MODULE-1

INTRODUCTION TO AUTOMATA THEORY

Automata theory is the study of abstract computing devices or "machines". Before there were computers in the 1930's, Allan Turing studied an abstract machine that had all the capabilities of today's computers at least as far as in what they could compute Turing's goal was to describe precisely the boundary between what a computing machine could do and what it could not do; his conclusions apply not only to his abstract Turing machines but to today's real machines.

Why Study Automata Theory?

There are several reasons why the study of automata and complexity is an important part of the core of Computer Science.

Finite automata are a useful model for many important kinds of hardware and Software.

1. Software for designing and checking the behavior of digital circuits.
2. The "lexical analyzer" of a typical compiler, that is, the compiler component that breaks the input text into logical units, such as identiers, keywords and punctuation.
3. Software for scanning large bodies of text, such as collections of WebPages, to find occurrences of words, phrases or other patterns.
4. Software for verifying systems of all types that have a finite number of distinct states, such as communications protocols or protocols for secure exchange of information.

Example: Perhaps the simplest nontrivial finite automaton is an on/off Switch. The device remembers whether it is in the "on" state or the "off" state, and it allows the user to press a button whose effect is different, depending on the state of the switch_ That is, if the switch is in the on state, then pressing the button changes it to the on state, and if the switch is in the on state, then pressing the same button turns it to the on state.

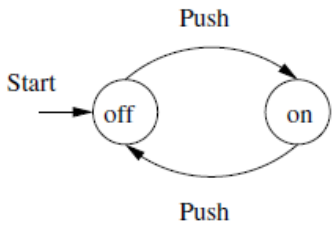


Figure 1.1 A finite automaton modelling an on/off switch

The Central Concepts of Automata Theory

Let us know the most important definitions of terms that pervade the theory of automata. These concepts include the "alphabet" (a set of symbols), "strings" (a list of symbols from an alphabet) and "language" (a set of strings from the same alphabet).

i) Alphabets:

An alphabet is a finite, nonempty set of symbols. Conventionally, we use the symbol $\sum$ for an alphabet.

Common alphabets include:

1. $\sum = \{0, 1\}$ the binary alphabet.
2. $\sum = \{a, b, \ldots., z\}$, the set of all lower-case letters.
3. The set of all ASCII characters, or the set of all printable ASCII characters.

ii) Strings:

A string (or sometimes word) is a finite sequence of symbols chosen from some alphabet. For example, 01101 is a string from the binary alphabet $\sum=\{0, 1\}$. The string 111 is another string chosen from this alphabet.

iii)The Empty String

The empty string is the string with zero occurrences of symbols. This string, denoted ε, is a string that may be chosen from any alphabet whatsoever.

iv)Length of a String

Length of a string is the number of positions for symbols in the string. Ex: 01101 has length 5. The standard notation for the length of a string w is |w|. For example, |011|=3 and |ε| =0.

v) Powers of an Alphabet

If $\sum$ is an Alphabet we can express the set of all strings of a certain length from that alphabet by using an exponential notation We define $\sum^k$ to be the set of strings of length k each of whose symbols is in $\sum$.

Example: Note $\sum^0 =\{\varepsilon\}$, regardless of what alphabet $\sum$ is.

That is, ε is the only string whose length is 0.

If $\sum$={0,1}, then

$\sum^1$={0,1},

$\sum^2$={00,01,10,11}

$\sum^3$={000,001,010,011,100,101,110, 111} and so on.

The set of all strings over an alphabet $\sum$ is conventionally denoted $\sum^*$.

For instance, {0, 1}* ={ ε, 0,1,00,01,10,11,000,……..}

$\sum^* = \sum^0 \cup \sum^1 \cup \sum^2 \cup$……..

Sometimes, we wish to exclude the empty string from the set of strings. The set of nonempty strings from alphabet $\sum$ is denoted $\sum^+$. Thus, two appropriate equivalences are:

$\sum^+ = \sum^0 \cup \sum^1 \cup \sum^2 \cup$……..

$\sum^* = \sum^0 \cup \{\varepsilon\}$

vi) Concatenation of Strings

Let x and y be strings. Then **xy** denotes the concatenation of x and y, that is, the string formed by making a copy of x and following it by a copy of y. More precisely, if x is the string composed of i symbols x =$a_1a_2…..a_i$ and y is the string composed of j symbols y=$b_1b_2…..b_j$, then xy is the string of length i+j: xy= $a_1a_2…..a_i\ b_1b_2…..b_j$.

vii) Languages

A set of strings all of which are chosen from some $\sum^*$, where $\sum$ is a particular Alphabet, is called a language. If $\sum$ is an alphabet, and L then L is a language over $\sum$.

