St. Joseph's College of Engineering & Technology Palai

Department of Computer Science & Engineering

S4 CS

CS010 406 Theory of Computation

Module 1

Theory of Computation - Module 1

Brought to you by
http://nutlearners.blogspot.com

Syllabus

Proving techniques- Mathematical induction - Diagonalization principle - Pigeonhole principle - Functions - Primitive recursive and partial recursive functions - Computable and non computable functions - Formal representation of languages - Chomsky classification

Contents

Part I. Introduction

The subject Theory of Computation is a core part of Computer Science. The aim of this subject is

1. to familiarize students with the foundation and principles of computer science,
2. to teach material that is useful in subsequent courses (compiler design, algorithm analysis),
3. to strengthen students' ability to carry out formal and rigorous mathematical arguments.

This subject includes topics such as

automata theory,
languages,
grammars,
computability and
complexity.

These constitute the theoretical foundation of Computer Science.

The field of Computer Science includes a wide range of topics from machine design to programming. But all these have some common underlying principles. To study these principles, we construct abstract models of computers and computation.

The ideas we discuss in this subject have some immediate and important applications. For example, the fields of digital design, programming languages and compilers are the important examples. Also it has applications in operating systems to pattern recognition.

This subject provides many challenging, puzzle like problems that can lead to many sleepless nights.

1 Origin of Theory of Computation

Always man was interested in creating machines to ease human labour. In 16th century, Pascal invented a computing device. In 19th century, Charles Babbage built an analytical engine that performed computations without human intervention. In 1930s, mathematicians and philosophers thought of an ideal model which has features of computation as an intelligent activity.

A. M. Turing brought the idea of a machine called Turing Machine. This turing machine is a model of automatic computation. This led turing to propose a thesis called Turing thesis.

Turing thesis states that any computable function can be solved by a Turing machine. Then only in 1950, it was able to create a computer based on this model.

In 1943, Prof. McCullough brought the model of finite automata. [this will be learned in detail in Module 2]. finite automata has a large number of applications including lexical analysis in compilers, text editors (example: notepad, kwrite), special purpose hardware design, control unit design etc..

Prof. Noam Chomsky proposed a mathematical model to specify the syntax of natural languages. This resulted in formal language theory. A grammar called context free grammar was proposed for programming languages.

Languages, Grammars and automata form three fundamental ideas of theory of Computation.

Part II. Proving Techniques

[Pandey2006]

An important requirement for learning this subject is the ability to follow proofs. There are three fundamental techniques to proof. They are,

1. Principle of mathematical induction,
2. Pigeonhole principle,
3. Diagonalization principle.

2 Principle of Mathematical Induction

Let we want to show that property P holds for all natural numbers. To prove this property, P using mathematical induction, following are the steps:

Basic Step:

First show that property P is true for 0 or 1.

Induction Hypotheis:

Assume that property P holds for n.

Induction step:

Using induction hypothesis, show that P is true for n+1.

Then by the principle of mathematical induction, P is true for all natural numbers.

Examples

Example 1:

Prove using mathematical induction, $n^4 - 4n^2$ is divisible by 3 for n>=0.

Basic step:

For n=0,

$n^4 - 4n^2$ =0, which is divisible by 3.

Induction hypothesis:

Let $n^4 - 4n^2$ is divisible by 3.

Induction step:

$(n+1)^4 - 4(n+1)^2$

$= [(n+1)^2]^2 - 4(n+1)^2$

$= (n^2+2n+1)^2 - (2n+2)^2$

$= (n^2+2n+1+2n+2)(n^2+2n+1-2n-2)$

$= (n^2+4n+3)(n^2-1)$

$= n^4+4n^3+3n^2-3-4n-n^2$

$= n^4+4n^3+2n^2-4n-3$

$= n^4+4n^3-4n^2+6n^2-4n-3$

$= n - 4n^2 + 6n^2 - 3 + 4n^3 - 4n$

$= (n^2 - 4n^2) + (6n^2) - (3) + 4(n^3 - n)$

$(n^2 - 4n^2)$ is divisible by 3 from our hypothesis.

$6n^2, 3$ are divisible by 3.

We need to prove that $4(n^3 - n)$ is divisible by 3.

Again use mathematical induction.

Basic step:

For n = 0,

4(0-0) = 0 is divisible by 3.

Induction hypothesis:

Let $4(n^3 - n)$ is divisible by 3.

Induction step:

$4[(n+1)^3 - (n+1)]$

$= 4[(n^3 + 3n^2 + 3n + 1) - (n+1)]$

$= 4[n^3 + 3n^2 + 3n + 1 - n - 1]$

$= 4[n^3 + 3n^2 + 2n]$

$= 4[n^3 - n + 3n^2 + 3n]$

$= 4(n^3 - n) + 4.3n^2 + 4.3n$

$4(n^3 - n)$ is divisible by 3 from our hypothesis.

$4.3n^2$ is divisible by 3.

$4.3n$ is divisible by 3.

Thus we can say that

$= (n^2 - 4n^2) + (6n^2) - (3) + 4(n^3 - n)$ is divisible by 3.

That is,

$n^4 - 4n^2$ is divisible by 3.

Example 2:

Prove using mathematical induction:

$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$

Let $P(n) = 1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$

Basic Step:

For n=1,

LHS = 1

RHS = 1(1+1)/2=1

Induction hypothesis;

Assume that P(n) is true for n=k,

Then,

$1 + 2 + 3 + ... + k = \frac{k(k+1)}{2}$

Induction step:

Show that P(n) is true for n=k+1.

$1 + 2 + 3 + ... + k + (k + 1)$

$= \frac{k(k+1)}{2} + k + 1$

$= (k + 1)[\frac{k}{2} + 1]$

$= (k + 1)(\frac{k+2}{2})$

$= \frac{(k+1)(k+2)}{2}$

That is, P(n) is true for n=k+1.

Thus by using the principle of mathematical induction, we proved

$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$

Example 3:

Prove using the principle of mathematical induction,

$\sum_{i=0}^{n} n^2 = \frac{n(n+1)(2n+1)}{6}$

Let $P(n) = \sum_{i=0}^{n} n^2 = \frac{n(n+1)(2n+1)}{6}$

Basic step:

For n=1,

LHS = $1^2 = 1$

RHS = $\frac{n(n+1)(2n+1)}{6} = \frac{1.2.3}{6} = 1$

P(n) is true for n=1.

Induction hypothesis:

Assume that result is true for n=k.

That is,

$1^2 + 2^2 + ... + k^2 = \frac{k(k+1)(2k+1)}{6}$

Induction step:

Prove that result is true for n=k+1.

$1^2 + 2^2 + ... + k^2 + (k + 1)^2 = \frac{k(k+1)(2k+1)}{6} + (k + 1)^2$

$= (k + 1)[\frac{k(2k+1)}{6} + (k + 1)]$

$= (k + 1)(\frac{2k^2+k+6k+6}{6})$

$= (k + 1)(\frac{2k^2+7k+6}{6})$

$= (k + 1)(\frac{2k^2+4k+3k+6}{6})$

$= (k + 1)(\frac{2k(k+2)+3(k+2)}{6})$

$= \frac{(k+1)(k+2)(2k+3)}{6}$

Thus it is proved.

Exercise:

1. Prove the following by principle of induction:

a. $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$

b. $\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$

c. 2i>i for all i>1.

d. $1 + 4 + 7 + ... + (3n - 2) = \frac{n(3n-1)}{2}$

e. $2^x \geq x^2$,if $x \geq 4$

f. $10^{2n} - 1$ is divisible by 11 for all n>1

g. $2^n > n$, for all n>1.

3 Diagonalization Principle

Let s be a non empty set and R any relation on S.

Let

$D = \{a \epsilon A | (a, a) \notin r\}$

For each $a \notin A$, let $R_a = \{b | (a, b) \in R\}$

Then diagonalization principle states that D is different from each R_a.

OR

Diagonalization principle states that the complement of the diagonal is different from each row.

For example,

Let S = {a, b, c, d}

R = { (a,a), (b,c), (b,d), (c,a), (c,c), (c,d), (d,a), (d,b) }

The above relation R is shown in matrix from as follows:

	a	b	c	d
a	X			
b			X	X
c	X		X	X
d	X	X		

Diagonal elements are marked.

From the figure, $R_a = \{a\}$

$R_b = \{c, d\}$

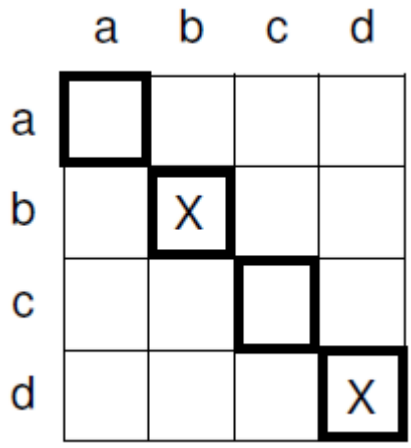
$R_c = \{a, c, d\}$

$R_d = \{a, b\}$

Complement of the diagonal is,

D = {b, d}

That is,



If we compare each of the above R_a, R_b, R_c, R_d with D, we can see that D is different from each R_a. Thus complement of the diagonal is distinct from each row.

Note:

Following information is needed to solve the example given below:

9's complement of a number

9's complement of 276 is 723.

9's complement of 425 is 574.

9's complement of 793 is 206.

Example 1:

Prove that the set of real numbers between 0 and 1 is uncountable.

An example for a real number between 0 and 1 is .34276

Let we represent a real number between 0 and 1 as

$$x = .x_0x_1x_2x_3.....$$

where each x_i is a decimal digit.

Let f(k) be an arbitrary function from natural numbers to the set [0,1].

We can arrange the elements in a 2d array as,

f(0): . x_{00} x_{01} x_{02} x_{03} _ _ _

f(1): . x_{10} x_{11} x_{12} x_{13} _ _ _

f(2): . x_{20} x_{21} x_{22} x_{23} _ _ _

_

_

f(n): . x_{n0} x_{n1} x_{n2} x_{n3} _ _ _

where x_{ni} is the ith digit in the decimal expansion of f(n).

Next we find the complement of the diagonal (9's complement) as follows:

$$Y = .y_0y_1y_2.....$$

where y_i =9's complement of x_{ii}

[Find out the 9's complement of $x_{00}, x_{11}, x_{22}, x_{33}.....x_{nn}$]

From the diagonalisation principle, it is clear that the complement of the diagonal is different from each row.

Here it is clear that Y is different from each f(i) in at least one digit. $Y \neq f(i)$. Hence Y cannot be present in the above array.

This means that the set of real numbers between 0 and 1 is countably infinite or not countable.

For instance suppose we arrange the real numbers as,

f(0):	.	9	4	2	4	_	_	_
f(1):	.	6	3	6	2	_	_	_
f(2):	.	2	8	6	4	_	_	_
f(3)	.	6	5	3	2	_	_	_
_	.	_	_	_	_	_	_	_
f(n):	.	4	1	5	7	_	_	_

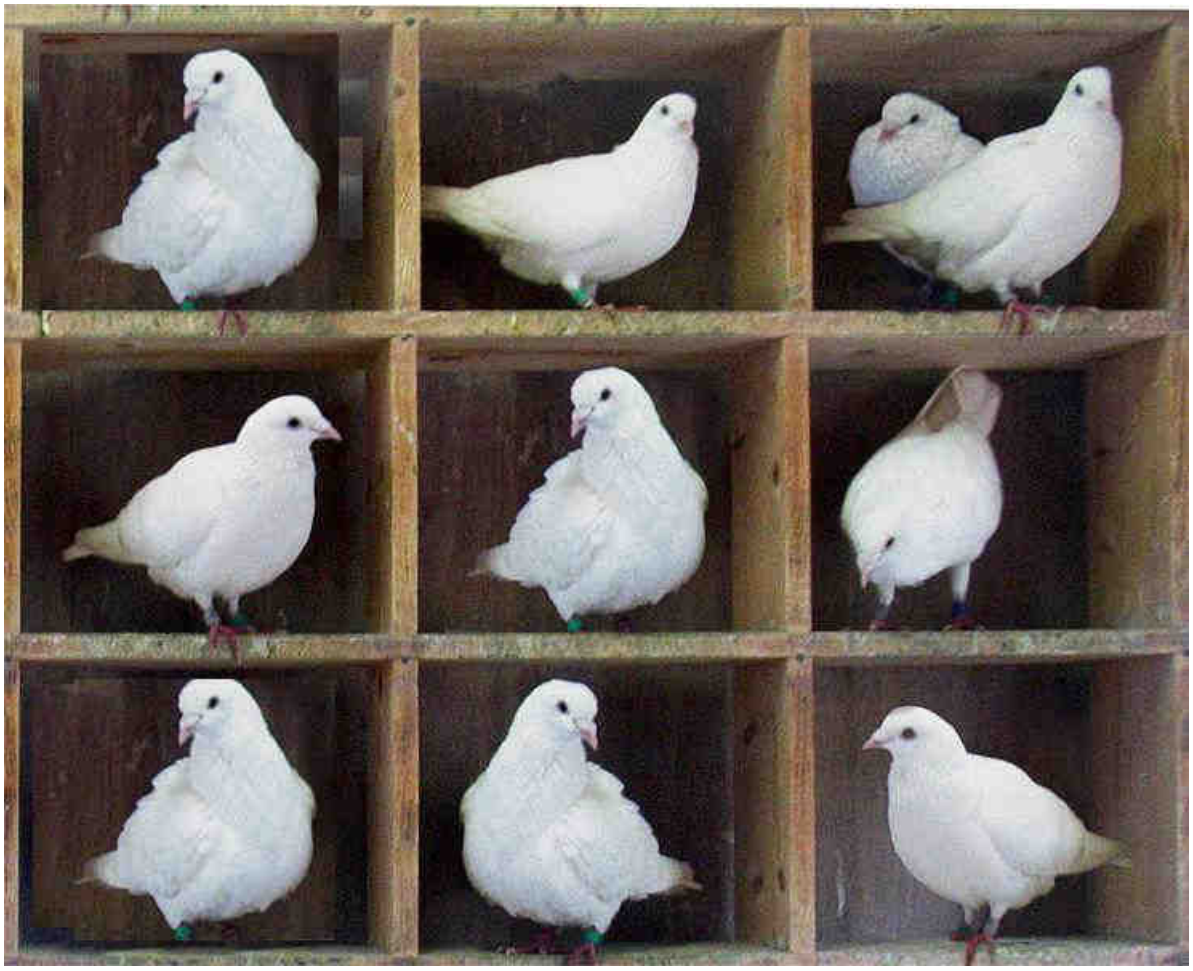
Here the diagonal is 9 3 6 2.....

The complement (9's complement) of the diagonal is 0 6 3 7.....

The real number .0637... is not in the above table.

Thus It is clear that this value is distinct from each row, f(0), f(1), f(2)...f(n).

4 Pigeonhole Principle



A pigeonhole is a hole for pigeons to nest.

From the above diagram, it is clear that if 10 pigeons are put into 9 pigeonholes, then one pigeonhole must contain

more than one item.

The pigeonhole principle states that if n pigeons are put into m pigeonholes with n>m, then at least one pigeonhole must contain more than one pigeon.

Thus if S1 and S2 are two non empty finite sets and $|S1| > |S2|$, then there is no one-to-one function from S1 to S2.

Pigeonhole principle can be used to show that certain languages are not regular. We will use pigeonhole principle in the topic pumping lemma later.

Part III. Recursive Function Theory

Functions

The concept of a function is a fundamantal topic in mathematics.

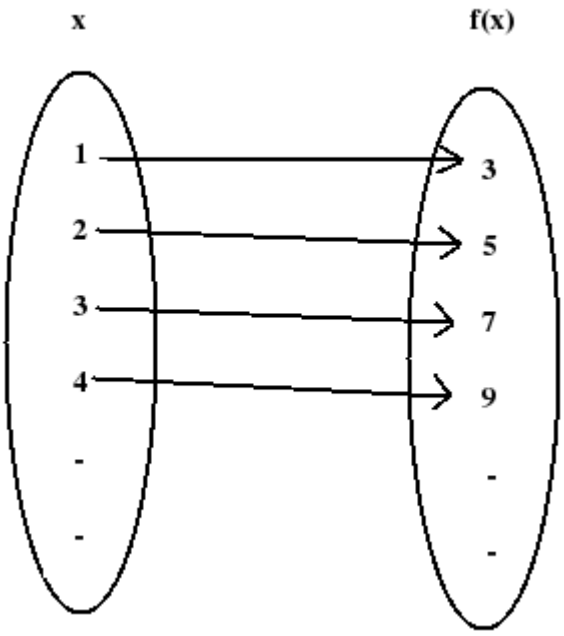
A function or a total function $f : X \longrightarrow Y$ is a rule that assigns to all the elements of one set, X, a unique element of another set, Y.

The first set, X is called the domain of the function, and the second set, Y is called its range.

For example,

$$f(x) = 2x + 1$$

is a function defined on one variable, x. Imagine we say that f is over all natural numbers only. Then the following diagram shows the function:



Here the domain of f is {1, 2, 3, 4, ...}

Range of f is {3, 5, 7, 9, ...}.

$$f(x, y) = x^2 + 2y$$

is a function on 2 variables, x and y.

$$f(x, y, z) = x + 2y^3 + 7z$$

is a function on 3 variables, x, y and z.

Total Function

The above three examples were total functions. A total function from X to Y is a rule that assigns to every element of X, a unique element of Y.

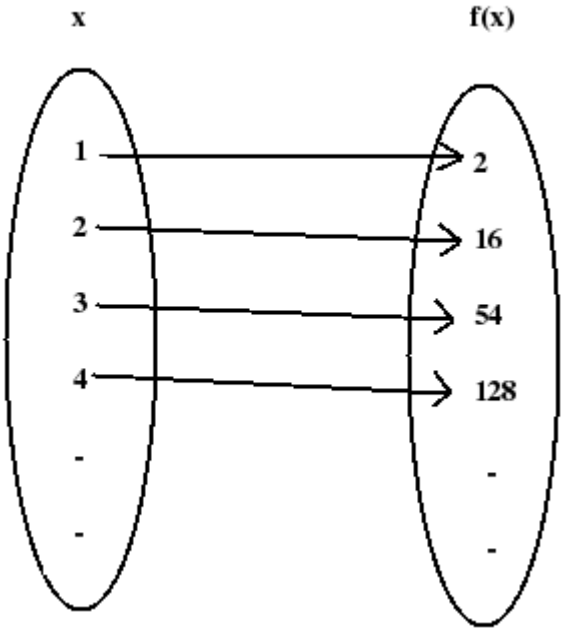
$$f : X \longrightarrow Y$$

For example,

$$f(x) = 2x^3$$

In all our discussions, we assume that f is over all natural numbers,

then



Here the domain of f is {1, 2, 3, 4 .....}.

That is, the domain contains all the elements of x. So f is a total function.

Partial Function

Here, to extend the class of computable functions, we use partial functions.

A partial function,

$$f : X \longrightarrow Y$$

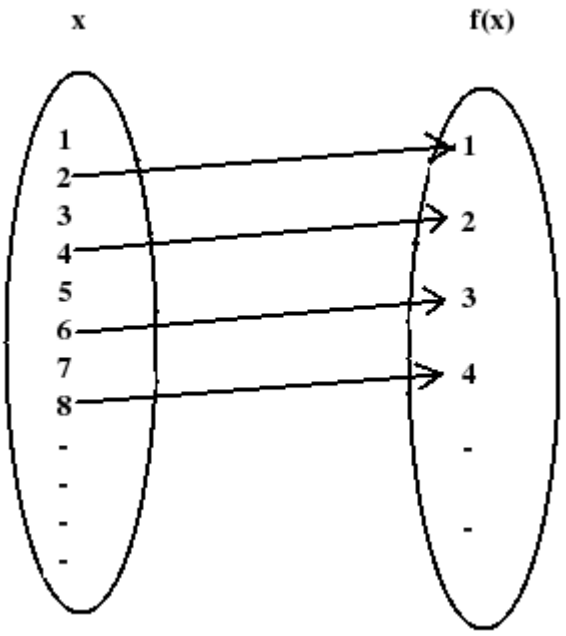
is a rule that assigns to some elements of X, a unique element of Y. For other elements, there may not be any corresponding element in Y.

