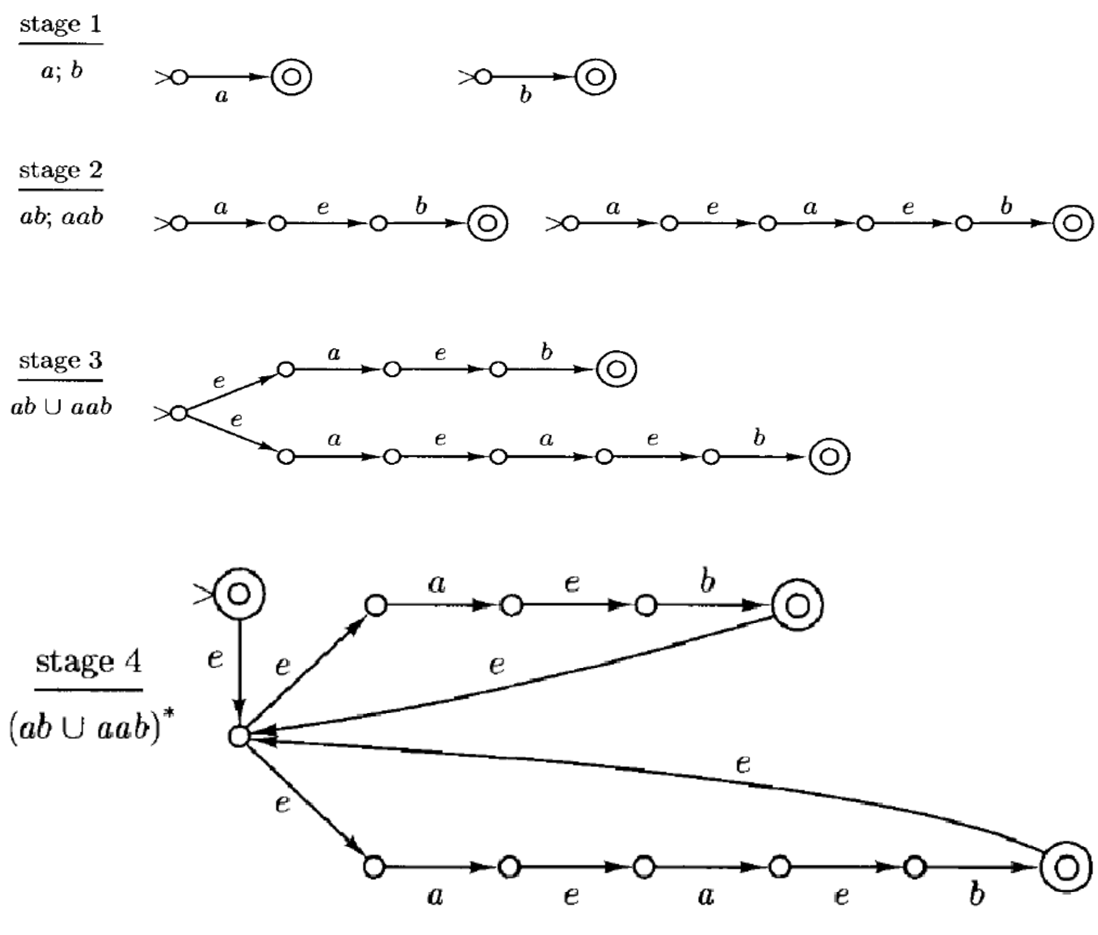
NFA for $r_1 + r_2$

6. The star (*) denotes zero or more applications of the regular expression, hence a loop has to be set up in the NFA.



Examples: See on Book /Class Notes

Consider the regular expression (ab U aab)*. A nondeterministic finite automaton accepting the language denoted by this regular expression can be built up using the methods in the proof of the various parts of Theorem.



IF Part

(b) Finite Automata to Regular Expression

Turning a DFA into a RE

Theorem: If L=L(A) for some DFA A, then there is a regular expression R such that L=L(R).