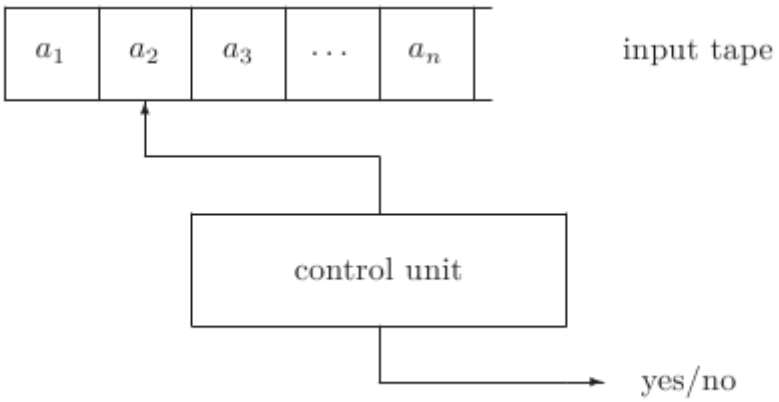
Examples:

1. The language of all strings consisting of n 0's followed by n 1's, for some n ≥ 0 :{ε, 01, 0011, 000111,……}
2. The set of strings of 0's and 1's with an equal number of each: {ε, 01, 10, 0011, 0101, 1001, …..}
3. The set of binary numbers whose value is a prime: {10, 11, 101, 111, 1011, …}
4. $\sum^*$ is a language for any alphabet $\sum$.

5. Φ, the empty language, is a language over any alphabet.
6. {ε}, the language consisting of only the empty string, is also a language over any alphabet. Note that Φ ≠ {ε}; the former has no strings and the latter has one string.

FINITE AUTOMATA

A finite automaton is a mathematical model which is used to study the abstract machines or abstract computing devices with inputs chosen from ∑, where ∑ stands for set of alphabets using which any string can be obtained. On reading the string, the machine may accept or reject the string.



There mainly 3 different types of FA,

1. Deterministic Finite Automata
2. Non-deterministic Finite Automata
3. Non-deterministic finite automata with ε-moves (ε-NFA)

1. Deterministic Finite Automata (DFA):

The term 'deterministic' refers to the fact that on each input there is one and only one state to which the automaton can transition from its current state.

Definition of a Deterministic Finite Automaton(2M):

A deterministic finite is defined by a quintuple as, A=(Q, ∑, δ, q0, F) where,

- Q is a finite set of states.
- ∑ is a finite non-empty set of symbols called input alphabet
- δ is a transition function that takes arguments a state and an input symbol and returns a state.

  If q is a state, and a is an input symbol, then δ(q,a) is that state p such that there is an arc labelled a from q to p.

  $$\delta : Q \times \sum \rightarrow Q$$
- q0 is the start state, one of the states in Q ic q0 ∈Q
- F is a set of final or accepting states. The set F is a subset of Q.

How a DFA Processes Strings: Initially the DFA is in state q0. The DFA reads the input string from left to right one symbol at a time. The changes to state of DFA is dictated by δ. When the last symbol of input string is read, if this results in the DFA reaching any of its final state, we say that the string is accepted by DFA, otherwise the string is rejected.

Simpler Notations for DFA's: There are two preferred notations for describing automata.

a) **A transition diagram**, which is a graph.

b) **A transition table**, which is a tabular listing of the function δ which by implication tells us the set of states and the input alphabet.

a) Transition Diagrams: A transition diagram for a DFA A=(Q, ∑, δ, q0, F) is a graph defined as follows:

- For each state in Q there is a node
- For each state q in Q and each input symbol a in ∑, let δ (q, a)= p. Then the transition diagram has an arc from node q to node p, labelled a. If there are several input symbols that cause transitions from q to p, then the transition diagram can have one arc, labeled by the list of these symbols.
- There is an arrow into the start state q_0 labeled *Start*. This arrow does not originate at any node.
- Nodes corresponding to accepting states (those in F) are marked by a double circle. States not in F have a single circle.

  Ex: If A=(Q, ∑, δ, q0, F) defined as A=({qo, q1, q2},{0,1}, δ,q0,{q1})

  where δ={ δ(qo,0)=q2,

δ(qo,1)=q0,

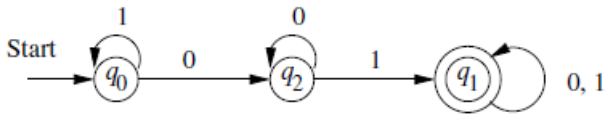
δ(q1,0)=q1,

δ(q1,1)=q1,

δ(q2,0)=q2,

δ(q2,1)=q1 }

The transition diagram is,



Transition Tables:

A transition table is a conventional, tabular representation of a function like δ that takes two arguments and returns a value. The rows of the table correspond to the states, and the columns correspond to the inputs. The entry for the row corresponding to state q and the column corresponding to input a is the state δ (q, a). The transition table for the above example is:

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

Extended Transition Function:

We define an extended transition function that describes what happens when we start in any state and follow any sequence of inputs. If δ is our transition function, then the extended transition function constructed from δ will be called $\hat{\delta}$.