For example,

Example 1:

$$f(x) = \frac{x}{2}$$

Suppose f is defined over natural numbers. Then f is a partial function. this can be seen from the following diagram.



In the above diagram, domain of f is {2, 4, 6, 8, ...}

Here the domain is not equal to the set X. So f is a partial function.

Example 2:

$$f(x, y) = x - y$$

f is defined over N.

f is a partial function.

This is because when x=6, y=8, f(x,y) = -2

-2 is not a natural number. So f is not defined for x=6, y=8. f is a partial function.

Recursive Functions

Consider the example,

$$f(n) = n^2 + 1$$

Here, given any value for n, multiply that value by itself and then add one. Here function is defined in a recursive way. Value of f can be computed in a mechanical fashion.

Example 2:

Factorial of a number is defined as,

n! = n (n-1) (n-2) . . . . . 1.

This is an explicit definition.

Another way to define factorial is,

0! = 1

n! = n (n-1)!

for $n \epsilon N$

This definition is recursive because to find the factorial at an argument n, we need to find the factorial at some simpler argument (n-1).

Example 3:

Exponentiation can be defined as,

$x^n = x.x.x.x.....x$ (n times)

This exponentiation function can be recursively defined as,

$x^0 = 1$

$x^n = x.x^{n-1}$

for $n \epsilon N$.

This is a recursive definition for exponentiation.

In all our discussions, a function, f is defined over natural numbers (N) only.

5 Primitive Recursive Functions

A function, f is called a primitive recursive function,

i) If it is one of the three basic functions, or,

ii) If it can be obtained by applying operations such as composition and recursion to the set of basic functions.

The basic functions and operations are explained below;

5.1 Basic Functions

We define three basic functions. They are,

Zero function,

Successor function, and

Projector function.

Zero Function

Z(x) = 0

is called Zero function.

Example:

We may write

Z(8) = 0.

Successor Function

The successor function is S(x), defined as

S(x) = x+1

Thus the value of S(x) is the integer next in sequence to x.

For example,

S(4) = 5

S(29) = 30

Projector Function

Projector function, P_k is defined as,

$$P_k(x_1, x_2, ...x_k, ...x_n) = x_k$$

For example,

$$P_3(8, 5, 6, 4) = 6$$

$$P_6(6, 34, 7, 2, 45, 23, 22) = 23$$

5.2 Operations

We can build complicated functions from the above basic functions by performing operations such as,

composition, and

recursion.

Composition

If functions, f_1, f_2 and g are given, then the composition of g with f_1, f_2 is given by,

$$h = g(f_1, f_2).$$

In general,

If functions $f_1, f_2.....f_k$ and g are given, then the composition of g with $f_1, f_2.....f_k$ is

$$h = g(f_1, f_2, f_3.....f_k)$$

Example 1:

Let

$$g(n) = n^2$$

$$h(n) = n + 3$$

Find the composition of h with g.

The composition of h with g is,

$$\begin{aligned} f(n) &= h[g(n)] \\ &= h[n^2] \\ &= n^2 + 3 \end{aligned}$$

Example 2:

Let

$g(n) = n^2$

$h(n) = n + 3$

Find the composition of g with h.

$$\begin{aligned} f(n) &= g[h(n)] \\ &= g[n+3] \\ &= (n+3)^2 \end{aligned}$$

Example 3:

Let

$f_1(x, y) = x + y$

$f_2(x, y) = 2x$

$f_3(x, y) = xy$, and

$g(x, y, z) = x + y + z$

be functions over N.

Find the composition of g with f_1, f_2, f_3.

$$\begin{aligned} h(x, y) &= g(f_1, f_2, f_3) \\ &= g\,(x + y,\ 2x,\ xy) \\ &= x + y + 2x + xy \\ &= 3x + y + xy \end{aligned}$$

Recursion

A function f can be constructed using recursion from the functions g and h as,

$f(x,\ 0) = g(x)$

$f(x,\ y + 1) = h\,[x,\ y,\ f(x, y)]$

In the above, f is of two variables.

g is of one variable and h is of 3 variables.

In general,

a function of n+1 variables is defined by recursion if there exists a function g of n variables and a function h of n+2 variables.

f is defined as,

$f(x_1, x_2, ...x_n, 0) = g(x_1, x_2, ...x_n)$

$f(x_1, x_2, ...x_n,\ y + 1) = h\,[x_1, x_2, ...x_n,\ y,\ f(x_1, x_2, ...x_n, y)\,]$

Example 1:

Addition of integers can be defined using recursion as,

$add(x, 0) = x$

$add(x, y + 1) = add(x, y) + 1$

Example 2:

The function multiplication can be defined using recursion operation as,

$prod(x, 0) = 0$

$prod(x, y + 1) = add\,[x,\, prod(x, y)\,]$

Above examples show how recursion operation is applied to a set of functions.

Primitive Recursive functions

Now we learned basic functions such as zero function, successor function and projector function, and operations such as composition and recursion.

Again, a function, f is a primitive recursive function if either,

i. it is one of the basic functions, or

ii. it is produced by performing operations such as composition and recursion to the basic functions.

Example 1:

The addition function, add(x,y) is a primitive recursive function because,

$$\begin{aligned} add(x, 0) &= x \\ &= P_1(x) \\ add(x, y + 1) &= add(x, y) + 1 \\ &= S[add(x, y)] \\ &= S[P_3(x, y, add(x, y))] \end{aligned}$$

Thus add(x, y) is a function produced by applying composition and recursion to basic functions, P_k and S.

Example 2:

The multiplication function mult(x, y) is a primitive recursive function because,

$$\begin{aligned} mult(x, 0) &= 0 \\ &= Z(x) \\ mult(x, y + 1) &= add\,[x,\, mult(x, y)\,] \\ &= add[P_1(x, y, mult(x, y)), P_3(x, y, mult(x, y))] \end{aligned}$$

From the previous example, add() is a primitve recursive function.

From the above, mult(x,y) is produced by performing operations on basic functions and add() function.

So mult(x,y) is a primitive recursive function.

Example 3:

The factorial function,

f(n) = n!

is a primitive recursive function. This can be proved using mathematical induction.

Basic Step:

For n = 0,

f(0) = 0! = 1

= $P_1(1)$

Induction Hypothesis:

Assume that f(p) is a primitive recursive function.

Induction Step:

f(p+1) = (p+1)!

= p! (p+1)

= f(p) (p+1)

= mult [f(p), p+1]

= mult [f(p), S(p)]

In the above, f(p) is primitive recursive from induction hypothesis.

S(p) is primitive recursive, since it is the successor function.

mult() is primitve recursive from the previous example.

So we can conclude that

f(n) = n!

is primitive recursive.

Example 4:

Show that function $f_1(x, y) = x + y$ is primitive recursive.

$$f_1(x, 0) = x + 0$$
$$= x$$
$$= P_1(x)$$
$$f_1(x, y + 1) = x + (y + 1)$$
$$= (x + y) + 1$$
$$= f_1(x, y) + 1$$
$$= S[f_1(x, y)]$$
$$= S[P_3(x, y, f_1(x, y))]$$

Since $f_1(x, 0)$ and $f_1(x, y + 1)$ are primitive recursive, $f_1(x, y) = x + y$ is primitive recursive.

Example 5:

Show that the function, $f_1(x, y) = x * y$ is primitive recursive.

$$f_2(x, 0) = x * 0$$
$$= 0$$
$$= Z(x)$$
$$f_1(x, y + 1) = x * (y + 1)$$
$$= (x * y) + x$$
$$= f_2(x, y) + x$$

$= f_1[f_2(x,y), x]$ from Example 4.

$= f_1[P_3(x,y,f_2(x,y)),\ P_1(x,y,f_2(x,y))]$

Since $f_2(x,0)$ and $f_2(x,y+1)$ are primitive recursive, $f_2(x,y) = x * y$ is primitive recursive.

Example 6:

Show that the function, $f(x,y) = x^y$ is a primitive recursive function.

$$
\begin{aligned}
f(x,0) &= x^0 \\
&= 1 \\
&= P_1(1) \\
f(x,y+1) &= x^{y+1} \\
&= x^y.x^1 \\
&= x.x^y \\
&= x.f(x,y) \\
&= P_1(x,y,f(x,y)) * P_3(x,y,f(x,y)) \\
&= mult\ [P_1(x,y,f(x,y)),\ P_3(x,y,f(x,y))\ ]
\end{aligned}
$$

mult() is a primitive recursive function as we found earlier.

Since $f(x,0)$ and $f(x,y+1)$ are primitive recursive, $f(x,y) = x^y$ is primitive recursive.

6 Partial Recursive Functions

A function, f is a partial recursive function if either,

i. it is one of the basic functions, or

ii. it is produced by performing operations such as composition, recursion and minimization to the basic functions.

Basic Functions

We learned three basic functions such as,

Zero function, Z(x),

Successor function, S(x), and

Projector function, P_k.

Operations

We learned the operations composition and recursion earllier.

A new operation now comes, minimization.

Minimization

μ is the minimization operator.

Let a function g(x,y) is defined over two variables, x and y.

Then minimization operator, μ is defined over g (x, y) is as follows:

$\mu_y[g(x,y)]$ =smallest y such that g (x, y) = 0.

Using the minimization operation, a function f(x) can be defined for some y;

we can write, f(x) = $\mu_y[g(x,y)]$

Example 1:

Let g(x,y) is given in the following table:

g(0,0)=5	g(1,0)=5	g(2,0)=8	g(3,0)=1
g(0,1)=4	g(1,1)=6	g(2,1)=5	g(3,1)=2
g(0,2)=6	g(1,2)=0	g(2,2)=undefined	g(3,2)=0
g(0,3)=0	g(1,3)=3	g(2,3)=0	g(3,3)=4
g(0,4)=1	g(1,4)=0	g(2,4)=7	g(3,4)=undefined

Then,

f(0) = 3,

[because g(0,3) = 0]

Minimization operation is denoted by $\mu[g(0,3)=0]=3$

f(1) = 2

[2 is the minimum y that g(1,2)=0; Though g(1,4)=0 also;]

Minimization operation is denoted by $\mu[g(1,2)=0]=2$

f(2) = Not defined

[because g(2,3)=0, but g(2,2) is undefined, where y=2 < y=3]

f(3) = 2

[because g(3,2)=0, though g(3,4) is undefined for y=4, but 4<2]

Minimization operation is denoted by $\mu[g(3,2)=0]=2$

For some values g(x,y) is undefined, but we can define minimization operation.

Partial Recursive Functions

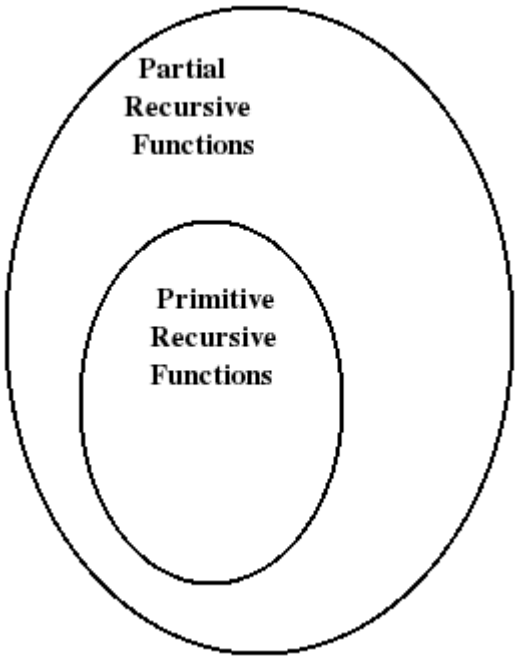
A function, f is a partial recursive function if either,

i. it is one of the basic functions, or

ii. it is produced by performing operations such as composition, recursion and minimization to the basic functions.

From the definition, we can say that,

primitive recursive functions are a subset of partial recursive functions. This is shown below:



Thus all primitive recursive functions are partial recursive functions.

Note that all partial recursive functions are computable functions.

Example 1:

Show that $f(x) = \frac{x}{2}$ is a partial recursive function over N.

$f(x) = \frac{x}{2}$

We may write

$y = \frac{x}{2}$

Then,

$x = 2y$

$2y - x = 0$

Let

$g(x, y) = |2y - x|$

Let

$g(x, y) = 0,$ for some x and y.

Let

$f_1(x) = \mu[|2y - x| = 0]$

Only for even values of x , $f_1(x)$ is defined. When x is odd, $f_1(x)$ is not defined. So f_1 is partial recursive.

Since, $f(x) = \frac{x}{2} = f_1(x)$, f is a partial recursive function.

Part IV. Computable and Non-Computable Functions

7 Computable Functions

Computable functions are the basic objects of study in Theory of Computation. Computable functions are the functions that can be calculated using a mechanical calculation device given unlimited time and memory space. Also any function that has an algorithm is computable.

Each computable function f takes a fixed finite number of natural numbers as arguments. The computable function f returns an output for some inputs.

For some inputs, it may not return an output. Due to this, a computable function is a paertial recursive function.

If the computable function is defined for all possible arguments, it is called a total computable function or total recursive function.

The notation $f(x_1, x_2, ...x_k)$ indicates that the partial function f is defined on arguments $x_1, x_2, ...x_k$.

We say that a function f on a certain domain is said to be computable, if there exists a Turing machine that computes the value of f for all arguments in its domain. A function is uncomputable if no such Turing machine exists.

The basic characteristic of a computable function is that there is a finite procedure or algorithm telling how to compute the function.

The characterisitics of a computable function are as follows:

1. There must be exact instructions (ie, a program, finite in length).

2. If the procedure is given a k tuple x in the domain of f, then after a finite number of steps, the procedure must terminate and produce f(x).

3. If the procedure is given a k-tuple x which is not in the domain of f, then the procedure must never halt, or it may get stuck at some point.

Following are some examples for computable functions:

1. Each function with a finite domain.

eg. Any finite sequence of natural numbers.

2. Each constant function $f : N^k \longrightarrow N$, $f(n_1, n_{2,}.....n_k) = n$.

3. Addition $f : N^2 \longrightarrow N$, $f(n_1, n_2) = n_1 + n_2$.

4. The function which gives the list of prime factors of a number.

5. The gcd of two numbers is a computable function.

8 Non-Computable Functions

The rela numbers are uncountable so most real numbers are not computable. Most subsets of natural numbers are not countable.

A non-computable function corresponds to an undecidable problem and it consists of a family of instances for which there is no computer program that given any problem instance as input terminates and outputs the required answer after a finite number of steps.

For an undecidable problem, it is impossible to construct a single algorithm that always leads to a correct yes/ no answer.

We say that a function f on a certain domain is said to be non-computable, if no Turing machine exists that computes the value of f for all arguments in its domain.

Examples for undecidable problems (non-computabel functions) are halting problem, post correspondence problem.

Part V. Formal Representation of Languages

[Kavitha2004]

Here we introduce the concept of formal languages and grammars.

We know that a grammar is specified for a natural language such as English. To define valid sentences and to give structural descriptions of sentences a grammar is used. Linguistics is a branch to study the theory of languages. Scientists in this branch of study have defined a formal grammar to describe English that would make language translation using computers very easy.

English Grammar

First we will consider how grammars can be specified for English language. Then we will extend this model to our programming language grammar specification.

An example for grammar is,

ARTICLE $\longrightarrow$ a | an | the

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

NOUN $\longrightarrow$ boy | apple | ε

Productions

The above are called productions or rules of the grammar. Two productions are given above.

First production is,

ARTICLE $\longrightarrow$ a | an | the

It can also be written as,

ARTICLE $\longrightarrow$ a

ARTICLE $\longrightarrow$ an

ARTICLE $\longrightarrow$ the

Second production is,

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

Third production is,

NOUN $\longrightarrow$ boy | apple | ε

It can also be written as,

NOUN $\longrightarrow$ boy

NOUN $\longrightarrow$ apple

NOUN $\longrightarrow$ ε

Non-terminals

In the above, ARTICLE, NOUN_PHRASE, NOUN are called non-terminal symbols. A non-terminal is specified using upper case letters.

Note that the left hand side of a production always contains a single non-terminal.

Terminals

In the above, 'a', 'an', 'the', 'boy', 'apple' are called terminal symbols. A terminal is written using lowercase letters.

ε means an empty string.

Start Symbol

In a grammar specification, one non-terminal symbol is called the start symbol. Here, NOUN_PHRASE is the start symbol.

Thus a grammar consists of a set of productions, terminals, non-terminals and start symbol.

If the matching left hand side of a certain rule can be found to occur in a string, it may be replaced by the corresponding right hand side.

8.1 Formal Definition of a Grammar

A grammar is $(V_N, \sum, P, S)$ where

V_N is a finite non empty set whose elements are called non-terminals.

$\sum$ is a finite non empty set whose elements are called terminals.

$V_N \cap \sum = \phi$

S is a special non terminal called start symbol.

P is a finite set whose elements are $\alpha \longrightarrow \beta$, whose α and β are terminals or non terminals. Elements of P are called productions or production rules.

For example,

Consider the productions,

$S \longrightarrow aB|bA$

$A \longrightarrow aS|bAA|a$

$B \longrightarrow bS|aBB|b$

where S is the start symbol.

For the above productions the grammar is defined as,

$V_N = \{S, A, B\}$

$\sum = \{a, b\}$

$S$is the start symbol S

$P = \{S \longrightarrow aB|bA, A \longrightarrow aS|bAA|a, B \longrightarrow bS|aBB|b\}$

Derivations

Productions are used to derive various sentences.

Example 1:

Consider the grammar given below:

SENTENCE ⟶ NOUN_PHRASE VERB_PHRASE

NOUN_PHRASE ⟶ ARTICLE NOUN

VERB_PHRASE ⟶ VERB NOUN_PHRASE

ARTICLE ⟶ a | an | the

NOUN ⟶ boy | apple

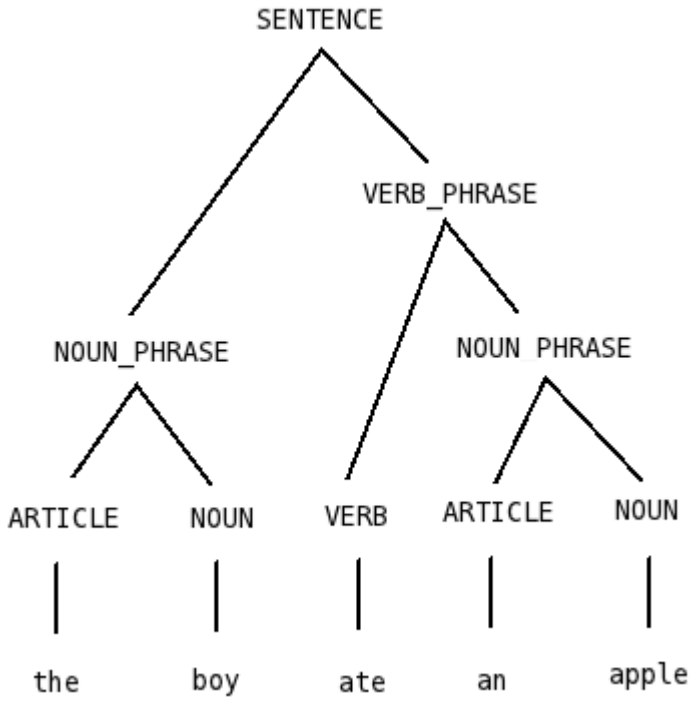
VERB ⟶ apple

where SENTENCE is the start symbol.