Proof

– Construct GNFA, Generalized NFA

– State Elimination

- Eliminates states of the automaton and replaces the edges with regular expressions that include the behavior of the eliminated states.
- Eventually we get down to the situation with just a start and final node, and this is easy to express as a RE

Let M = (K,Σ,Δ, s, F) be a finite automaton (not necessarily deterministic). We shall construct a regular expression R such that L(R) = L(M).

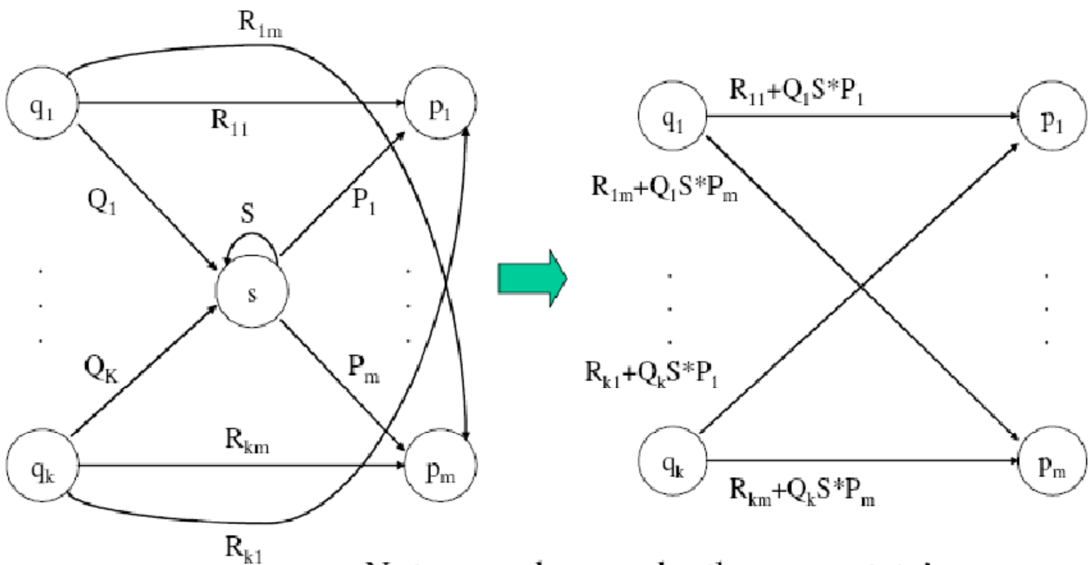
The basic approach to convert NFA, to Regular Expressions is as follows:

1. If an NFA has more than one final state, convert it to an NFA with only one final state. Make the original final states non final, and add a e-transition from each to the new (single) final state.
2. Consider the NFA to be a generalised transition graph, which is just like an NFA except that the edges may be labeled with arbitrary regular expressions. Since the labels on the edges of an NFA may be either e or members of each of these can be considered to be a regular expression.
3. Removes states one by one from the NFA, relabeling edge as you go, until only the initial and the final state remain.
4. Read the final regular expression from the two state automatons that results. The regular expression derived in the final step accepts the same language as the original NFA.

Examples

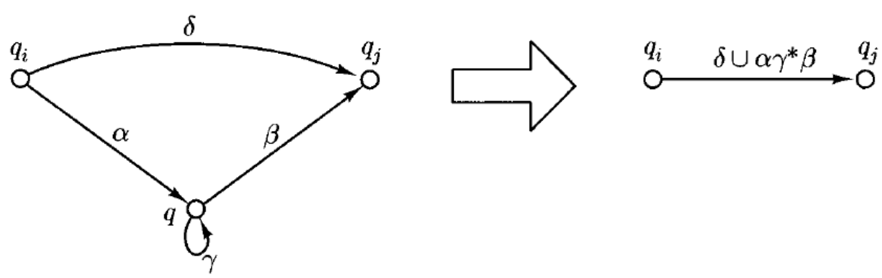
State Elimination

- Consider the figure below, which shows a generic state s about to be eliminated. The labels on all edges are regular expressions.
- To remove s, we must make labels from each q_i to p_1 up to p_m that include the paths we could have made through s.