"The extended transition function is a function that takes a state q and a string w and returns a state p - the state that the automaton reaches when starting in state q and processing the sequence of inputs w".

We define $\hat{\delta}$ by induction on the length of the input string, as follows:

BASIS: $\hat{\delta}(q, \epsilon) = q$. That is, if we are in state q and read no inputs, then we are still in state q.

INDUCTION: Suppose w is a string of the form xa; that is, a is the last symbol of w, and x is the string consisting of all but the last symbol.[3] For example, $w = 1101$ is broken into $x = 110$ and $a = 1$. Then

$$\hat{\delta}(q, w) = \delta\big(\hat{\delta}(q, x), a\big) \qquad (2.1)$$

Example 2.4: Let us design a DFA to accept the language

$$L = \{w \mid w \text{ has both an even number of 0's and an even number of 1's}\}$$

It should not be surprising that the job of the states of this DFA is to count both the number of 0's and the number of 1's, but count them modulo 2. That is, the state is used to remember whether the number of 0's seen so far is even or odd, and also to remember whether the number of 1's seen so far is even or odd. There are thus four states, which can be given the following interpretations:

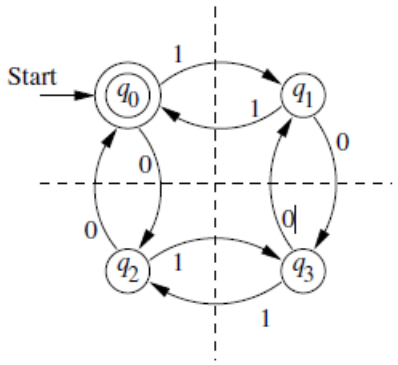
q_0: Both the number of 0's seen so far and the number of 1's seen so far are even.

q_1: The number of 0's seen so far is even, but the number of 1's seen so far is odd.

q_2: The number of 1's seen so far is even, but the number of 0's seen so far is odd.

q_3: Both the number of 0's seen so far and the number of 1's seen so far are odd.

State q_0 is both the start state and the lone accepting state. It is the start state, because before reading any inputs, the numbers of 0's and 1's seen so far are both zero, and zero is even. It is the only accepting state, because it describes exactly the condition for a sequence of 0's and 1's to be in language L.



Therefore DFA A=({q0, q1,q2,q3},{0,1}, δ, q0,{q0}} where δ is described by the transition diagram as shown above.

Suppose the input is 110101. Since this string has even numbers of 0's and1's both, we expect it is in the language. Thus, we expect that $\hat{\delta}$(q0,110101)=q0 since q0 is the only accepting state.

The check involves computing $\hat{\delta}(q_0, w)$ for each prefix w of 110101, starting at ϵ and going in increasing size. The summary of this calculation is:

- $\hat{\delta}(q_0, \epsilon) = q_0$.
- $\hat{\delta}(q_0, 1) = \delta\big(\hat{\delta}(q_0, \epsilon), 1\big) = \delta(q_0, 1) = q_1$.
- $\hat{\delta}(q_0, 11) = \delta\big(\hat{\delta}(q_0, 1), 1\big) = \delta(q_1, 1) = q_0$.
- $\hat{\delta}(q_0, 110) = \delta\big(\hat{\delta}(q_0, 11), 0\big) = \delta(q_0, 0) = q_2$.
- $\hat{\delta}(q_0, 1101) = \delta\big(\hat{\delta}(q_0, 110), 1\big) = \delta(q_2, 1) = q_3$.
- $\hat{\delta}(q_0, 11010) = \delta\big(\hat{\delta}(q_0, 1101), 0\big) = \delta(q_3, 0) = q_1$.
- $\hat{\delta}(q_0, 110101) = \delta\big(\hat{\delta}(q_0, 11010), 1\big) = \delta(q_1, 1) = q_0$.

The Language of a DFA:

Now, we can define the *language* of a DFA $A = (Q, \Sigma, \delta, q_0, F)$. This language is denoted $L(A)$, and is defined by

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \text{ is in } F\}$$

That is, the language of A is the set of strings w that take the start state q_0 to one of the accepting states. If L is $L(A)$ for some DFA A, then we say L is a *regular language*.

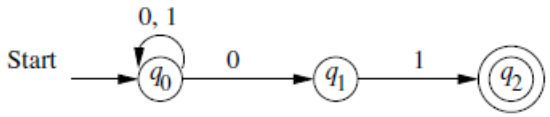
2.Nondeterministic Finite Automata (NFA):

A “nondeterministic finite automaton” (NFA) has the power to be in several states at once. This ability is often expressed as an ability to “guess” something about its input.