Using this grammar, the sentence

'the boy ate an apple'

is derived as shown below:



Example 2:

Consider the grammar given below:

$S \longrightarrow baaS \mid baa$

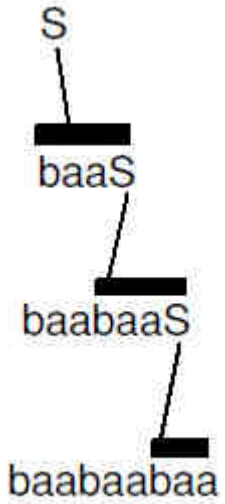
In the above, S is the start symbol, S is a non-terminal and a, b are terminals.

To derive a sentence, we begin with the start symbol, S.

From the above, we can rewrite, S as baa.

Thus the sentence, baa is a valid sentence according to the grammar.

From the above rule, rewrite S as baaS. Then rewrite S in baaS as baaS. Then we get baabaaS. Again rewrite S as baa. We get baabaabaa.



Thus baabaabaa is a valid sentence.

Example 3:

Following is a grammar for expressions:

Expr ⟶ Expr Op num | num

Op ⟶ + | - | * | /

In the above grammar, Expr, Op are non-terminals.

num, +, -, *, / are terminals.

Using the above grammar, a large set of expressions can be derived.

Example 4:

The following is a grammer for if..else construct in C.

Stmt ⟶ if (Expr) Stmt else Stmt | if (Expr) Stmt

Example 5:

The following is a grammar for a set of statements in C.

Stmt $\longrightarrow$ id = Expr;

| if (Expr) Stmt

| if (Expr) Stmt else Stmt

| while (Expr) stmt

| do Stmt while (Expr);

| { Stmts }

Stmts $\longrightarrow$ Stmts Stmt

| ε

8.2 Formal Definition of a Language

The language generated by a grammar G is L(G) is defined as

$L(G) = \{ w \epsilon \sum * \mid S* \Longrightarrow_G w \}$

The elements of L(G) are called sentences.

In a simple way, L(G) is the set of all sentences derived from the start symbol S.

For example, for the grammar

$S \longrightarrow baaS \mid baa$

L(G) = {baa, baabaa, baabaabaa, ... }

Example 1:

Consider the grammar given below:

ARTICLE $\longrightarrow$ a | an | the

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

NOUN $\longrightarrow$ boy | apple

where NOUN_PHRASE is the start symbol.

Find the language L(G) generated by the grammar.

i)

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

$\Longrightarrow$a boy

ii)

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

$\Longrightarrow$an apple

iii)

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

$\Longrightarrow$the boy

iv)

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

$\Longrightarrow$the apple

v)

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

$\Longrightarrow$a apple

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

$\Longrightarrow$an boy

Hence L(G) = { a boy, an apple, the apple, the boy, a apple, an boy}

Note that 'a apple' and 'an boy' are in the set even if they are not valid English phrases.

Exercises:

1. Let grammar G = [{S,C}, {a,b}, P, S]

where P consists of

$S \longrightarrow aCa$

$C \longrightarrow aCa|b$. Find L(G).

2. Let $G = [\{S, A_1, A_2\}, \{a, b\}, P, S]$, where P consists of

$S \longrightarrow aA_1A_2a$

$A_1 \longrightarrow baA_1A_2b$

$A_2 \longrightarrow A_1ab$

$aA_1 \longrightarrow baa$

$bA_2b \longrightarrow abab$

Check whether w = baabbabaaabbaba is in L(G).

2. Let grammar G = [{S}, {0,1}, {S $\longrightarrow$0S1, S $\longrightarrow \epsilon$}, S]. Find L(G).

9 Chomsky Classification of Languages

[Kavitha2004]

A number of language families are there. Noam Chomsky, a founder of formal language theory, provided an initial classification into four language types, type 0 to type 3.

A grammar represents the structure of a language as we saw earlier.

Four types of languages and their associated grammars are defined in Chomsky Hierarchy (defined by Noam Chomsky).

The languages are

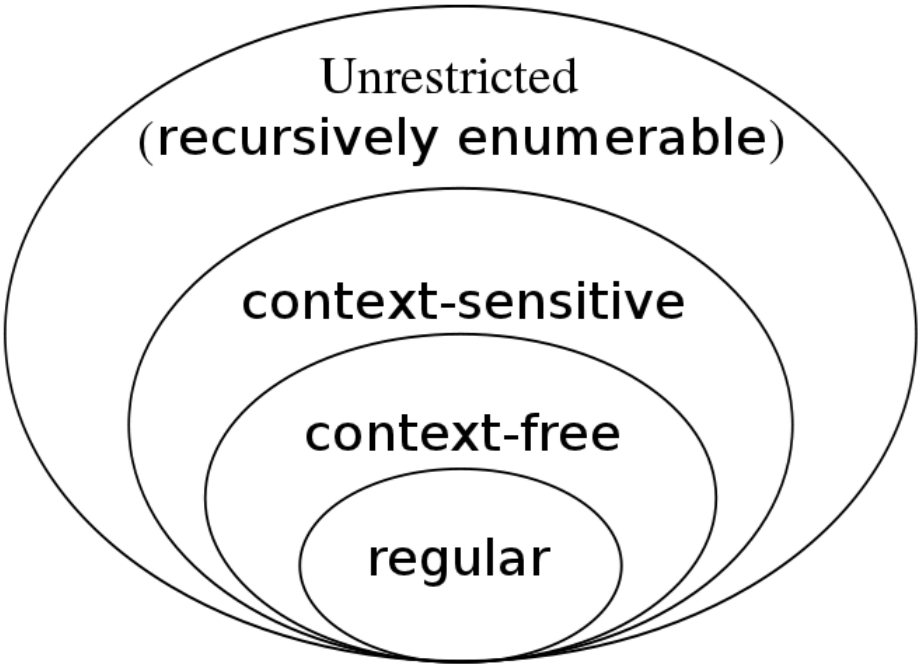
Type- 0 languages or Unrestricted languages,

Type- 1 languages or Context sensitive languages,

Type- 2 languages or Context free languages,

Type- 3 languages or regular languages.

Each language family is a proper subset of another one. The following figure shows the relationship.



Type-0 languages are generated by Type-0 grammars, Type-1 languages by Type-1 grammars, Type-2 languages by Type-2 grammars and Type-3 languages by Type-3 grammars.

Thus the corresponding grammars are

Type- 0 grammars or Unrestricted grammars,

Type- 1 grammars or Context sensitive grammars,

Type- 2 grammars or Context free grammars,

Type- 3 grammars or regular grammars.

Type- 0 or Unrestricted languages vs Type- 0 grammars

Type- 0 languages are defined by arbitrary grammars (Type-0).

Type- 0 grammars have productions of the form, $\alpha \longrightarrow \beta$, such that $\alpha \neq \varepsilon$.

Type- 0 languages are recognized using Turing Machines (TM).

The following is an example for an Unrestricted grammar:

$S \longrightarrow ACaB$

$Ca \longrightarrow aaC$

$CB \longrightarrow DB$

$aD \longrightarrow Da$

$aE \longrightarrow Ea$

$AE \longrightarrow \varepsilon$

Type- 1 or Context- sensitive languages vs Type- 1 grammars

Type- 1 languages are described by Type- 1 grammars.

LHS of a Type- 1 grammar is not longer than the RHS.

Type- 1 grammars are represented by the productions $\alpha \longrightarrow \beta$, such that $|\alpha| \leq |\beta|$.

Type- 1 languages are recognized using Linear Bounded Automata (LBA).

The following is an example for a Context Sensitive grammar:

$S \longrightarrow SBC$

$S \longrightarrow aC$

$B \longrightarrow a$

$CB \longrightarrow BC$

$Ba \longrightarrow aa$

$C \longrightarrow b$

Type- 2 or Context- free languages vs Type- 2 grammars

Type- 2 languages are described by Type- 2 grammars or context free grammars.

LHS of a Context free grammar (Type-2) is a single non-terminal. The RHS consists of an arbitrary string of terminals and non-terminals.

Context free grammars are represented by the productions $A \longrightarrow \beta$.

Type 2 languages are recognized using Pushdown Automata (PDA).

The following is an example for a Context free grammar:

$S \longrightarrow aSb$

$S \longrightarrow \varepsilon$

The English grammar given below is a context free grammar.

ARTICLE $\longrightarrow$ a | an | the

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

NOUN $\longrightarrow$ boy | apple

Type- 3 or Regular languages vs Type- 3 grammars

Type- 3 languages are described by Type- 3 grammars.

The LHS of a Type- 3 grammar is a single non-terminal. The RHS is empty, or consists of a single terminal or a single terminal followed by a non- terminal.

Type- 3 grammars are represented by the productions $A \longrightarrow aB$ or $A \longrightarrow a$.

Type- 3 languages are recognized using Finite State Automata (FA).

The following is an example for a regular grammar:

$A \longrightarrow aA$

$A \longrightarrow \varepsilon$

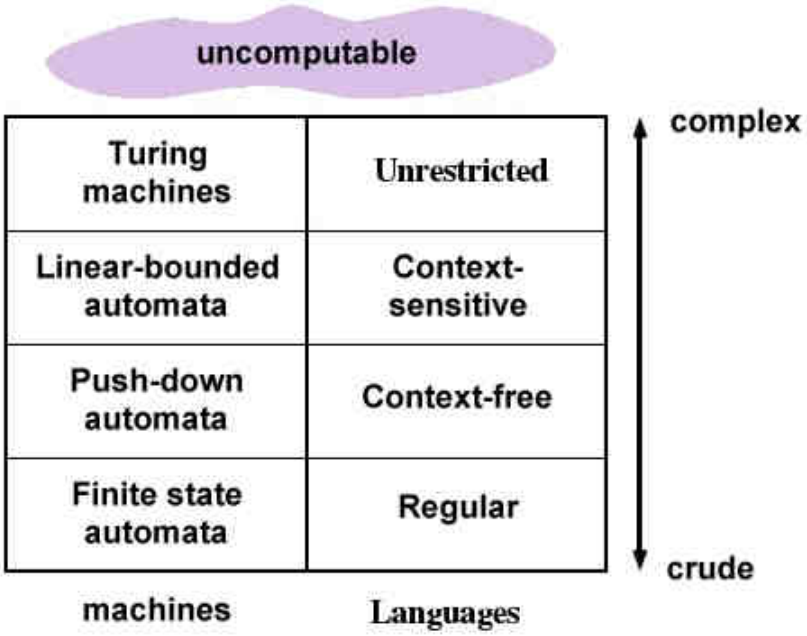
$A \longrightarrow bB$

$B \longrightarrow bB$

$B \longrightarrow \varepsilon$

$A \longrightarrow c$

Following figure shows the relation between the four types of languages and machines.



Questions (from Old syllabus of S7CS TOC)

MGU/Nov2011

1. What is meant by a formal proof? (4marks).

2. a. Differentiate between primitive recursive and partial recursive functions with examples (12marks).

OR

b. Write notes on: ii) diagonalization principle. (12marks)

MGU/April2011

2. What is a primitive recursive function (4marks)?

3. a. Show that

ii) For n is an integer >=0, $n^3 + 2n$ is divisible by 3 (12marks).

OR

b. i) Show that a problem whose language is recursive is undecidable.

ii) If Z is the decimal expansion of pi, then show that the function f(x) = 1 for exactly x consecutive 3's in z and f(x) = 0, otherwise (12marks).

MGU/Nov2010

1. What is meant by computable and non-computable? (4marks)

2. Define a grammar and what are various types of grammars (4marks).

3 a. show that factorial of a number is a primitive recursive function.

OR

b. i) Show that multiplication is a computable function on the set of natural numbers.

ii). Discuss the computability of f(x) = 1 for xth digit of Z = 3 and f(x) = 0 otherwise Z is a decimal expansion of $\pi = 3.14285714.....$(12marks)

MGU/May2010

1. Define primitive recursive function (4marks).

2. Differentiate cmputable function from non-computable function (4marks).

3 a. Explain what is meant by the following statements:-

i) f: N1N is a total recursive (TR) function.

ii) The sequence $\{f_n : N1N*\}n2N$ of TR functions of a single variable is recursively enumerable.

iii) Show that no recursive enumeration can include the set of all TR functions of a single variable (12marks).

MGU/Nov2009

1. What does it mean for a subset "s" of the set "N" of natural numbers to be register machine decidable? (4marks).

2. What is diagonalisation principle? (4marks)

3a. Prove that the union and intersection of two recursive languages are also recursive.

OR

b. Discuss the proof that the closure properties of regular sets are closed (12mraks).

MGU/Nov2008

1. Give the formal definition of context sensitive grammar (4marks).

2. Show that $\Sigma_{k=0}^{n} k = \frac{n(n+1)}{2}$. (4marks)

3a. Prove that 2^n is uncountable using diagonalisation principle (12marks).

OR

b. Compare recursive function, primitive recursive function and partial recursive function (12marks).

4a

ii) Show that $f(r) = \frac{x}{2}$ is a partial recursive function (12marks).

MGU/May2008

1. What is primitive recursive function (4marks).

2. Show that $\Sigma_{i=0}^{n} i = \frac{n(m+1)}{2}$ by induction (4marks).

3a i) For any finite set A, $|2^A| = 2^{|A|}$, is cardinality of any power set a is 2 raised to a power equal to cardinality of A.

OR

b ii) Explain Chomsky classification (12marks).

MGU/JDec2007

2 a i) Check whether the language $L = \{a^n b^n / n \geq 1\}$ is regular or not. Justify your answer.

ii) Explain diagonalization principle (12marks).

OR

b ii) Explain primitive recursive and partial recursive function (12marks).

MGU/July2007

1. Define primitive recursive function (4marks).

2. Explain composition of functions (4marks).

3a. i) Prove that the function $f_{add}(x, y) = x + y$ is primitive recursive.

ii) Prove that the function $g(x, y) = x^y$ is primitive recursive (12marks).

OR

b. ii) Explain the significance of Theory of Computation in computer science (12marks).

MGU/Jan2007

1. Prove that for any set A having n>=0 elements, the power set of a has 2^n elements (4marks).

2. Explain the types of functions- injection, surjection, bijection and invertible function (4marks).

3b. i) Explain the Chomsky hierarchy of formal languages.

ii) Define formal language. Give examples for finite and infinite languages (12marks).

MGU/July2006

1. Define a partial recursive function (4marks).

2. Differentiate between deterministic and non deterministic algorithms (4marks).

4a. i)

ii) Define context sensitive and regular grammars. Give examples.

OR

b. i) Find a function f(x) such that f(2) = 3, f(4) = 5, f(7) =2 and f(x). assume any arbirtrary value for other arguments. Show that f(x) is primitive recursive.

MGU/Nov2005

1. Give the formal definition of a grammar (4marks).

2b. i) Show that $f(x) = x/2$ is a partial recursive function.

ii) Explain the different types of grammars with suitable examples (12marks).

References

.

Linz, P (2006). An Introduction to Formal Languages and Automata. Narosa.

Moret, B, M (2007). The Theory of Computation. Pearson Education.

Mishra, K, L, P; Chandrasekaran, N (2009). Theory of Computer Science. PHI.

Pandey, A, K (2006). An Introduction to automata Theory and Formal Languages. Kataria & Sons.

Kavitha,S; Balasubramanian,V (2004). Theory of Computation. Charulatha Pub.

website: http://sites.google.com/site/sjcetcssz

St. Joseph's College of Engineering & Technology Palai

Department of Computer Science & Engineering

S4 CS

CS010 406 Theory of Computation

Module 2

Theory of Computation - Module 2

Syllabus

Introduction to Automata Theory - Definition of Automation - Finite automata - Language acceptibility by finite automata - Deterministic and non deterministic finite automation -

Regular expressions - Finite automation with ϵ transitions - Conversion of NFA to DFA - Minimisation of DFA - DFA to Regular expressions conversion -

Pumping lemma for regular languages -

Applications of finite automata - NFA with o/p (moore / mealy)

Contents

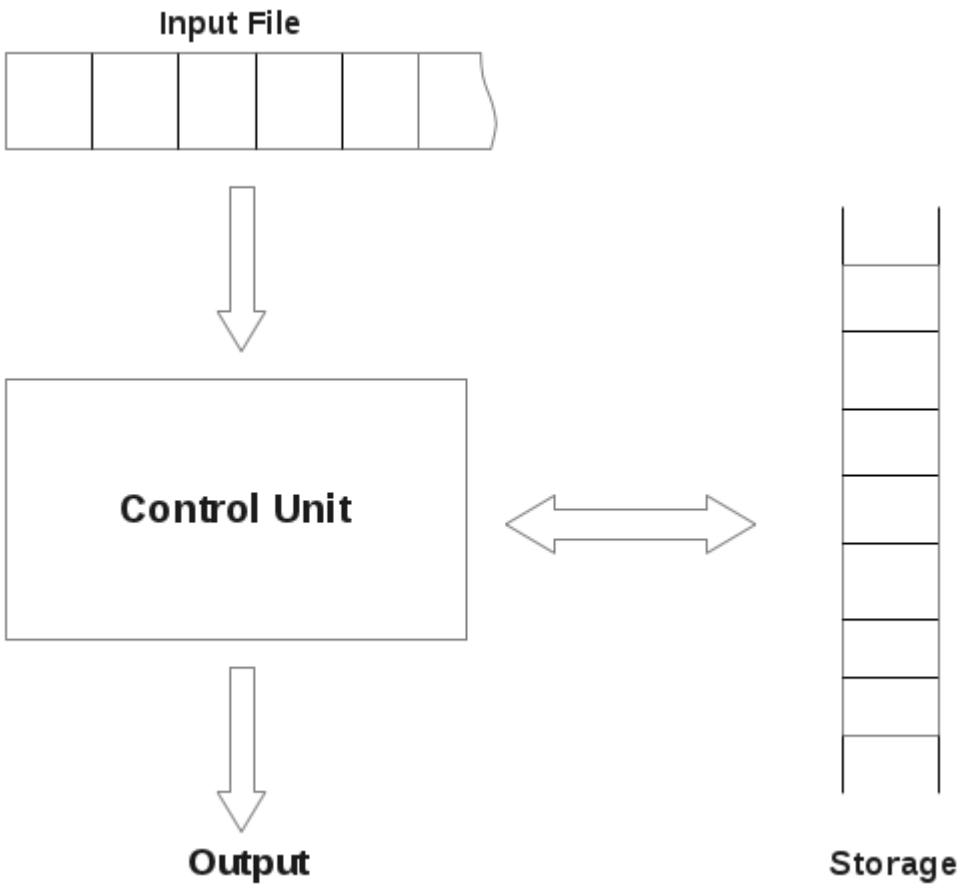
Part I. Introduction to Automata Theory

1 Automation

An automation is defined as a system that performs certain functions without human intervention. It accepts some input as row materials and converts it to a product under the guidance of control signals. An automation can be designed to perform a variety of tasks related to various domain of human life.

In terms of computer science, an automation is an abstract model of a digital computer. An automation has some features.

following figure shows the representation of an automation.



Input

An automation has a mechanism for reading input. It is assumed that input is in a file. The input file is divided into cells. Each cell can hold one symbol. Input mechanism can read input file from left to right.

Output

The automation can produce output of some form.

Storage

An automation can have a temporary storage device. the storage device can consist of unlimited number of cells. The automation can read and change the contents of the storage cells.

Control Unit

Automation has a control unit. At a given time, control unit is in some internal state.

2 Automata Theory

automata theory is the study of abstract computing devices or machines.

The study of automata is important because ,

1. Automata theory plays an important role when we make software for designing and checking the behaviour of a digital circuit.
2. The lexical analysis of a compiler breaks a program into logical units, such as variables, keywords and punctuation using this mechanism.
3. Automata theory works behind software for scanning large bodies of text, such as web pages to find occurrence of words, phrases etc..
4. Automata theory is a key to software for verifying systems of all types (software testing).
5. Automata theory is the most useful concept of software for natural language processing.

Definition of an Automation

an automation is defined as a program where energy, material and information are transformed, transmitted and used for performing some functions without direct human intervention.

Example: Automatic machine tools,
Automatic packing machines,
Automatic photo copying machines.