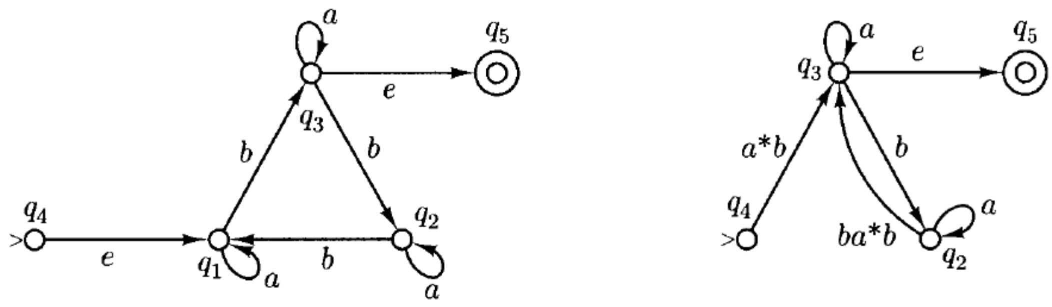


Note: q and p may be the same state!

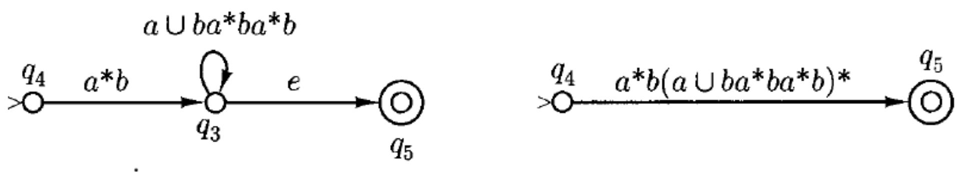
Example 2



Example 3



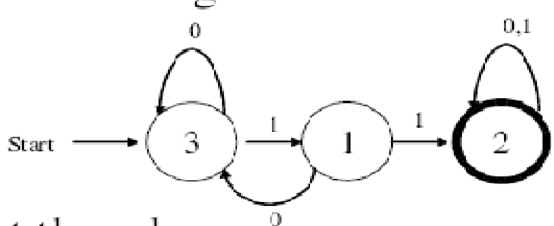
(a) (b)



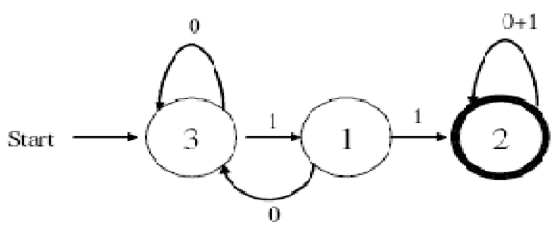
(c) (d)

So regular expression R = a*b(aUba*ba*b)*

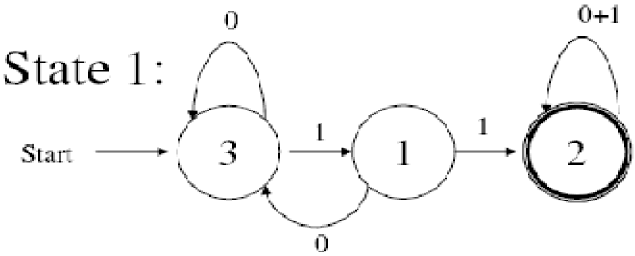
- Convert the following to a RE



- First convert the edges to RE's:

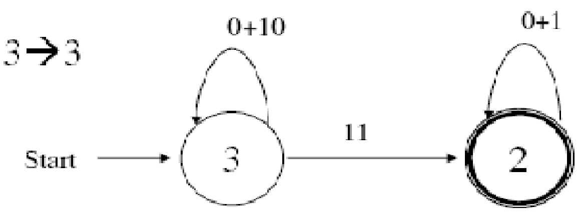


- Eliminate State 1:



- To:

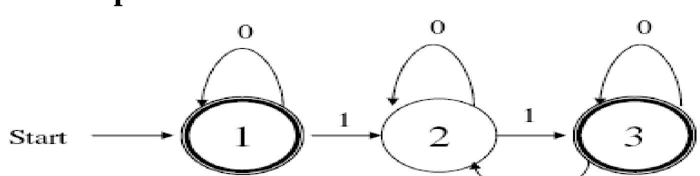
Note edge from 3→3



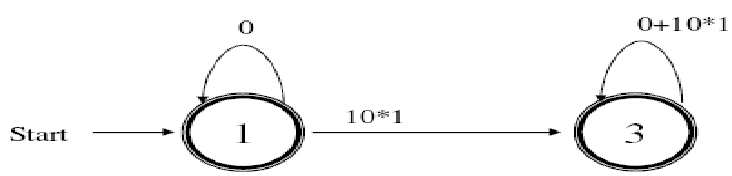
Answer: (0+10)*11(0+1)*

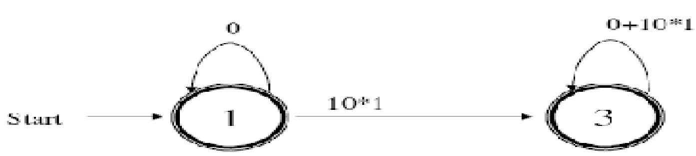
Second Example

Automata that accepts even number of 1's

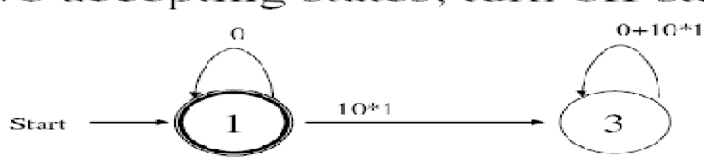


- Eliminate state 2:

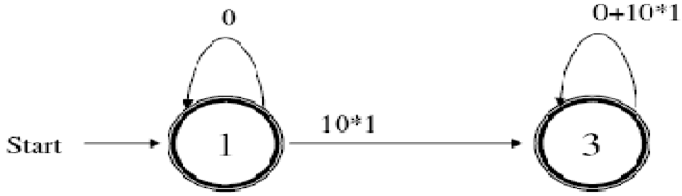




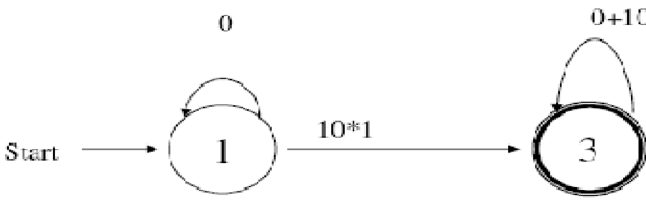
- Two accepting states, turn off state 3 first



This is just 0*; can ignore going to state 3 since we would "die"



- Turn off state 1 second:



This is just 0*10*1(0+10*1)*

Combine from previous slide to get
0* + 0*10*1(0+10*1)*

Generalized Finite Automata

Generalized finite automaton, with transitions that may be labeled not only by symbols in Σ or e, but by entire regular expressions. In fig above fig (a) is Finite automata and fid (d) is Generalized Finite Automata of fig a.

Pumping Lemma for Regular Languages

Let L be a regular language. There is an integer $n \geq 1$ such that any string $W \in L$ with $|w| \geq n$ can be rewritten as $W = xyz$ such that

- $y \neq e$,
- $|xy| \leq n$, and
- $xy^iz \in L$ for each $i \geq 0$.

Examples

See on class notes

Decision property

Decision property for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.

Example : Is language L empty?

1. The Membership Question
2. The Emptiness Problem
3. The Infiniteness Problem