Like the DFA, an NFA has a finite set of states, a finite set of input symbols, one start state and a set of accepting states. It also has a transition function, which we shall commonly call δ. The difference between the DFA and the NFA is in the type of δ. For the NFA δ is a function that takes a state and input symbol as arguments but returns a set of zero, one, or more states, rather than returning exactly one state as the DFA must.

Ex: Let us design a NFA to accept all and only the strings of 0’s and 1’s that end in **01**.

Here the possible strings are, L={01, 101, 001, 11101,101001,..}



Here the NFA has the option of going either to q0 or to q1 and in fact it does both, as we shall see when we make the definitions precise. In state q1 the NFA checks that the next symbol is 1 and if so, it goes to state q2 and accepts. Notice that there is no arc out of q1 labeled 0, and there are no arcs at all out of q2.

Definition of Nondeterministic Finite Automata:

An NFA is represented as a quintuple,

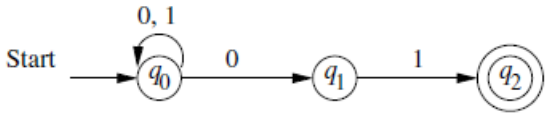
$$A = (Q, \Sigma, \delta, q_0, F)$$

where:

1. Q is a finite set of *states*.
2. Σ is a finite set of *input symbols*.
3. q_0, a member of Q, is the *start state*.
4. F, a subset of Q, is the set of *final* (or *accepting*) states.
5. δ, the *transition function* is a function that takes a state in Q and an input symbol in Σ as arguments and returns a subset of Q. Notice that the only difference between an NFA and a DFA is in the type of value that δ returns: a set of states in the case of an NFA and a single state in the case of a DFA.

$$\delta: Q \quad X \quad \sum \rightarrow \quad 2^Q$$

The transition table for given NFA can be represented as,



	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$*q_2$	$\emptyset$	$\emptyset$

The Extended Transition Function for NFA:

As for DFA's, we need to extend the transition function δ of an NFA to a function $\hat{\delta}$ that takes a state q and a string of input symbols w, and returns the set of states that the NFA is in if it starts in state q and processes the string w.

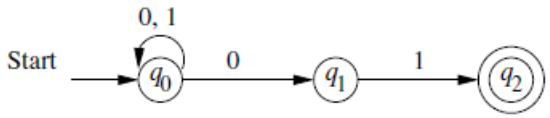
BASIS: $\hat{\delta}(q, \epsilon) = \{q\}$. That is, without reading any input symbols, we are only in the state we began in.

INDUCTION: Suppose w is of the form $w = xa$, where a is the final symbol of w and x is the rest of w. Also suppose that $\hat{\delta}(q, x) = \{p_1, p_2, \ldots, p_k\}$. Let

$$\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, \ldots, r_m\}$$

Then $\hat{\delta}(q, w) = \{r_1, r_2, \ldots, r_m\}$. Less formally, we compute $\hat{\delta}(q, w)$ by first computing $\hat{\delta}(q, x)$, and by then following any transition from any of these states that is labeled a.

Example: Let us trace the input 00101 by the NFA to accept strings ending with 01.



1. $\hat{\delta}(q_0, \epsilon) = \{q_0\}$.
2. $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$.
3. $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$.
4. $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.
5. $\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$.
6. $\hat{\delta}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.

Since we reached the end of the string and automata is in states {qo, q2} where q2 is the final state, the given string is accepted.

The Language of an NFA:

An NFA accepts a string w if it is possible to make any sequence of choices of next state, while reading the characters of w, and go from the start state to any accepting state.

Formally, if A=(Q, ∑, δ, q0, F), then,

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

That is, $L(A)$ is the set of strings w in Σ^* such that $\hat{\delta}(q_0, w)$ contains at least one accepting state.

Equivalence of Deterministic and Nondeterministic Finite Automata

- Although there are many languages for which an NFA is easier to construct than a DFA, it is a surprising fact that every language that can be described by some NFA can also be described by some DFA.
- Moreover, the DFA in practice has about as many states as the NFA, although it often has more transitions. In the worst case, however, the smallest DFA can have 2^n states while the smallest NFA for the same language has only n states.
- The proof that DFA's can do whatever NFA's can do involves an important "construction" called the **subset construction** because it involves constructing all subsets of the set of states of the NFA. In general, many proofs about automata involve constructing one automaton from another.

The subset construction starts from an NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Its goal is the description of a DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that $L(D) = L(N)$. Notice that the input alphabets of the two automata are the same, and the start state of D is the set containing only the start state of N. The other components of D are constructed as follows.