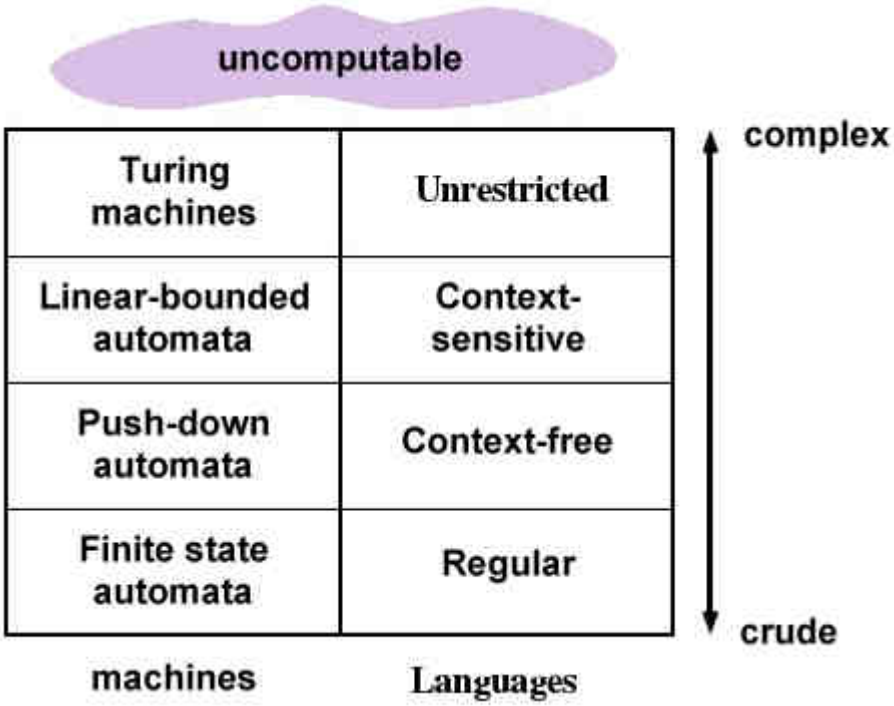
Different classes of automata are ,
Finite automata, and
Pushdown automata.

We will learn both of them in detail.

Part II. Finite Automata

Different classes of automata are there. examples: finite automata, pushdown automata. Of them finite automata is the simplest one.

In Module I, during the discussion of chomsky classification, it was seen that type-3 or regular languages are recognised using finite automata.



Definition

A finite automation, M is a 5 tuple structure,

$$M\,(\,Q,\ \sum,\ q_0,\ \delta,\ F\,)$$

where M is the finite automation,

Q is a finite set of states,

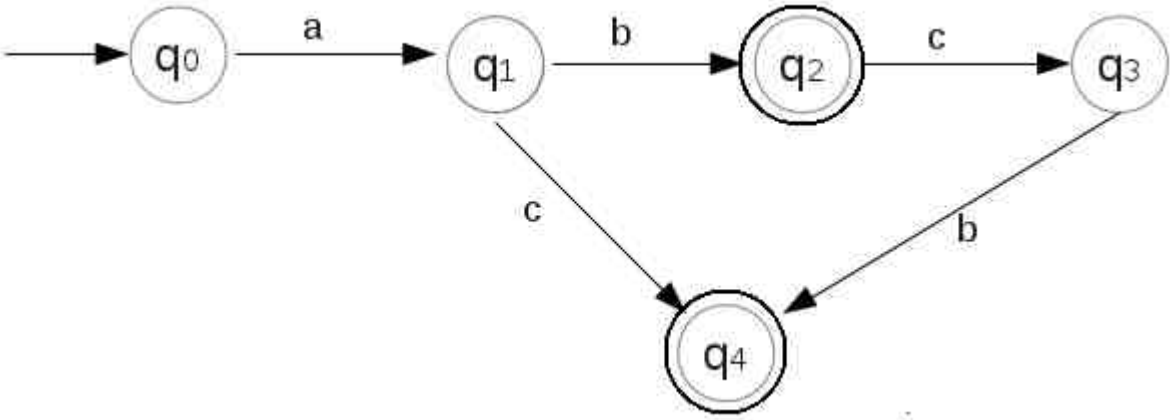
$\sum$ is a set of input symbols,

q_0 is the start state,

δ is the set of transition functions,

F is the set of final states.

Consider the following example for a finite automation, M.



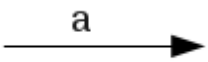
In the above finite automation, M,

1. $Q = \{q_0, q_1, q_2, q_3, q_4\}$

Q is the set of states in the finite automata. In the above finite automata, q_0, q_1, q_2, q_3, q_4 are the states. The states are shown in circles.

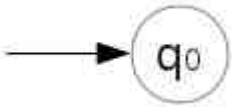
2. $\sum = a, b, c$

$\sum$ is the set of input symbols in the finite automata. In the above, a, b, c are the input symbols. The arrowsare labelled with input symbols.



3. $q_0 = \{q_0\}$

q_0 is the start state. In the above, q_0 is the start state. For our study, a finite automata contains only one start state. A state with arrow from free space (not coming from any other state) is designated as start state.



4. δ describes the operation of finite automation. δ is the transition function.

For example,

$$\delta(q_0, a) = q_1$$

This means, given the current state q_0 and the input symbol, a, the finite automation moves (transits) to state q_1.

Similarly,

$$\delta(q_1, b) = q_2$$

$$\delta(q_2, c) = q_3$$

$$\delta(q_1, c) = q_4$$

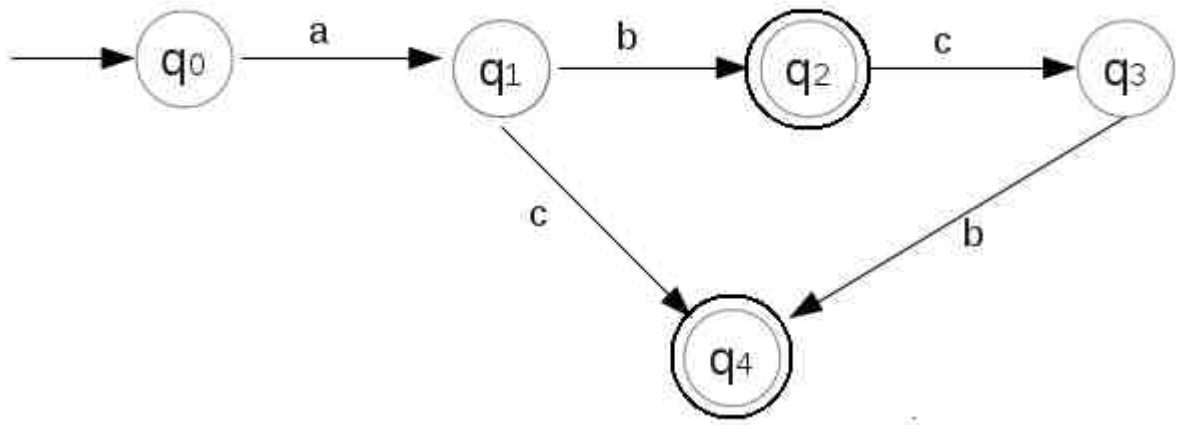
$$\delta(q_3, b) = q_4$$

5. $F = \{q_2, q_4\}$

F is the set of final states or accepting states. In the above finite automation, $q_2$and q_4 are the final states. They are shown enclosed in double circles.

Transition Diagrams and Transition Tables

Consider the following finite automation, M.



The above representation of finite automation is called a transition diagram.

A finite automata can also be represented as a transition table. This is shown below:

	Input Symbol		
Current State	a	b	c
$\longrightarrow q_0$	q_1	ϕ	ϕ
q_1	ϕ	q_2	q_4
$*q_2$	ϕ	ϕ	q_3
q_3	ϕ	q_4	ϕ
$*q_4$	ϕ	ϕ	ϕ

$\longrightarrow q_0$ denotes that q_0 is the start state.

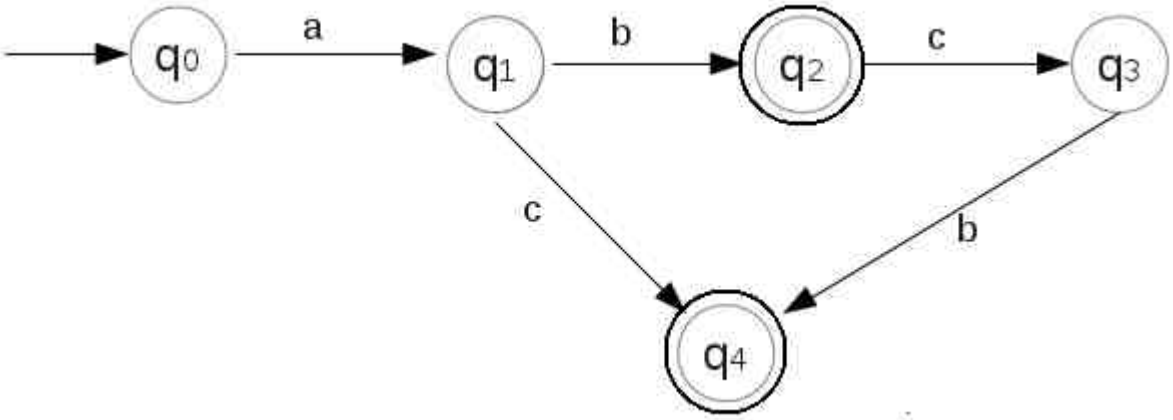
$*$ before q_2 and q_4 indicates that q_2 and q_4 are the final states.

3 Language Acceptibility by Finite Automata

String Processing by Finite Automation

Example 1:

Consider the following finite automation, M.



Check whether the string abcb is accepted by the above finite automation.

In the above, q_0 is the start state. the string abcb has the first symbol, a.

So,

$\delta(q_0, \underline{a}bcb) = q_1$

$\delta(q_1, a\underline{b}cb) = q_2$

$\delta(q_2, ab\underline{c}b) = q_3$

$\delta(q_3, abc\underline{b}) = q_4$

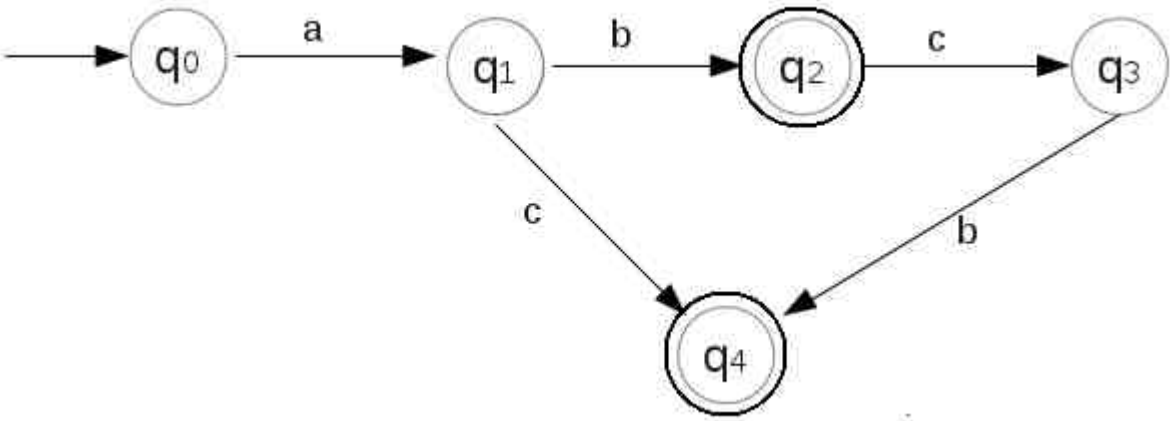
Thus,

$q_0 \overset{a}{\Longrightarrow} q_1 \overset{b}{\Longrightarrow} q_2 \overset{c}{\Longrightarrow} q_3 \overset{b}{\Longrightarrow} q_4$

Thus, after processing the string abcb, we reached the state, q_4. Here q_4 is a final state. So the string abcb is accepted by the finite automation.

Example 2:

Consider the following finite automation, M



Check whether the string abc is accepted by M.

On processing the string, beginning from the start state, q_0,

$\delta(q_0, \underline{a}bc) = q_1$

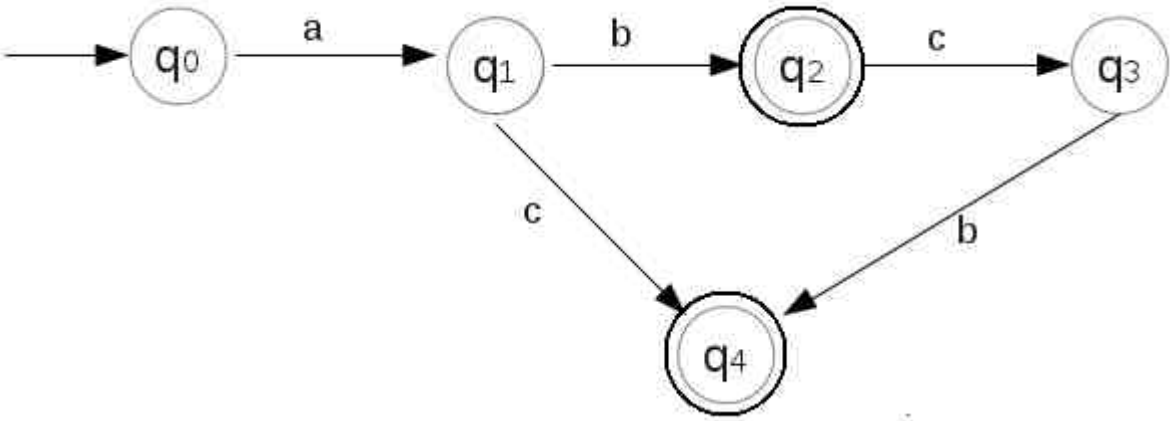
$\delta(q_1, a\underline{b}c) = q_2$

$\delta(q_2, ab\underline{c}) = q_3$

Here M halts at q_3. But q_3 is not a final state. Hence the string abc is rejected by the above finite automation, M.

Example 3:

Conisder the finite automation, M,



Check whether the string abca is accepted by the finite automation, M.

Beginning from the start state, q_0,

$\delta(q_0, \underline{a}bca) = q_1$

$\delta(q_1, a\underline{b}ca) = q_2$

$\delta(q_2, ab\underline{c}a) = q_3$

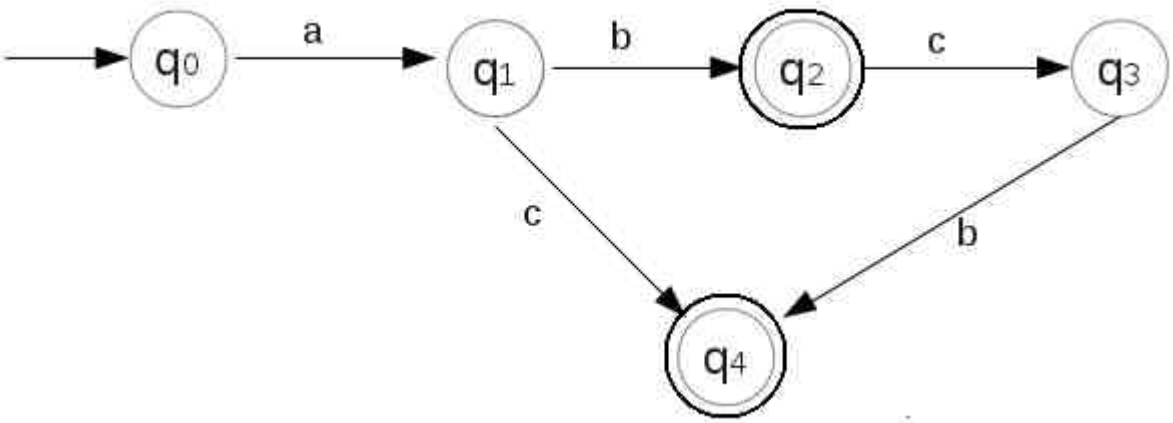
$\delta(q_3, abc\underline{a}) =$

Here for current state, q_3, and for input symbol, a, there is no transition. This means M is unable to process the string abca completely. So the string abca is rejected by M.

Language of Finite Automation

In the above, finite automation is used to check the validity of a string. A language is a finite set of strings. So we can use finite automation to check whether a string belongs to a language.

Consider the following finite automation,



Let L be the language corresponding to the above finite automation, M. Then a string belongs to the language L only if it is accepted by the finite automation, M.

We can say that,

The string abcb belongs to the language, L.

The string abc does not belong to the language, L.

The string abca does not belong to the language, L.

The string ab belongs to the language, L.

The string ac belongs to the language, L.

The string abcbc does not belong to the language , L.

4 Types of Finite Automata

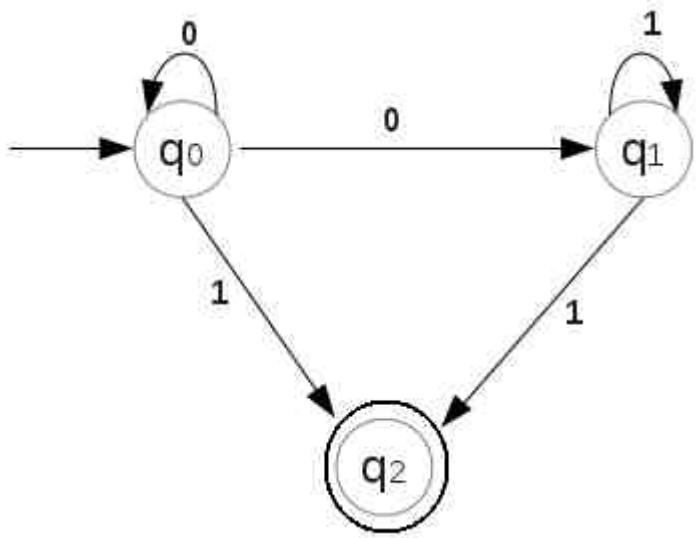
Finite automation is of different types. They are,

Non-deterministic Finite Automata (NFA), and

Deterministic Finite Automata (DFA).

4.1 Non- Deterministic Finite Automata (NFA)

Consider the following finite automation, M,

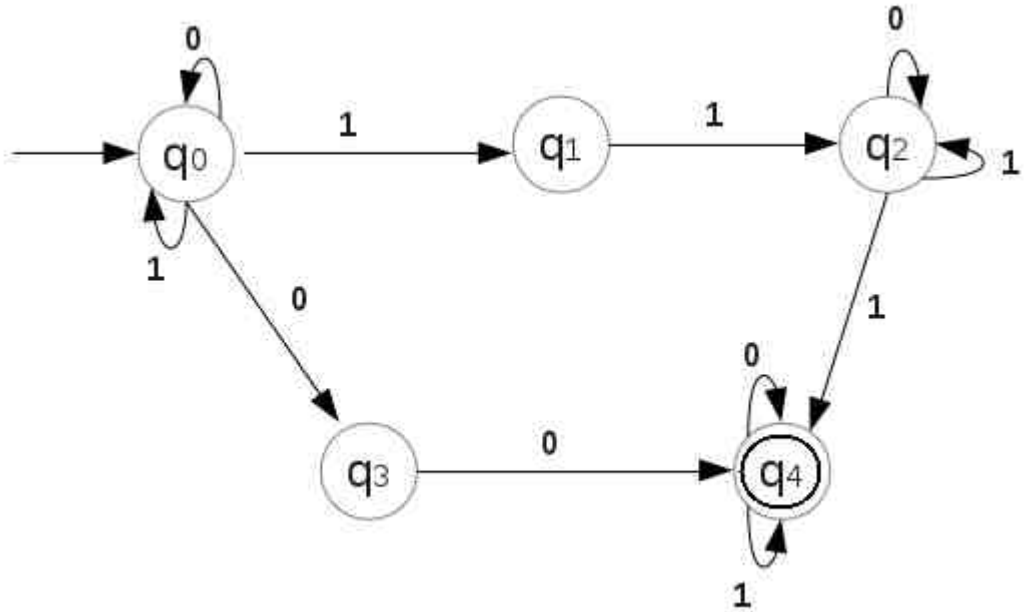


In the above, from state q_0 with input symbol 0, there are two possible next states. The next state can be either q_0 or q_1. Also from state q_1, on input symbol, 1, there are two possible next states. The next state can be either $q_1$or q_2.

Thus some moves of the finite automation cannot be detrmined uniquely by the input symbol and the current state. Such finite automation are called Non-deterministic Finite automata (NFA or NDFA).

Example 1:

Consider the following NFA,



Check whether the string 0100 is accepted by the above NFA.

Beginning from the start symbol, q_0,

$\delta(q_0, \underline{0}100) = q_0$

$\delta(q_0, 0\underline{1}00) = q_0$

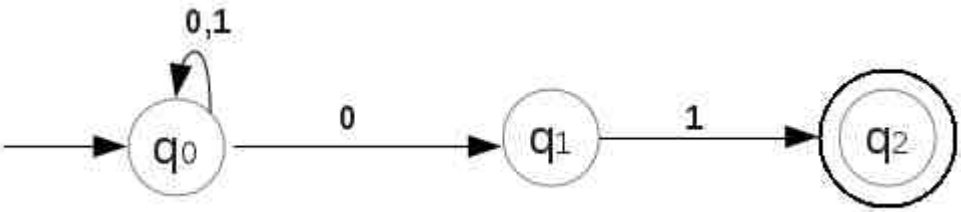
$\delta(q_0, 01\underline{0}0) = q_3$

$\delta(q_3, 010\underline{0}) = q_4$

q_4 is a final state. So the string 0100 is accepted by the above finite automation.

Example 2:

Consider the following NFA,



Check whether the string 01101 is accepted by the above NFA.

$\delta(q_0, \underline{0}1101) = q_0$

$\delta(q_0, 0\underline{1}101) = q_0$

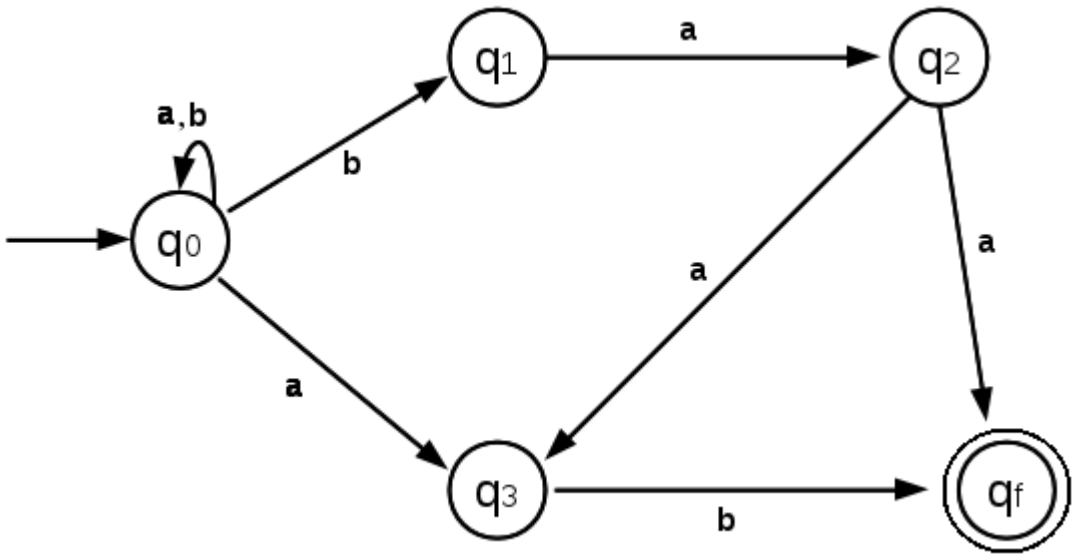
$\delta(q_0, 01\underline{1}01) = q_0$

$\delta(q_0, 011\underline{0}1) = q_1$

$\delta(q_1, 0110\underline{1}) = q_2$

q_2 is a final state. So the string 01101 is accepted by the above NFA.

Example 3:



Check whether the string abbaa is accepted by the above NFA.

$\delta(q_0, \underline{a}bbaa) = q_0$

$\delta(q_0, a\underline{b}baa) = q_0$

$\delta(q_0, ab\underline{b}aa) = q_1$

$\delta(q_1, abb\underline{a}a) = q_2$

$\delta(q_1, abba\underline{a}) = q_f$

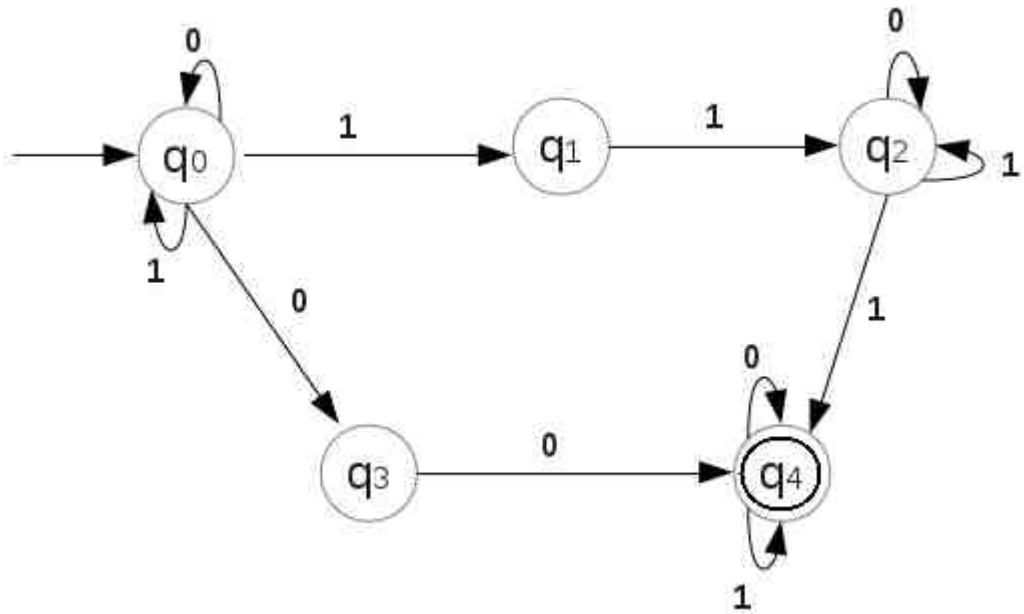
q_f is a final state. So the string abbaa is accepted by the above NFA.

4.1.1 NFA with Epsilon Transitions

An NFA can contain certain transitions on input symbol, ε (epsilon).

ε means 'null'.

For example, consider the following NFA,



Check whether the string 100110 is accepted by the above NFA.

$\delta(q_0, \underline{1}00110) = q_0$

$\delta(q_0, 1\underline{0}0110) = q_0$

$\delta(q_0, 10\underline{0}110) = q_0$

$\delta(q_0, 100\underline{1}10) = q_1$

$\delta(q_1, 1001\underline{1}0) = q_2$

$\delta(q_2, 10011\underline{0}) = q_2$

$\delta(q_2, \varepsilon) = q_4$

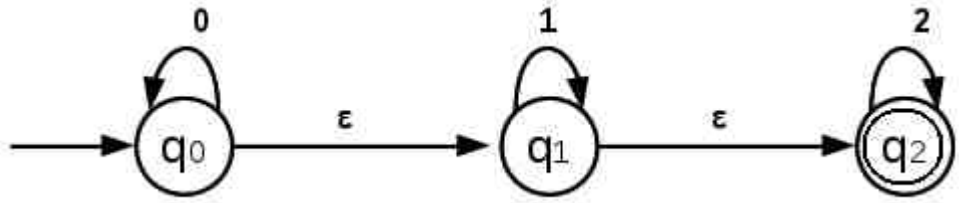
q_4 is a final state. So the string 100110 is accepted by the above NFA.

Here note that, we made an ε move to reach the final state, q_4.

Thus, ε transition means a transition without scanning the symbol in the input string.

Example 2:

Consider the following NFA,



Check whether the string 0012 is accepted by the above NFA.

Beginning with the start state, q_0,

$\delta(q_0, \underline{0}012) = q_0$

$\delta(q_0, 0\underline{0}12) = q_0$

$\delta(q_0, \varepsilon) = q_1$

$\delta(q_1, 00\underline{1}2) = q_1$

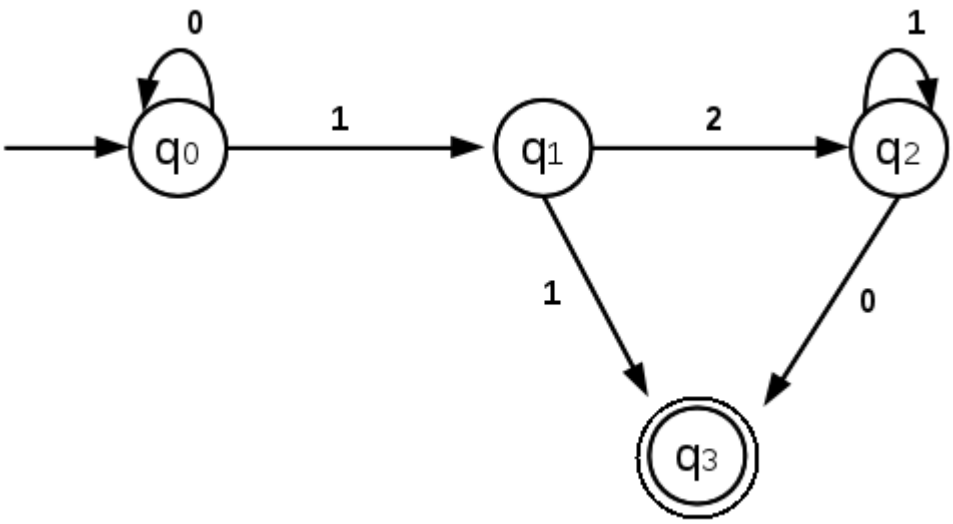
$\delta(q_1, \varepsilon) = q_2$

$\delta(q_2, 001\underline{2}) = q_2$

Here q_2 is a final state. So the string 0012 is accepted by the NFA.

4.2 Deterministic Finite Automata

Consider the following finite automation,



In the above, input symbols are 0, 1 and 2. From every state, there is only one transition for an input symbol.

From state, q_0, for symbol, 0, there is only one transition, that is to state q_0 itself.

From state, q_0, for symbol, 1, there is only one transition, that is to state q_1.

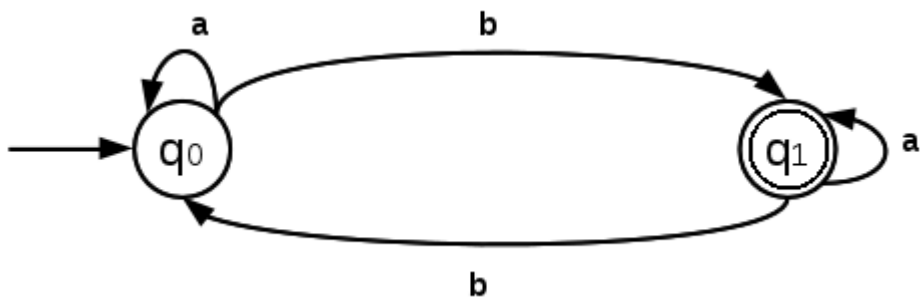
From state, q_1, for symbol, 1, there is only one transition, that is to state q_3, and so on.

Thus all the moves of the above finite automation can be determined uniquely by the current state and input symbol. Such a finite automation is called Deterministic finite Automation (DFA).

Thus in a DFA, at every step, next move is uniquely determined. No ε transitions are present in a DFA.

Example 1:

The following shows a DFA,



Above transition diagram can be represented using the transition table as follows:

	Input	Symbol
Current State	a	b
$\longrightarrow q_0$	q_0	q_1
$*q_1$	q_1	q_0

Check whether the string aaba is accepted by the above DFA.

Beginning from the start state, q_0,

$$\delta(q_0, \underline{a}aba) = q_0$$

$$\delta(q_0, a\underline{a}ba) = q_0$$

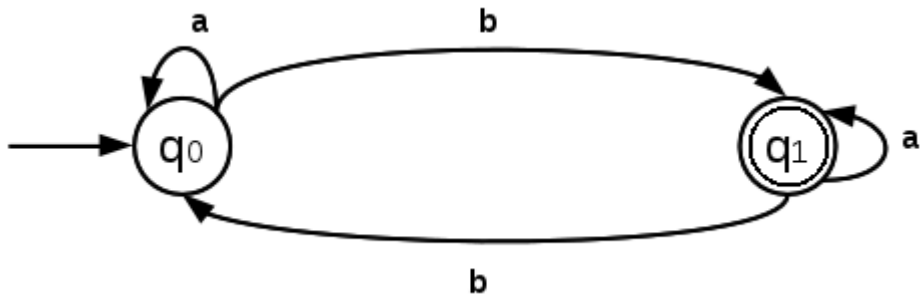
$$\delta(q_0, aa\underline{b}a) = q_1$$

$$\delta(q_1, aab\underline{a}) = q_1$$

After processing all the characters in the input string, final state $q_1$is reached. So the string aaba is accepted by the above finite automation.

Example 2:

Consider the following DFA,



Check whether the string aabba is accepted by the above DFA.

$\delta(q_0, \underline{a}abba) = q_0$

$\delta(q_0, a\underline{a}bba) = q_0$

$\delta(q_0, aa\underline{b}ba) = q_1$

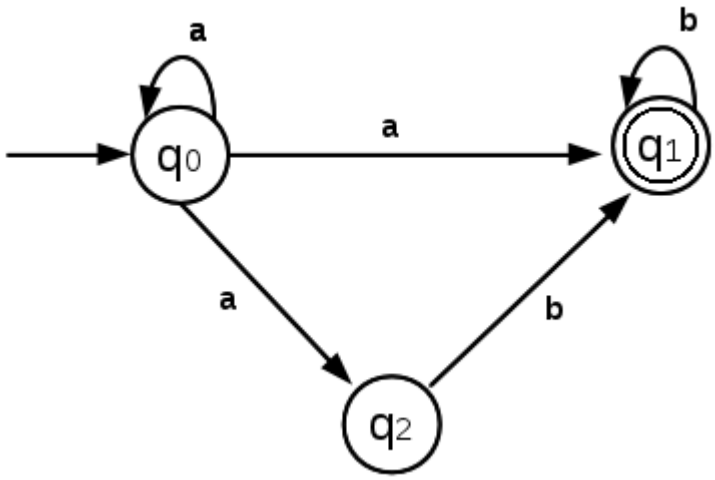
$\delta(q_1, aab\underline{b}a) = q_0$

$\delta(q_0, aabb\underline{a}) = q_0$

After processing all the symbols in the input string, aabba, the state q_0 is reached. But q_0 is not a final state. So the DFA rejects the string aabba.

Part III. NFA to DFA Conversion

Consider the following NFA,



In the above NFA, from state q_0, there are 3 possible moves for input symbol, a. ie.. to q_0, q_1, q_2. When we simulate this NFA using a program, the move from q_0 for symbol a is non-deterministic. That means, which transition is to be made next cannot be determined in one move.

But in a DFA, from a state, for an input symbol, only one transition exists. So it is very easy to simulate a DFA using a program. So we need to convert an NFA to an equivalent DFA.

For every NFA, there exists an equivalent DFA.

5 Epsilon Closure

For converting an NFA to DFA, we first need to learn how to find the epsilon closure of a state.

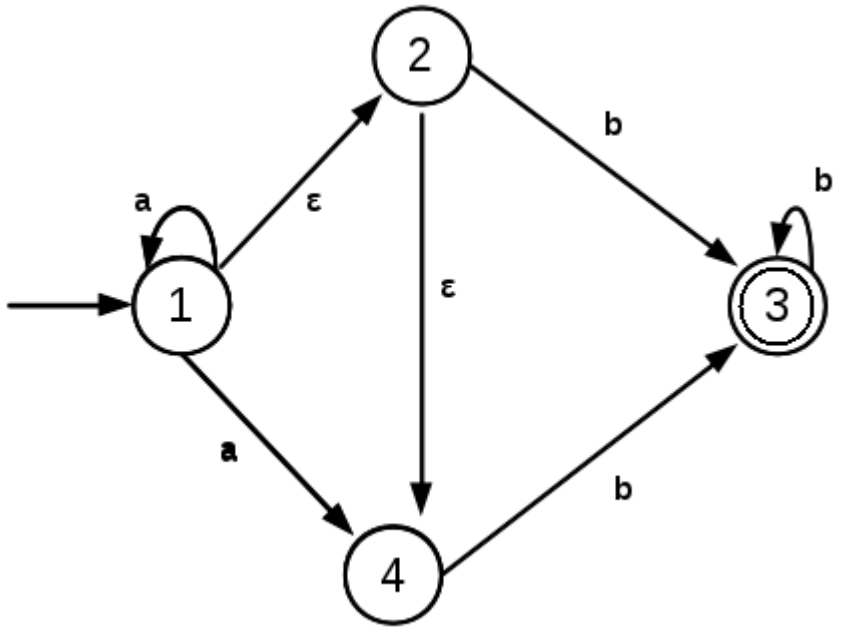
Epsilon closure (ε- closure) of a state q is the set that contains the state q itself and all other states that can be reached from state q by following ε transitions.

We use the term, ECLOSE(q_0) to denote the Epsilon closure of state q_0.

For example,

Example 1:

Consider the following NFA,



Find the epsilon closure of all states.

Here,

$ECLOSE(1) = \{1, 2, 4\}$

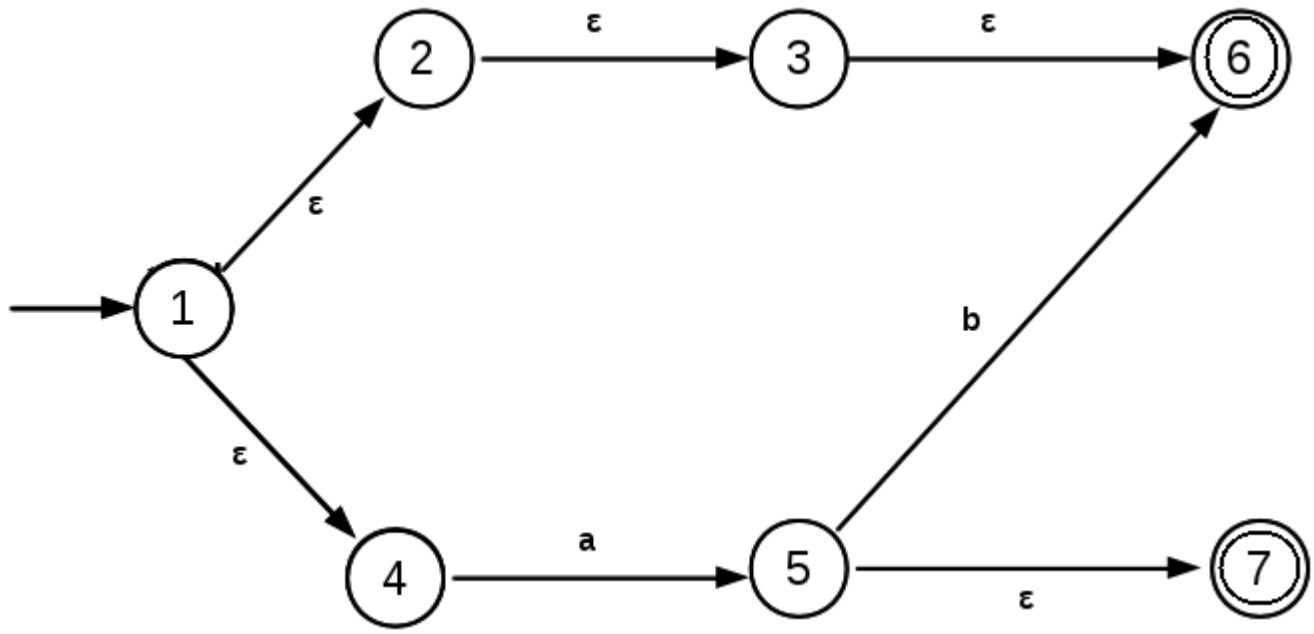
$ECLOSE(2) = \{2, 4\}$

$ECLOSE(3) = \{3\}$

$ECLOSE(4) = \{4\}$

Example 2:

Consider the following NFA,



Find the epsilon closure of all states.