- Q_D is the set of subsets of Q_N; i.e., Q_D is the *power set* of Q_N. Note that if Q_N has n states, then Q_D will have 2^n states. Often, not all these states are accessible from the start state of Q_D. Inaccessible states can

be "thrown away," so effectively, the number of states of D may be much smaller than 2^n.

- F_D is the set of subsets S of Q_N such that $S \cap F_N \neq \emptyset$. That is, F_D is all sets of N's states that include at least one accepting state of N.
- For each set $S \subseteq Q_N$ and for each input symbol a in Σ,

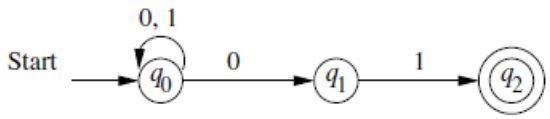
$$\delta_D(S, a) = \bigcup_{p \text{ in } S} \delta_N(p, a)$$

That is, to compute $\delta_D(S, a)$ we look at all the states p in S, see what states N goes to from p on input a, and take the union of all those states.

Example:

Let us convert the following NFA to DFA using subset construction method.

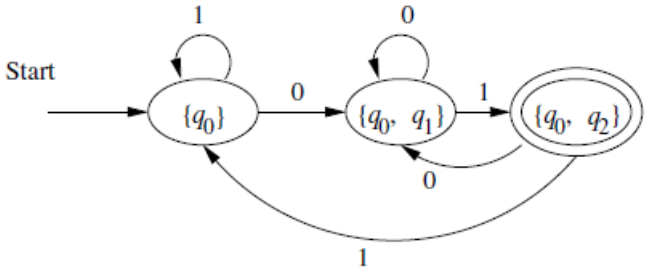
Given NFA,



Since N's set of states is {q0, q1, q2} the subset construction produces a DFA with 2^3 =8 states, corresponding to all the subsets of these three states. Following figure shows the transition table for these eight states.

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$*\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

The DFA constructed from the NFA is,



3.Finite Automata with Epsilon-Transitions:

An NFA that is allowed to make a transition spontaneously, without receiving an input symbol is called ε-NFA.

Ex: Let us design a ε-NFA that accepts **decimal numbers** consisting of:

1. An optional + or – sign
2. A string of digits
3. A decimal point and
4. Another string of digits. Either this string of digits or the string (2) can be empty, but both cannot be empty at the same time.

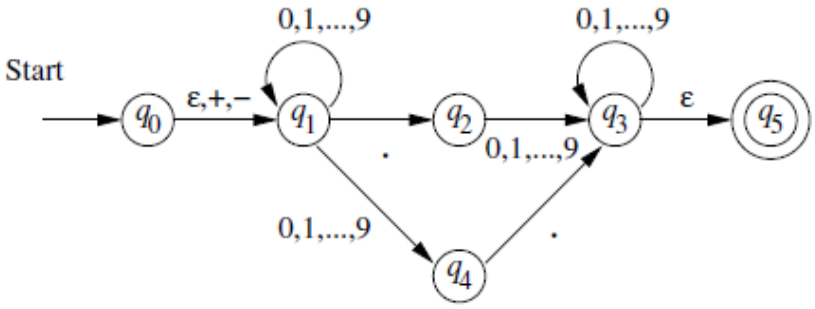


Figure: An ε-NFA accepting decimal numbers

The Formal Notation for an ε-NFA:

An ε-NFA may be represented exactly as we do an NFA with one exception: the transition function must include information about transition on ε.

Definition: Formally ε-NFA is defined by,

$$A=(Q, \sum, \delta, q0, F)$$

- Q is a finite set of states.
- ∑ is a finite non-empty set of symbols called input alphabet
- δ is a transition function that takes arguments a state and an input symbol or the symbol ε and returns zero or one or more states.

$$\delta : Q \times \sum \cup \{ \varepsilon \} \rightarrow 2^Q$$

- q0 is the start state, one of the states in Q ic q0∈Q
- F is a set of final or accepting states. The set F is a subset of Q.

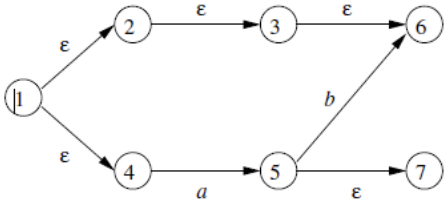
Epsilon-Closures:

The ε-closure of any state q denoted by ECLOSE(q) is the set of all states which are reachable from q on ε transitions only. Formally ε-closure ECLOSE(q) recursively, as follows:

BASIS: State q is in ECLOSE(q).