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

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

$ECLOSE(3) = \{3, 6\}$

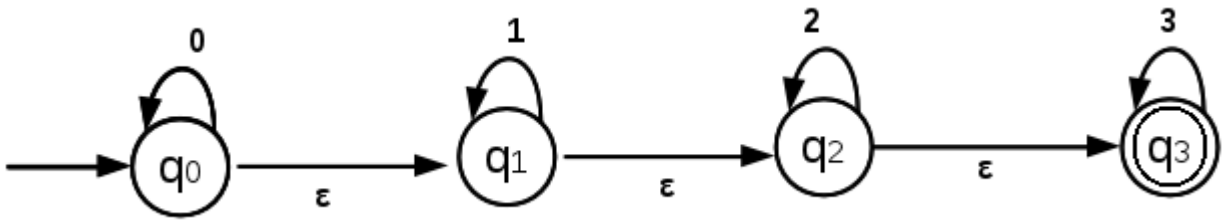
$ECLOSE(4) = \{4\}$

$ECLOSE(5) = \{5, 7\}$

$ECLOSE(6) = \{6\}$

Example 3:

Consider the following NFA,



Compute the Epsilon closure of all states.

$ECLOSE(q_0) = \{q_0, q_1, q_2, q_3\}$

$ECLOSE(q_1) = \{q_1, q_2, q_3\}$

$ECLOSE(q_2) = \{q_2, q_3\}$

$ECLOSE(q_3) = \{q_3\}$

6 Conversion of NFA to DFA

The steps involved in the conversion of NFA to DFA are,

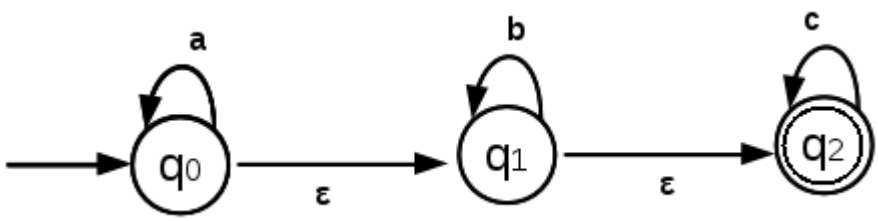
1. Transform the NFA with Epsilon transitions to NFA without epsilon transitions.
2. convert the resulting NFA to DFA.

These steps are exaplained in detail as follows:

1. Transform the NFA with ε transitions to NFA without ε transitions.

Example 1:

Consider the following NFA with epsilon transitions:



The above NFA has states, q_0, q_1, q_2. The start state is q_0. The final state is q_2.

Step a: Find the Epsion closure of all states.

$ECLOSE(q_0) = \{q_0, q_1, q_2\}$

$ECLOSE(q_1) = \{q_1, q_2\}$

$ECLOSE(q_2) = \{q_2\}$

The states of the new NFA will be $\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}$.

The start state of the new NFA will be the epsilon closure of the start state q_0.

Thus the start state of the new NFA will be $\{q_0, q_1, q_2\}$

The final states of the new NFA will be those new states that contain the final states of the old NFA.

Thus the final states of the new NFA are $\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}$, since they contain the final state q_2 of the old NFA.

Next we need to find the transitions of these new states on input symbols a, b and c.

Consider state $\{q_0, q_1, q_2\}$,

$$\begin{aligned}
\delta'(\{q_0, q_1, q_2\}, a) &= ECLOSE[\delta(\{q_0, q_1, q_2\}, a)] \\
&= ECLOSE[\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)] \\
&= ECLOSE[q_0 \cup \phi \cup \phi] \\
&= ECLOSE[q_0] \\
&= \{q_0, q_1, q_2\} \\
\delta'(\{q_0, q_1, q_2\}, b) &= ECLOSE[\delta(\{q_0, q_1, q_2\}, b)] \\
&= ECLOSE[\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)] \\
&= ECLOSE[\phi \cup q_1 \cup \phi] \\
&= ECLOSE[q_1] \\
&= \{q_1, q_2\} \\
\delta'(\{q_0, q_1, q_2\}, c) &= ECLOSE[\delta(\{q_0, q_1, q_2\}, c)] \\
&= ECLOSE[\delta(q_0, c) \cup \delta(q_1, c) \cup \delta(q_2, c)] \\
&= ECLOSE[\phi \cup \phi \cup q_2] \\
&= ECLOSE[q_2] \\
&= \{q_2\}
\end{aligned}$$

Consider the state $\{q_1, q_2\}$

$$\begin{aligned}
\delta'(\{q_1, q_2\}, a) &= ECLOSE[\delta(\{q_1, q_2\}, a)] \\
&= ECLOSE[\delta(q_1, a) \cup \delta(q_2, a)] \\
&= ECLOSE[\phi \cup \phi] \\
&= ECLOSE[\phi] \\
&= \phi \\
\delta'(\{q_1, q_2\}, b) &= ECLOSE[\delta(\{q_1, q_2\}, b)] \\
&= ECLOSE[\delta(q_1, b) \cup \delta(q_2, b)] \\
&= ECLOSE[q_1 \cup \phi] \\
&= ECLOSE[q_1] \\
&= \{q_1, q_2\}
\end{aligned}$$

$$\begin{aligned}\delta'(\{q_1,q_2\},c) &= ECLOSE[\delta(\{q_1,q_2\},c)]\\ &= ECLOSE[\delta(q_1,c)\cup\delta(q_2,c)]\\ &= ECLOSE[\phi\cup q_2]\\ &= ECLOSE[q_2]\\ &= \{q_2\}\end{aligned}$$

Conisider the state q_2

$$\begin{aligned}\delta'(q_2,a) &= ECLOSE[\delta(q_2,a)]\\ &= ECLOSE[\phi]\\ &= \phi\\ \delta'(q_2,b) &= ECLOSE[\delta(q_2,b)]\\ &= ECLOSE[\phi]\\ &= \phi\\ \delta'(q_2,c) &= ECLOSE[\delta(q_2,c)]\\ &= ECLOSE[q_2]\\ &= \{q_2\}\end{aligned}$$

Now we have the new NFA without epsilon transitions.

The transition table for the new NFA is,

Current state	Input Symbol		
	a	b	c
$\longrightarrow *\{q_0,q_1,q_2\}$	$\{q_0,q_1,q_2\}$	$\{q_1,q_2\}$	$\{q_2\}$
$*\{q_1,q_2\}$	ϕ	$\{q_1,q_2\}$	$\{q_2\}$
$*\{q_2\}$	ϕ	ϕ	$\{q_2\}$

Let us say,

$\{q_0,q_1,q_2\}$ as q_x

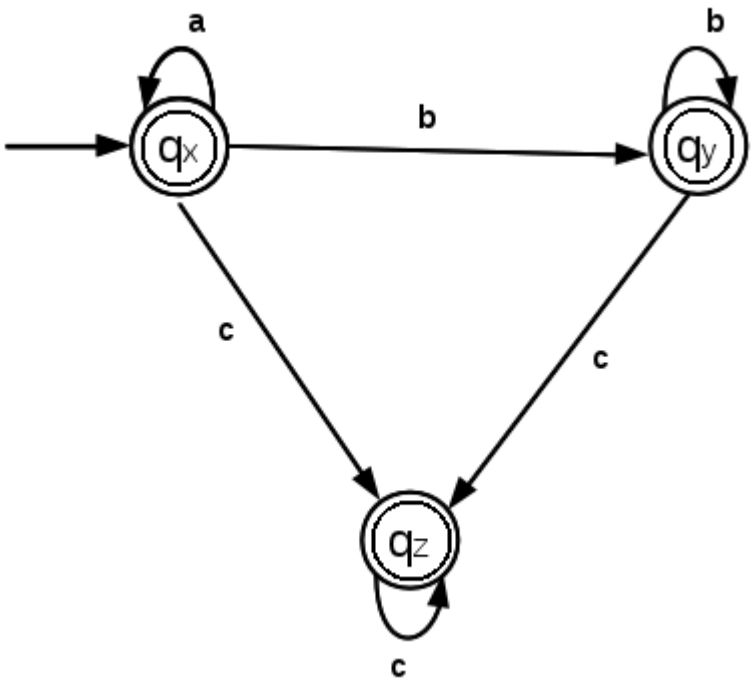
$\{q_1,q_2\}$ as q_y

$\{q_2\}$ as q_z

Then the transition table will become,

Current state	Input Symbol		
	a	b	c
$\longrightarrow *q_x$	q_x	q_y	q_z
$*q_y$	ϕ	q_y	q_z
$*q_z$	ϕ	ϕ	q_z

The transition diagram for the new NFA is,

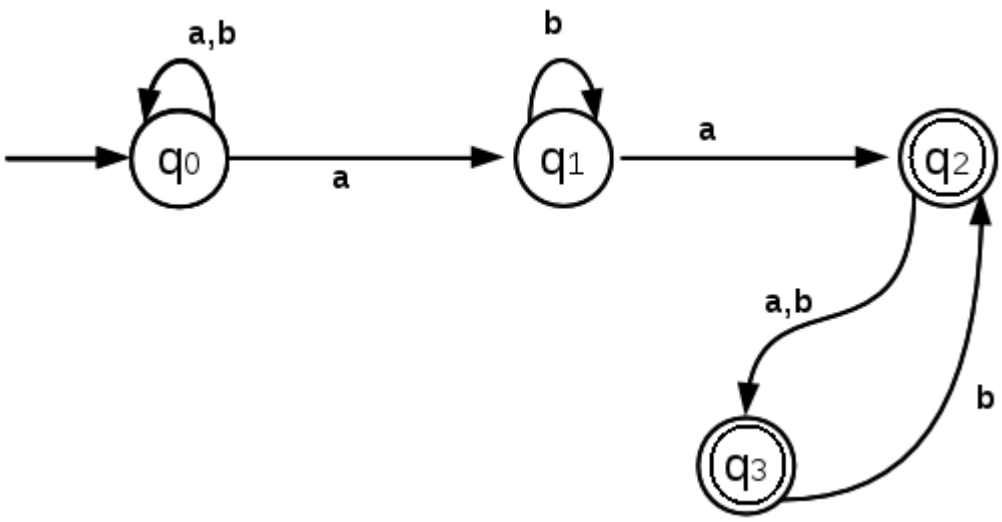


Above is the NFA without epsilon transitions.

Note that above NFA is also a DFA.

Example 2:

Convert the following NFA to DFA.



Step 1: Transform the NFA with Epsilon transitions to NFA without epsilon transitions.

Note that above NFA does not contain any epsilon transitions.

Step 2: Convert the resulting NFA to DFA.

Step a:

Consider the start state q_0,

Seek all the transitions from state q_0 for all input symbols a and b.

We get,

$\delta(q_0, a) = \{q_0, q_1\}$

$\delta(q_0, b) = \{q_0\}$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_0$	$\{q_0, q_1\}$	q_0

Here we got a new state, $\{q_0, q_1\}$

Step b:

Consider the state $\{q_0, q_1\}$,

Seek all the transitions from state $\{q_0, q_1\}$for all input symbols a and b.

We get,

$\delta(\{q_0, q_1\}, a) = \delta(q_0, a) \cup \delta(q_1, a)$

$= \{q_0, q_1\} \cup q_2$

$= \{q_0, q_1, q_2\}$

$\delta(\{q_0, q_1\}, b) = \delta(q_0, b) \cup \delta(q_1, b)$

$= q_0 \cup q_1$

$= \{q_0, q_1\}$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_0$	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$

Here we got a new state, $\{q_0, q_1, q_2\}$

Step c:

Consider the state $\{q_0, q_1, q_2\}$,

Seek all the transitions from state $\{q_0, q_1, q_2\}$for all input symbols a and b.

We get,

$\delta(\{q_0, q_1, q_2\}, a) = \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)$

$= \{q_0, q_1\} \cup q_2 \cup q_3$

$= \{q_0, q_1, q_2, q_3\}$

$\delta(\{q_0, q_1, q_2\}, b) = \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)$

$= q_0 \cup q_1 \cup q_3$

$= \{q_0, q_1, q_3\}$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_0$	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_3\}$

Here we got 2 new states, $\{q_0, q_1, q_2, q_3\}$ and $\{q_0, q_1, q_3\}$.

Step d:

Consider each one of these states, $\{q_0, q_1, q_2, q_3\}$ and $\{q_0, q_1, q_3\}$

Step d.i:

Consider the state $\{q_0, q_1, q_2, q_3\}$,

Seek all the transitions from state $\{q_0, q_1, q_2, q_3\}$for all input symbols a and b.

We get,

$$\delta(\{q_0, q_1, q_2, q_3\}, a) = \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a)$$
$$= \{q_0, q_1\} \cup q_2 \cup q_3 \cup \phi$$
$$= \{q_0, q_1, q_2, q_3\}$$

$$\delta(\{q_0, q_1, q_2, q_3\}, b) = \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b) \cup \delta(q_3, b)$$
$$= q_0 \cup q_1 \cup q_3 \cup q_2$$
$$= \{q_0, q_1, q_2, q_3\}$$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_0$	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_3\}$
$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$

Step d.i:

Consider the state $\{q_0, q_1, q_3\}$,

Seek all the transitions from state $\{q_0, q_1, q_3\}$for all input symbols a and b.

We get,

$$\delta(\{q_0, q_1, q_3\}, a) = \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_3, a)$$
$$= \{q_0, q_1\} \cup q_2 \cup \phi$$
$$= \{q_0, q_1, q_2\}$$

$$\delta(\{q_0, q_1, q_3\}, b) = \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_3, b)$$
$$= q_0 \cup q_1 \cup q_2$$
$$= \{q_0, q_1, q_2\}$$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_0$	$\{q_0, q_1\}$	q_0
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_3\}$
$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$

Now there are no new states generated. above is the transition table corresponding to the new DFA.

In the above DFA, the start state is q_0.

The final states of the DFA are $\{q_0, q_1, q_2\}$,$\{q_0, q_1, q_2, q_3\}$,$\{q_0, q_1, q_3\}$, since they contain at least one final state of the NFA (q_2 or q_3).

Let us say,

q_0 as A,

$\{q_0, q_1\}$ as B,

$\{q_0, q_1, q_2\}$ as C,

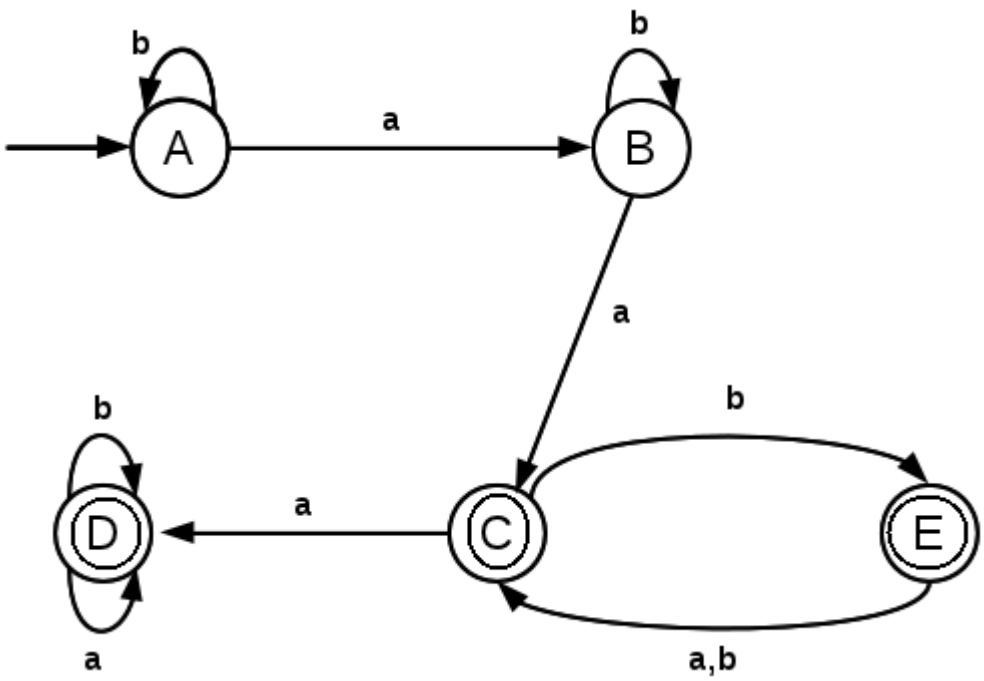
$\{q_0, q_1, q_2, q_3\}$ as D, and

$\{q_0, q_1, q_3\}$ as E

Then the transition table will become,

	Input	symbol
Current State	a	b
$\longrightarrow$A	B	A
B	C	B
* C	D	E
* D	D	D
* E	C	C

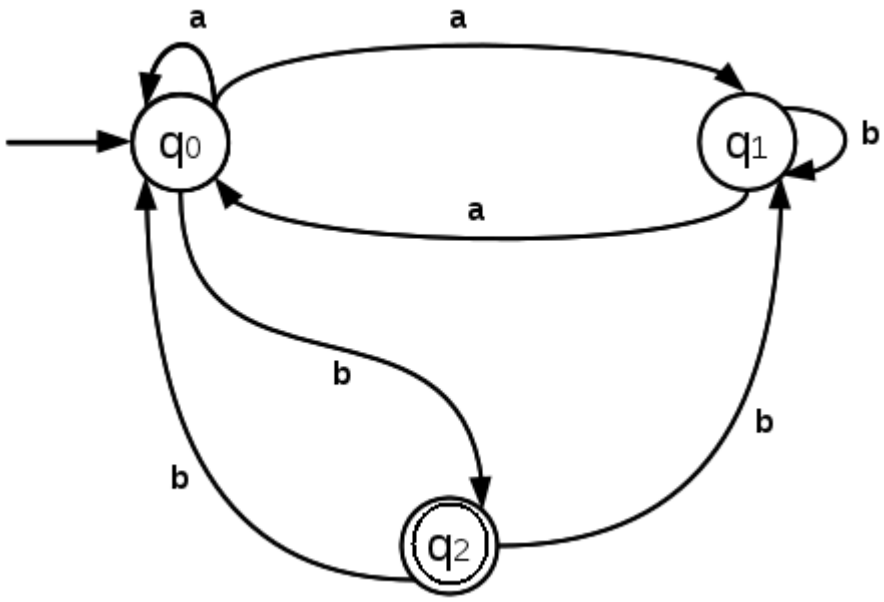
The transition diagram corresponding to the DFA is,



From the above diagram, it is a DFA since, there is only one transition corresponding to an input symbol from a state.

Example 3:

Consider the following NFA,



Step 1: Transform the NFA with Epsilon transitions to NFA without epsilon transitions.

Note that above NFA does not contain any epsilon transitions.

Step 2: Convert the resulting NFA to DFA.

Consider the start state q_0,

Seek all the transitions from state q_0 for all input symbols a and b.

We get,

$\delta(q_0, a) = \{q_0, q_1\}$

$\delta(q_0, b) = \{q_2\}$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_0$	$\{q_0, q_1\}$	q_2

Step b:

Step b.i:

$$\delta(\{q_0, q_1\}, a) = \delta(q_0, a) \cup \delta(q_1, a)$$
$$=\{q_0, q_1\} \cup q_0$$
$$=\{q_0, q_1\}$$
$$\delta(\{q_0, q_1\}, b) = \delta(q_0, b) \cup \delta(q_1, b)$$
$$=q_2 \cup q_1$$
$$=\{q_1, q_2\}$$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_0$	$\{q_0, q_1\}$	q_2
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$

Step b. i. i:

$$\delta(\{q_1, q_2\}, a) = \delta(q_1, a) \cup \delta(q_2, a)$$
$$= q_0 \cup \phi$$
$$= q_0$$
$$\delta(\{q_1, q_2\}, b) = \delta(q_1, b) \cup \delta(q_2, b)$$
$$= q_1 \cup \{q_0, q_1\}$$
$$= \{q_0, q_1\}$$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_0$	$\{q_0, q_1\}$	q_2
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
$\{q_1, q_2\}$	q_0	$\{q_0, q_1\}$

Step b.ii

$$\delta(q_2, a) = \phi$$
$$\delta(q_2, b) = \{q_0 \cup q_1\}$$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_0$	$\{q_0, q_1\}$	q_2
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
$*\{q_1, q_2\}$	q_0	$\{q_0, q_1\}$
$*q_2$	ϕ	$\{q_0, q_1\}$

Thus there are no more new states. The above is the transition table for the new NFA.

The start state is q_0.

The final states are q_2 and $\{q_1, q_2\}$, since they contain the final state, q_2 of the NFA.

Let we say,

$q_0$as A

$\{q_0, q_1\}$ as B

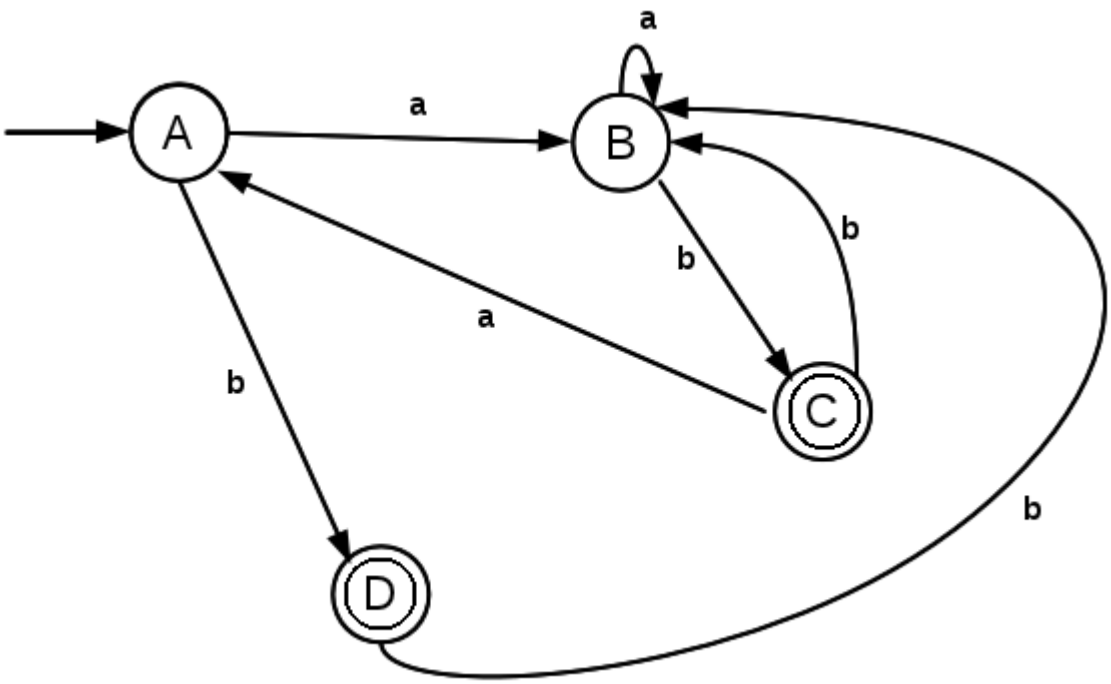
$\{q_1, q_2\}$ as C

q_2 as D.

The new transition table is,

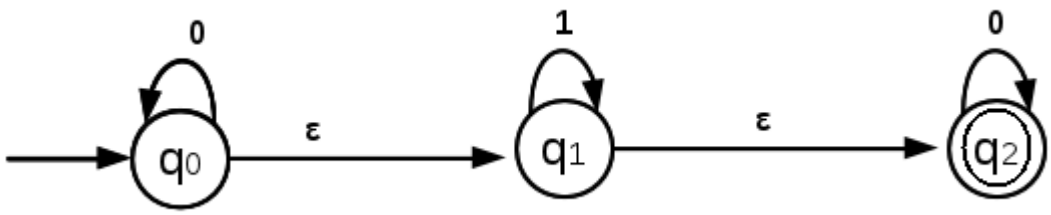
	Input	symbol
Current State	a	b
$\longrightarrow$A	B	D
B	B	C
* C	A	B
* D	ϕ	B

The transition diagram for the DFA is



Example 4:

Consider the NFA,



Convert the above NFA to DFA.

Step 1: Transform the NFA with Epsilon transitions to NFA without epsilon transitions.

Step a:

The states of the new NFA will be $ECLOSE(q_0), ECLOSE(q_1), ECLOSE(q_2)$.

Find the Epsion closure of all states.

$ECLOSE(q_0) = \{q_0, q_1, q_2\}$

$ECLOSE(q_1) = \{q_1, q_2\}$

$ECLOSE(q_2) = \{q_2\}$

The states of the new NFA will be $\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}$.

The start state of the new NFA will be the epsilon closure of the start state q_0.

Thus the start state of the new NFA will be $\{q_0, q_1, q_2\}$

The final states of the new NFA will be those new states that contain the final states of the old NFA.

Thus the final states of the new NFA are $\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}$, since they contain the final state q_2 of the

old NFA.

Next we need to find the transitions of these new states on input symbols 0 and 1.

Consider state $\{q_0, q_1, q_2\}$,

$$\begin{aligned}
\delta'(\{q_0, q_1, q_2\}, 0) &= ECLOSE[\delta(\{q_0, q_1, q_2\}, 0)] \\
&= ECLOSE[\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)] \\
&= ECLOSE[q_0 \cup \phi \cup q_2] \\
&= ECLOSE[q_0, q_2] \\
&= \{q_0, q_1, q_2\} \\
\delta'(\{q_0, q_1, q_2\}, 1) &= ECLOSE[\delta(\{q_0, q_1, q_2\}, 1)] \\
&= ECLOSE[\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)] \\
&= ECLOSE[\phi \cup q_1 \cup \phi] \\
&= ECLOSE[q_1] \\
&= \{q_1, q_2\} \\
\delta'(\{q_1, q_2\}, 0) &= ECLOSE[\delta(\{q_1, q_2\}, 0)] \\
&= ECLOSE[\delta(q_1, 0) \cup \delta(q_2, 0)] \\
&= ECLOSE[\phi \cup q_2] \\
&= ECLOSE[q_2] \\
&= \{q_2\} \\
\delta'(\{q_1, q_2\}, 1) &= ECLOSE[\delta(\{q_1, q_2\}, 1)] \\
&= ECLOSE[\delta(q_1, 1) \cup \delta(q_2, 1)] \\
&= ECLOSE[q_1 \cup \phi] \\
&= ECLOSE[q_1] \\
&= \{q_1, q_2\} \\
\delta'(q_2, 0) &= ECLOSE[\delta(q_2, 0)] \\
&= ECLOSE[q_2] \\
&= \{q_2\} \\
\delta'(q_2, 1) &= ECLOSE[\delta(q_2, 1)] \\
&= ECLOSE[\phi] \\
&= \phi
\end{aligned}$$

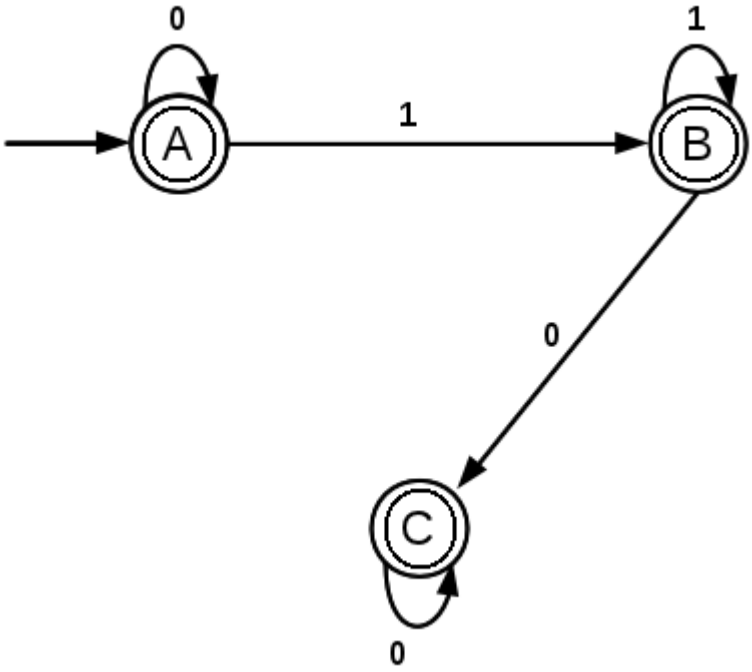
Let us say,

$\{q_0, q_1, q_2\}$ as A,

$\{q_1, q_2\}$ as B, and

$\{q_2\}$ as C.

Then the NFA without epsilon transitions is,



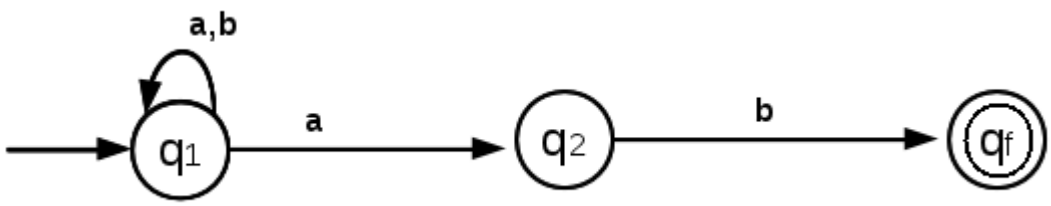
The above is an NFA without epsilon transitions.

Step 2: Convert the resulting NFA to DFA.

Note that above NFA is also a DFA, since from a state there is only one transition corresponding to an input symbol.

Example 5:

Consider the following NFA,



Convert this NFA to DFA.

Step 1: Transform the NFA with Epsilon transitions to NFA without epsilon transitions.

Above NFA does not contain epsilon transitions. So we need to eliminate epsilon transitions.

Step 2: Transform above NFA to DFA.

Begin from the start state, q_1,

$\delta(q_1, a) = \{q_1, q_2\}$

$\delta(q_1, b) = \{q_1\}$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_1$	$\{q_1, q_2\}$	q_1

Step b:

$$\delta(\{q_1, q_2\}, a) = \delta(q_1, a) \cup \delta(q_2, a)$$
$$=\{q_1, q_2\} \cup \phi$$
$$=\{q_1, q_2\}$$
$$\delta(\{q_1, q_2\}, b) = \delta(q_1, b) \cup \delta(q_2, b)$$
$$=q_1 \cup q_f$$
$$=\{q_1, q_f\}$$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_1$	$\{q_1, q_2\}$	q_1
$\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_1, q_f\}$

Step c:

$$\delta(\{q_1, q_f\}, a) = \delta(q_1, a) \cup \delta(q_f, a)$$
$$=\{q_1, q_2\} \cup \phi$$
$$=\{q_1, q_2\}$$
$$\delta(\{q_1, q_f\}, b) = \delta(q_1, b) \cup \delta(q_f, b)$$
$$=q_1 \cup \phi$$
$$=\{q_1\}$$

Now we got,

	Input	symbol
Current State	a	b
$\longrightarrow q_1$	$\{q_1, q_2\}$	q_1
$\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_1, q_f\}$
$*\{q_1, q_f\}$	$\{q_1, q_2\}$	q_1

The start state is q_1.

The final state is $\{q_1, q_f\}$.

Let us say,

q_1 as A,

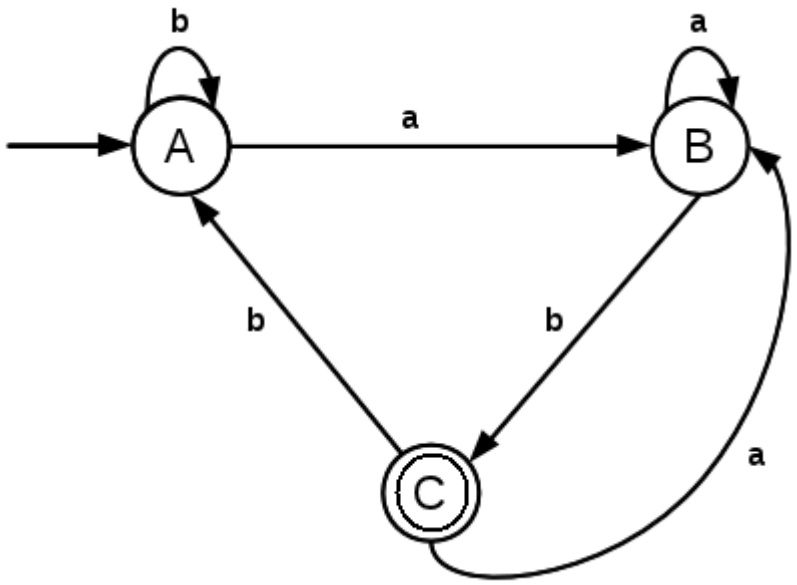
$\{q_1, q_2\}$ as B,

$\{q_1, q_f\}$ as C.

Then the transition table will become,

Current State	Input a	symbol b
⟶A	B	A
B	B	C
* C	B	A

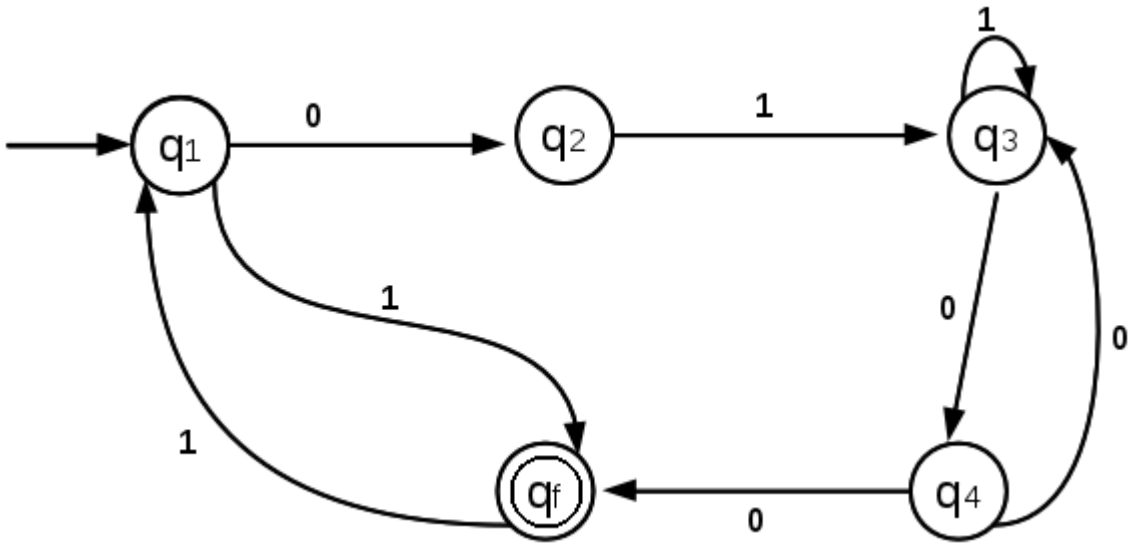
The transition diagram is,



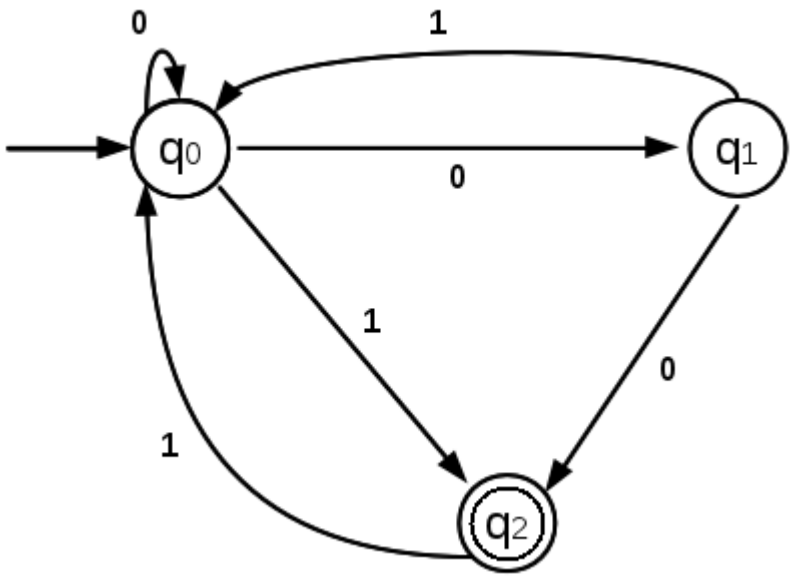
The above is a DFA, since there is only one transition corresponding to an input symbol from every state.

Exercises:

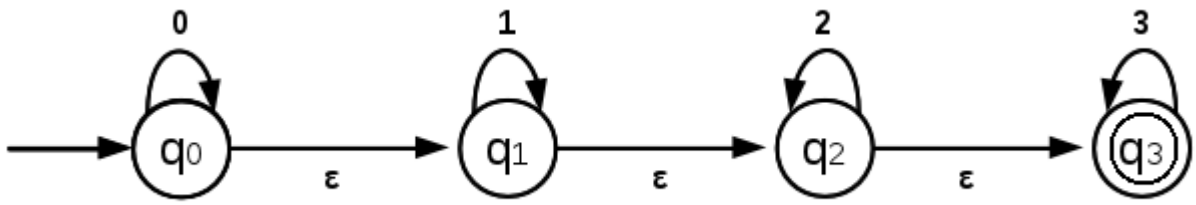
1. Convert the following NFA to DFA.



2. Convert the following NFA to DFA.



3. Convert the following NFA to DFA.



Part IV. Minimization of DFA

For a given language, many DFAs may exist that accept it. The DFA we produce from an NFA may contain many dead states, inaccessible states and indistinguishable states. all these unnecessary states can be eliminated from a DFA through a process called minimization.

For practical applications, it is desirable that number of states in the DFA is minimum.

The algorithm for minimising a DFA is as follows:

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

Step 2: Partition the remaining states into blocks so that all states in the same block are equivalent, and no pair of states from different blocks are equivalent.

The process is demonstrated using an example:

Example 1:

Minimise the following DFA,

	Input	symbol
Current State	a	b
$\longrightarrow q_0$	q_5	q_1
q_1	q_2	q_6
$*q_2$	q_2	q_0
q_4	q_5	q_7
q_5	q_6	q_2
q_6	q_4	q_6
q_7	q_2	q_6
q_3	q_6	q_2

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 state q_3 cannot be reached. So remove the corresponding to q_3 from the transition table.

Now the new transition table is,

Current State	Input symbol a	Input symbol b
$\longrightarrow q_0$	q_5	q_1
q_1	q_2	q_6
$*q_2$	q_2	q_0
q_4	q_5	q_7
q_5	q_6	q_2
q_6	q_4	q_6
q_7	q_2	q_6

Step 2:

Divide the rows of the transition table into 2 sets as,

1. One set containing only those rows which starts from non-final states.

Set 1

q_0	q_5	q_1
q_1	q_2	q_6
q_4	q_5	q_7
q_5	q_6	q_2
q_6	q_4	q_6
q_7	q_2	q_6

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

Set 2

$*q_2$	q_2	q_0

Step 3a. Consider Set 1.

Find out the similar rows:

q_0	q_5	q_1	Row 1
q_1	q_2	q_6	Row 2
q_4	q_5	q_7	Row 3
q_5	q_6	q_2	Row 4
q_6	q_4	q_6	Row 5
q_7	q_2	q_6	Row 6

Row 2 and Row 6 are similar, since q_1 and q_7 transit to same states on inputs a and b.