INDUCTION: If state p is in ECLOSE(q), and there is a transition from state p to state r labeled ϵ, then r is in ECLOSE(q). More precisely, if δ is the transition function of the ϵ-NFA involved, and p is in ECLOSE(q), then ECLOSE(q) also contains all the states in $\delta(p, \epsilon)$.

Example:



ECLOSE(1) = {1, 2, 3, 4, 6}

ECLOSE(2)={2, 3, 6}

ECLOSE(3)={3, 6}

ECLOSE(4)={4}

ECLOSE(5)={5,7}

Extended Transitions and Languages for ε-NFA's:

Suppose that $E = (Q, \Sigma, \delta, q_0, F)$ is an ϵ-NFA. We first define $\hat{\delta}$, the extended transition function, to reflect what happens on a sequence of inputs. The intent is that $\hat{\delta}(q, w)$ is the set of states that can be reached along a path whose labels, when concatenated, form the string w. As always, ϵ's along this path do not contribute to w. The appropriate recursive definition of $\hat{\delta}$ is:

BASIS: $\hat{\delta}(q, \epsilon) =$ ECLOSE(q). That is, if the label of the path is ϵ, then we can follow only ϵ-labeled arcs extending from state q; that is exactly what ECLOSE does.

INDUCTION: Suppose w is of the form xa, where a is the last symbol of w. Note that a is a member of Σ; it cannot be ϵ, which is not in Σ. We compute $\hat{\delta}(q, w)$ as follows:

1. Let $\{p_1, p_2, \ldots, p_k\}$ be $\hat{\delta}(q, x)$. That is, the p_i's are all and only the states that we can reach from q following a path labeled x. This path may end

with one or more transitions labeled ϵ, and may have other ϵ-transitions, as well.

2. Let $\bigcup_{i=1}^{k} \delta(p_i, a)$ be the set $\{r_1, r_2, \ldots, r_m\}$. That is, follow all transitions labeled a from states we can reach from q along paths labeled x. The r_j's are *some* of the states we can reach from q along paths labeled w. The additional states we can reach are found from the r_j's by following ϵ-labeled arcs in step (3), below.

3. Then $\hat{\delta}(q, w) = \text{ECLOSE}(\{r_1, r_2, \ldots, r_m\})$. This additional closure step includes all the paths from q labeled w, by considering the possibility that there are additional ϵ-labeled arcs that we can follow after making a transition on the final "real" symbol, a.

Language of an ε-NFA E=(Q, ∑,δ, qo, F) is defined as follows:

$$\boxed{L(E) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}}$$

Eliminating ε-transitions: Converting ϵ-NFA to DFA

Given any ϵ-NFA E, we can find a DFA D that accepts the same language as E. The construction we use is very close to the subset construction, as the states of D are subsets of the states of E. The only difference is that we must incorporate ϵ-transitions of E, which we do through the mechanism of the ϵ-closure.

Let $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$. Then the equivalent DFA

$$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$$

is defined as follows:

1. Q_D is the set of subsets of Q_E. More precisely, we shall find that all accessible states of D are *ϵ-closed* subsets of Q_E, that is, sets $S \subseteq Q_E$ such that $S = \text{ECLOSE}(S)$. Put another way, the ϵ-closed sets of states S are those such that any ϵ-transition out of one of the states in S leads to a state that is also in S. Note that $\emptyset$ is an ϵ-closed set.

2. $q_D = \text{ECLOSE}(q_0)$; that is, we get the start state of D by closing the set consisting of only the start state of E. Note that this rule differs from the original subset construction, where the start state of the constructed automaton was just the set containing the start state of the given NFA.

3. F_D is those sets of states that contain at least one accepting state of E. That is, $F_D = \{S \mid S \text{ is in } Q_D \text{ and } S \cap F_E \neq \emptyset\}$.

4. $\delta_D(S, a)$ is computed, for all a in Σ and sets S in Q_D by:

   (a) Let $S = \{p_1, p_2, \ldots, p_k\}$.

   (b) Compute $\bigcup_{i=1}^{k} \delta_E(p_i, a)$; let this set be $\{r_1, r_2, \ldots, r_m\}$.

   (c) Then $\delta_D(S, a) = \text{ECLOSE}(\{r_1, r_2, \ldots, r_m\})$.

Example: Convert the given NFA to DFA

	ϵ	a	b	c
$\rightarrow p$	$\{q, r\}$	$\emptyset$	$\{q\}$	$\{r\}$
q	$\emptyset$	$\{p\}$	$\{r\}$	$\{p, q\}$
$*r$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Ans:

Let us find **ECLOSE** of all the states of given NFA,

ECLOSE(p)={p, q, r}

ECLOSE(q)={q}

ECLOSE(r)={r}

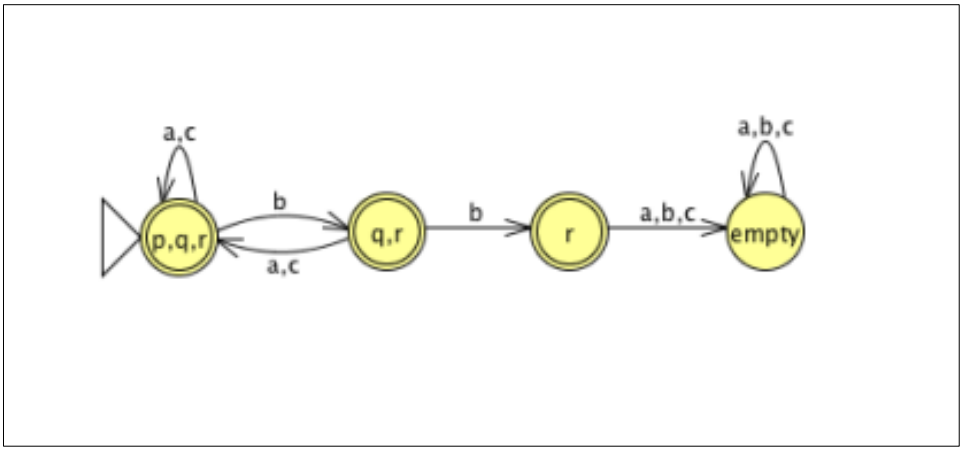
Step 1: Since p is the start state of the given NFA, start state of DFA is,

ECLOSE(p)={p, q, r}

Step 2: Find transition function of the DFA as follows (use subset-construction method) :

	a	b	c
$\rightarrow *\{p, q, r\}$	$\{p, q, r\}$	$\{q, r\}$	$\{p, q, r\}$
$*\{q, r\}$	$\{p, q, r\}$	$\{r\}$	$\{p, q, r\}$
$*\{r\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

The transition diagram of the DFA is,



An Application of Finite Automata: Text Search

1. Finding Strings in Text:

A common problem in the age of the Web and other online text repositories is the following: Given a set of words, and all documents that contain one (or all) of those words. A search engine is a popular example of this process.

The search engine uses a particular technology, called *inverted indexes*, where for each word appearing on the Web (there are 100,000,000 different words), a list of all the places where that word occurs is stored. Machines with very large amounts of main memory keep the most common of these lists available, allowing many people to search for documents at once.

Inverted-index techniques do not make use of finite automata, but they also take very large amounts of time for crawlers to copy the Web and set up the indexes. There are a number of related applications that are unsuited for inverted indexes, but are good applications for automaton-based techniques. The characteristics that make an application suitable for searches that use automata are:

1. The repository on which the search is conducted is rapidly changing. For example:
   (a) Every day, news analysts want to search the day's online news articles for relevant topics. For example, a financial analyst might search for certain stock ticker symbols or names of companies.
   (b) A "shopping robot" wants to search for the current prices charged for the items that its clients request. The robot will retrieve current catalog pages from the Web and then search those pages for words that suggest a price for a particular item.

2. The documents to be searched cannot be catalogued. For example, Amazon.com does not make it easy for crawlers to find all the pages for all the books that the company sells. Rather, these pages are generated "on the fly" in response to queries. However, we could send a query for books on a certain topic, say "finite automata" and then search the pages retrieved for certain words, e.g., "excellent" in a review portion.

2. Nondeterministic Finite Automata for Text Search:

Suppose we are given a set of words, which we shall call the keywords, and we want to find occurrences of any of these words. In applications such as these. A useful way to proceed is to design a nondeterministic finite automaton, which signals, by entering an accepting state, that it has seen one of the keywords.

The text of a document is fed, one character at a time to this NFA, which then recognizes occurrences of the keywords in this text. There is a simple form to an NFA that recognizes a set of keywords.

1. There is a start state with a transition to itself on every input symbol, e.g. every printable ASCII character if we are examining text. Intuitively, the start state represents a "guess" that we have not yet begun to see one of the keywords, even if we have seen some letters of one of these words.

2. For each keyword $a_1 a_2 \cdots a_k$, there are k states, say $q_1, q_2, \ldots, q_k$. There is a transition from the start state to q_1 on symbol a_1, a transition from q_1 to q_2 on symbol a_2, and so on. The state q_k is an accepting state and indicates that the keyword $a_1 a_2 \cdots a_k$ has been found.

Example 2.14: Suppose we want to design an NFA to recognize occurrences of the words `web` and `ebay`. The transition diagram for the NFA designed using the rules above is in Fig. 2.16. State 1 is the start state, and we use Σ to stand for the set of all printable ASCII characters. States 2 through 4 have the job of recognizing `web`, while states 5 through 8 recognize `ebay`. □

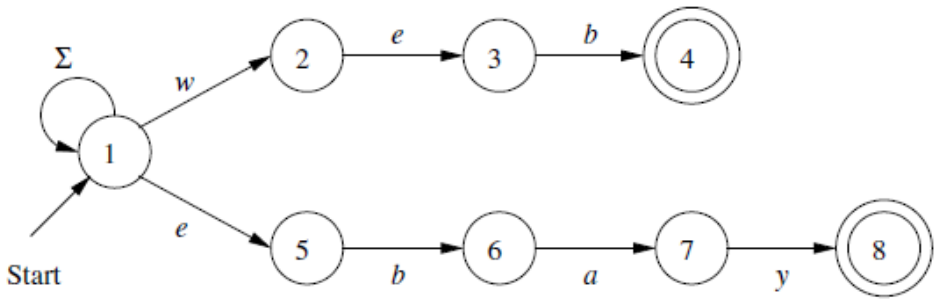


Figure 2.16: An NFA that searches for the words `web` and `ebay`

3. A DFA to Recognize a Set of Keywords:

The rules for constructing the set of DFA states is as follows:

a) If q_0 is the start state of the NFA, then $\{q_0\}$ is one of the states of the DFA.

b) Suppose p is one of the NFA states, and it is reached from the start state along a path whose symbols are $a_1 a_2 \cdots a_m$. Then one of the DFA states is the set of NFA states consisting of:

   1. q_0.
   2. p.
   3. Every other state of the NFA that is reachable from q_0 by following a path whose labels are a suffix of $a_1 a_2 \cdots a_m$, that is, any sequence of symbols of the form $a_j a_{j+1} \cdots a_m$.

The DFA to search keywords **web** and **ebay** is:

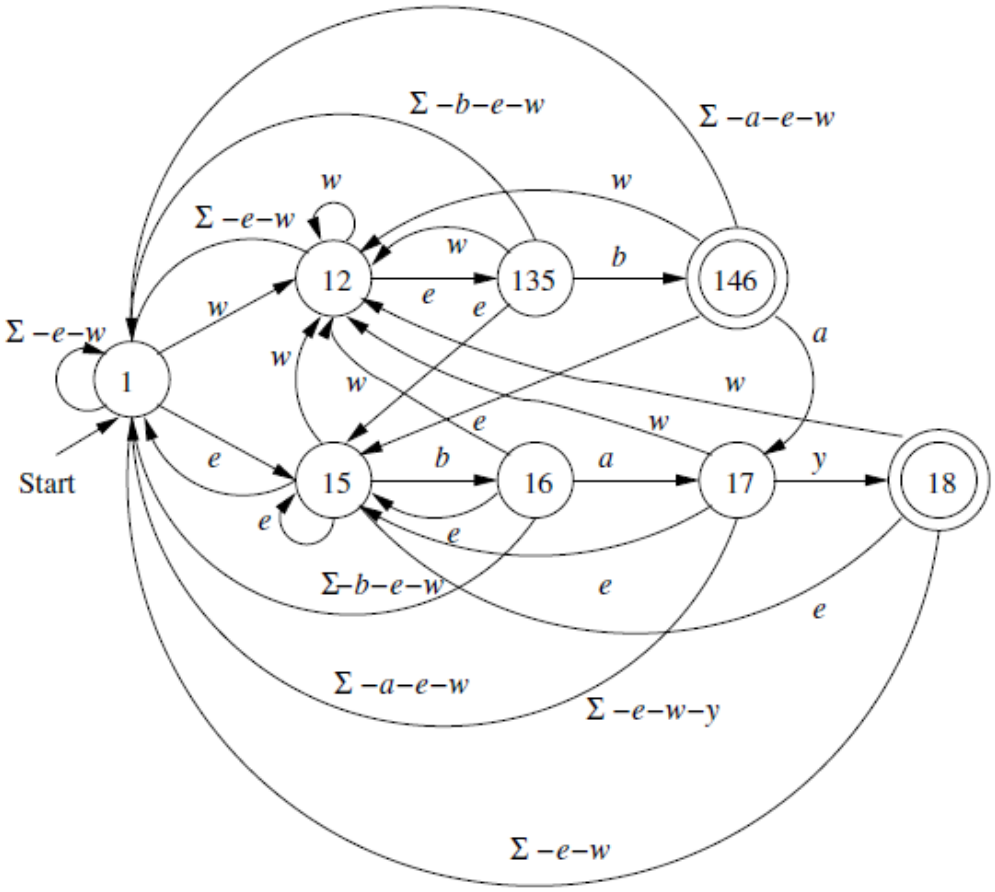


Figure 2.17: Conversion of the NFA from Fig. 2.16 to a DFA

Note: For exercises/problems under this module- refer class work.