So remove one of them (for instance, q_7) and replace q_7 with q_1 in the rest.

We get,

Set 1

q_0	q_5	q_1	Row 1
q_1	q_2	q_6	Row 2
q_4	q_5	q_1	Row 3
q_5	q_6	q_2	Row 4
q_6	q_4	q_6	Row 5

Now Row 1 and Row 3 are similar. So remove one of them (for instance, q_4) and replace q_4 with q_0 in the rest.

We get,

Set 1

q_0	q_5	q_1	Row 1
q_1	q_2	q_6	Row 2
q_5	q_6	q_2	Row 3
q_6	q_0	q_6	Row 4

Now there are no more similar rows.

Step 3b.

Consider Set 2,

Set 2

$*q_2$	q_2	q_0

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,

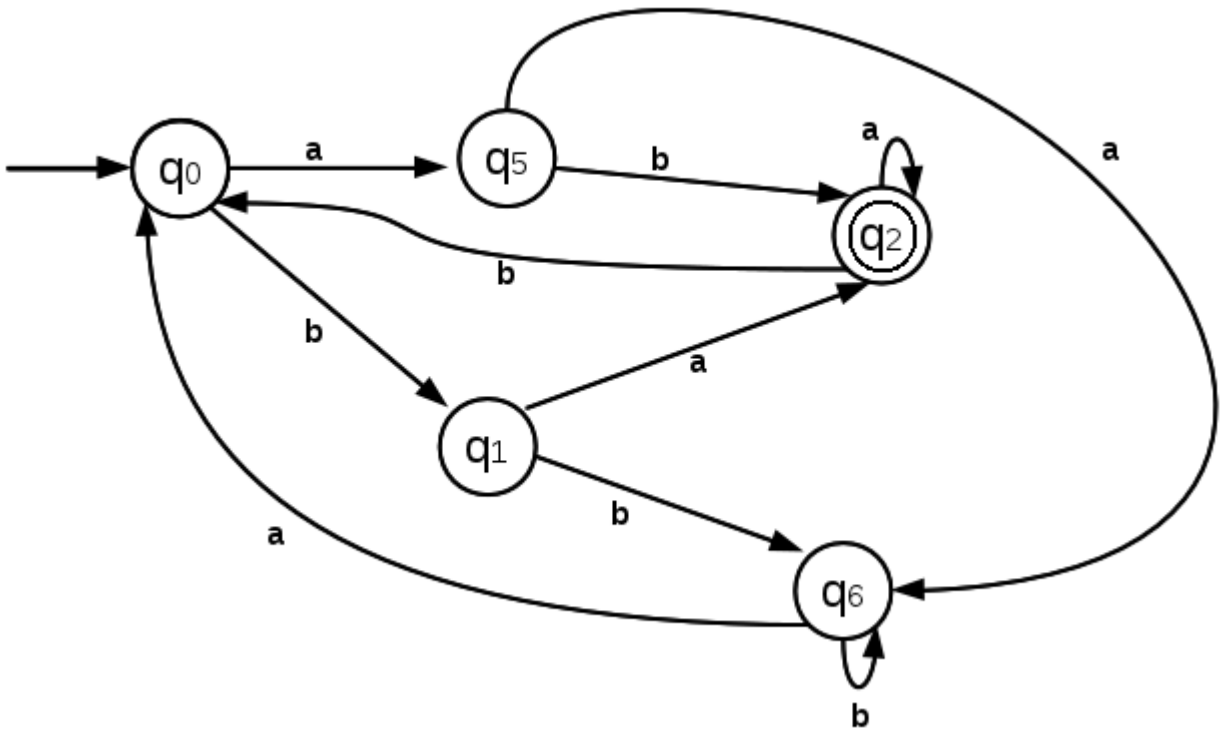
q_0	q_5	q_1	Row 1
q_1	q_2	q_6	Row 2
q_5	q_6	q_2	Row 3
q_6	q_0	q_6	Row 4
$*q_2$	q_2	q_0	Row 5

The DFA corresponding to this is,

	Input	symbol
Current State	a	b
$\longrightarrow q_0$	q_5	q_1
q_1	q_2	q_6
q_5	q_6	q_2
q_6	q_0	q_6
$*q_2$	q_2	q_0

Now this is a minimised DFA.

Transition diagram is,



Example 2:

Minimise the following DFA,

	Input	symbol
Current State	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.

We get,

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

Step 2: Divide the rows of the transition table into 2 sets as,

1. One set containing only those rows which starts from non-final states.

   Set 1

q_0	q_1	q_3
q_1	q_0	q_3

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

   Set 2

$*q_3$	q_5	q_5
$*q_5$	q_5	q_5

Step 3a. Consider Set 1.

Find out the similar rows:

Set 1

q_0	q_1	q_3	Row 1
q_1	q_0	q_3	Row 2

There are no similar rows.

Step 3b. Consider Set 2.

Find out the similar rows:

Set 2

$*q_3$	q_5	q_5	Row 1
$*q_5$	q_5	q_5	Row 2

Row 1 and Row 2 are similar, since q_3 and q_5 transit to same states on inputs 0 and 1.

So remove one of them (for instance, q_5) and replace q_5 with q_3 in the rest.

We get,

$*q_3$	q_3	q_3	Row 1

Step 4:

Combine Set 1 and Set 2

We get,

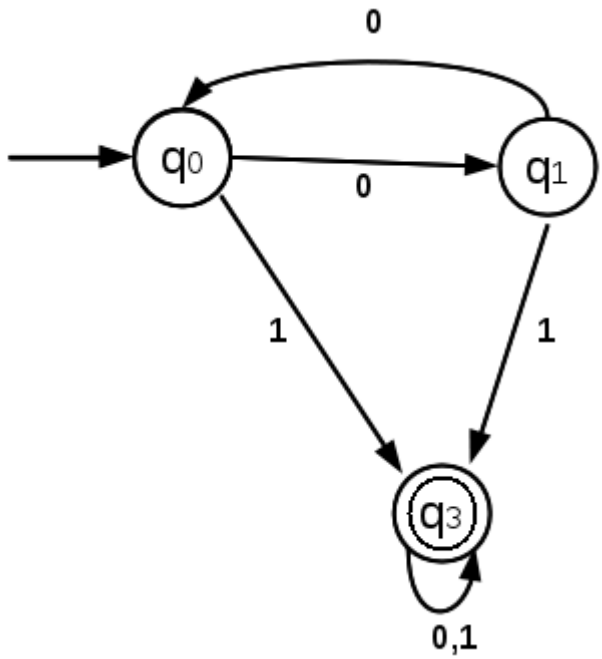
q_0	q_1	q_3	Row 1
q_1	q_0	q_3	Row 2
$*q_3$	q_3	q_3	Row 3

The DFA corresponding to this is,

Current State	Input symbol	
	a	b
$\longrightarrow q_0$	q_1	q_3
q_1	q_0	q_3
$*q_3$	q_3	q_3

Now this is a minimised DFA.

Transition diagram is,



Exercise:

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

2. Minimise the following DFA represnted as transition table,

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

3. Minimise the following DFA represnted as transition table,

Current State	Input symbol 0	Input symbol 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

4. Minimise the following DFA represnted as transition table,

Current State	Input symbol 0	Input symbol 1
$\longrightarrow q_0$	q_1	q_5
q_1	q_6	q_2
$*q_2$	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

5. Minimise the following DFA represnted as transition table,

Current State	Input symbol	
	a	b
$\longrightarrow q_0$	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
$*q_3$	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

6. Minimise the following DFA represnted as transition table,

Current State	Input symbol	
	a	b
$\longrightarrow q_0$	q_1	q_2
q_1	q_1	q_3
q_2	q_3	q_4
q_3	q_1	q_5
q_4	q_4	q_2
$*q_5$	q_6	q_6

7. Minimise the following DFA represnted as transition table,

Current State	Input symbol	
	a	b
$\longrightarrow q_0$	q_1	q_2
q_1	q_4	q_3
q_2	q_4	q_3
$*q_3$	q_5	q_6
$*q_4$	q_7	q_6
q_5	q_3	q_6
q_6	q_6	q_6
q_7	q_4	q_6

Part V. Regular Expressions

We found in earlier sections that a language can be described by using an NFA or DFA. But a more concise way of describing a lnguage is by using regular expressions.

Thus one way of describing a language is via the notion of regular expressions.

For example,

ab,

a,

ab^*,

$(ab)^+$,

$abcd^*$,

x/y,

a/bcd^*,

$letter(letter/digit)^*$,

$a(b/c)d^*$

are different examples of regular expressions.

We will learn the meaning of all these symbols later.

A regular expression generates a set of strings.

For example, the regular expression, a^* generates the strings, $\varepsilon, a, aa, aaa, aaaa.....$

The regular expression, $a(b/a)$ generates the strings ab, ba.

7 Definition of a Regular Expression

Let $\sum$ denotes the character set, ie. a, b, c....z.

The regular expression is defined by the following rules:

Rule 1: Every character of $\sum$ is a regular expression.

For example, a is a regular expression, b is a regular expression, c is a regular expression and so on.

Rule 2: Null string, ε is a regular expression.

Rule 3: If R1 and R2 are regular expressions, then R1 R2 is also a regular expression.

For example, let R1 denotes abc and

let R2 denotes $(pq)^*$,

then $abc(pq)^*$ is also a regular expression.

Rule 4: If R1 and R2 are regular expressions, then R1 / R2 is also a regular expression.

For example, let R1 denotes $(ab)^+$ and

let R2 denotes cd,

then $(ab)^+/(cd)$ is also a regular expression.

Rule 5: If R is a regular expression, then (R) is also a regular expression.

For example, let R denotes adc,

then (adc) is also a regular expression.

Rule 6: If R is a regular expression, then R^* is also a regular expression.

For example, let R denotes ab,

then $(ab)^*$ is also a regular expression.

Rule 7: If R is a regular expression, then R^+ is also a regular expression.

For example, let R denotes pqr,

then $(pqr)^+$ is also a regular expression.

Rule 8: Any combination of the preceding rules is also a regular expression.

For example, let R1 denotes $(ab)^*$,

let R2 denotes d^+,

let R3 denotes pq,

then $((ab)^*/(d^+pq)^*)^*$ is also a regular expression.

Strings Generated from a Regular Expression

A regular expression generates a set of strings.

1. A regular expression, a^* means zero or more repretitions of a.

   So the strings generated from this regular expression are,

   ε,

   a,

   aa,

   aaa,

   aaaa, and so on.

2. A regular expression, p^+ means one or more repretitions of a.

   So the strings generated from this regular expression are,

   p,

   pp,

   ppp,

   pppp, and so on.

3. A regular expression, a/b means the string can be either a or b.

So the strings generated from this regular expression are,

a,

b.

4. A regular expression, (a/b)c means a string that starts with either a or b and will be followed by c.

So the strings generated from this regular expression are,

ac,

bc

5. A regular expression, (a/b)c(d/e) means that a string will begin with either a or b, followed by c, and will end with either d or e.

So the strings generated from this regular expression are,

acd,

ace,

bcd,

bce.

Following shows some examples for the strings generated from regular expressions:

Regular Expression	Strings Generated
a	a
$a/b/c$	a, b, c
$(ab)/(ba)$	ab, ba
$ab(d/e)$	abd, abe
$(aa)^*$	$\varepsilon, aa, aaaa, aaaaaa, ...$
a^+	$a, aa, aaa, aaaa, ...$
$(a/b)(c/d/e)u$	$acu, adu, aeu, bcu, bdu, beu$
abc^*de	$abde, abcde, abccde, abcccde, ...$
$(a/b)c^*$	$a, ac, acc, accc, ..., b, bc, bcc, ...$
$(abc)^*$	$\varepsilon, abc, abcabc, abcabcabc, ...$
$(0/(11))2((33)/1)$	$0233, 021, 11233, 1121$
a^*a	$a, aa, aaa, aaaa, ...$
aa^*	$a, aa, aaa, aaaa, ...$
$(abc)k^*$	$abc, abck, abckk, abckkk, ...$
$(ab)^+$	$ab, abab, ababab, ...$

Example 1:

Write regular expression for the language,

L={aa, aaaa, aaaaaa, aaaaaaaa, ...}

Here aa repeats a number of times starting from 1. So the regular expression is,

$$(aa)^+$$

Example 2:

Write regular expression for the language,

L={0, 1, 2}

The regular expression is,

$$0/1/2$$

Example 3:

Write regular expression for the language, L over {0,1}, such that every string in L begins with 00 and ends with 11.

The regular expression is,

$$00(0/1)^*11.$$

Example 4:

Write regular expression for the language, L over {0,1}, such that every string in L contains alternating 0s and 1s.

The regular expression is,

$$(0(10)^*)/(0(10)^*1)/(1(01)^*)/(1(01)^*0)$$

Example 5:

Write regular expression for the language, L over {0,1}, such that every string in L ends with 11.

The regular expression is,

$$(0/1)^*11$$

Example 6:

Write regular expression for the language, L over {a,b,c}, such that every string in L contains a substring ccc.

The regular expression is,

$$(a/b/c)^*ccc(a/b/c)^*$$

Example 7:

Write regular expression for the language, $L = \{a^n b^m | n \geq 4, m \leq 3\}$

This means that language contains the set of strings such that every string starts with at least 4 a's and end with at most 3 b's.

The regular expression is,

$$a^4a^*(\varepsilon/b/b^2/b^3)$$

Example 8:

Write regular expression for the language, $L = \{a^nb^m|$(n+m) is even }

n+m can be even in two cases.

Case 1: Both n and m are even,

The regular expression is,

$$(aa)^*(bb)^*$$

Case 2: Both n and m are odd,

The regular expression is,

$$(aa)^*a(bb)^*b$$

Then the regular expression for L is,

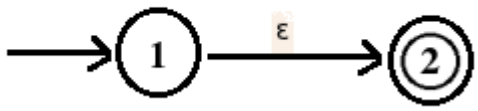
$$((aa)^*(bb)^*)/((aa)^*a(bb)^*b)$$

8 Transforming Regular Expressions to Finite Automata

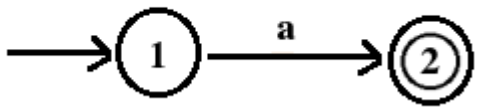
For every regular expression, a corresponding finite automata exist.

Here we will learn how to construct a finite automation corresponding to a regular expression.

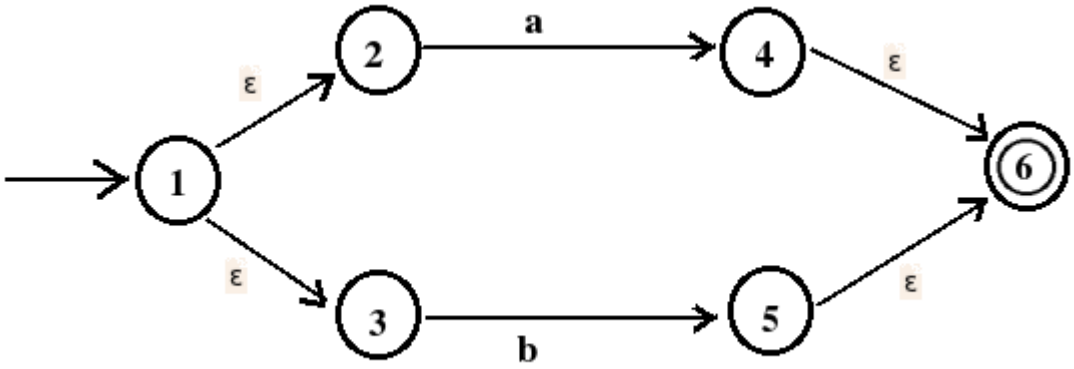
1. The NFA corresponding to regular expression, ε is



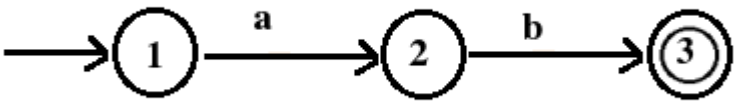
2. The NFA corresponding to regular expression, a is



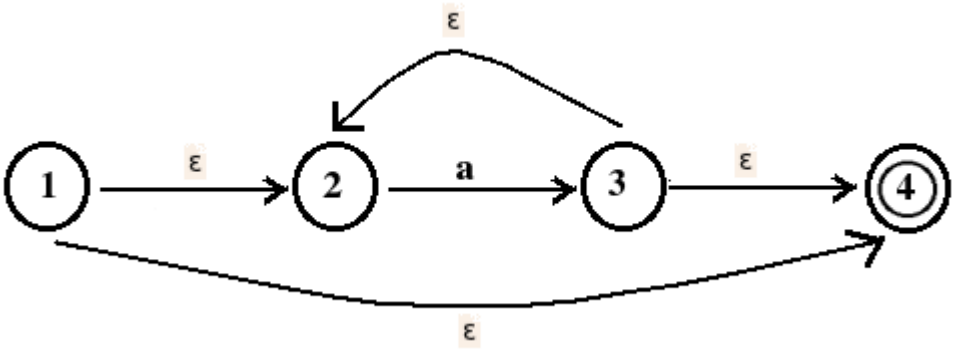
3. The NFA corresponding to regular expression, a | b is



4. The NFA corresponding to regular expression, ab is



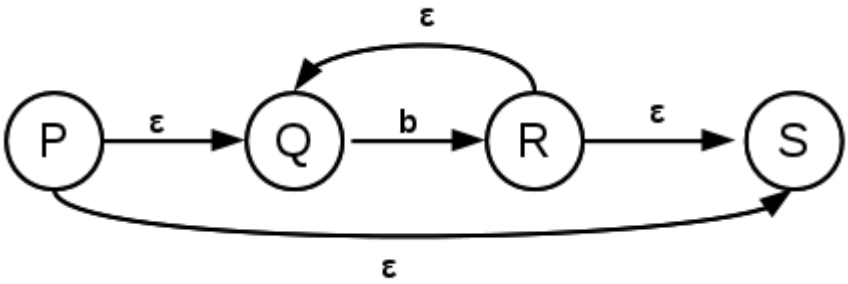
5. The NFA corresponding to regular expression, a^* is



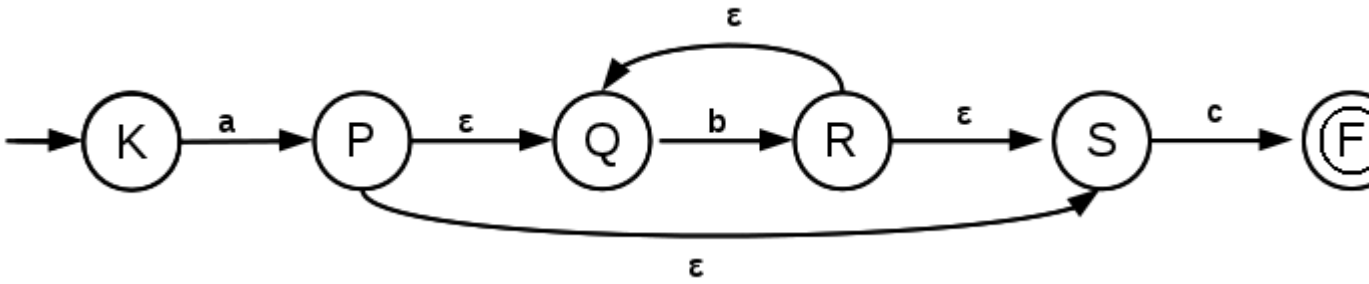
Example 1:

Construct a finite automation for the regular expression, ab^*c.

The finite automation corresponding to b^* is



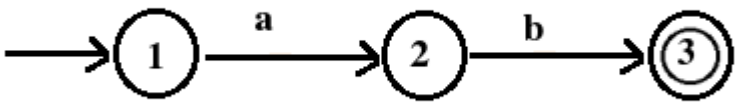
Then the finite automation corresponding to ab^*c is



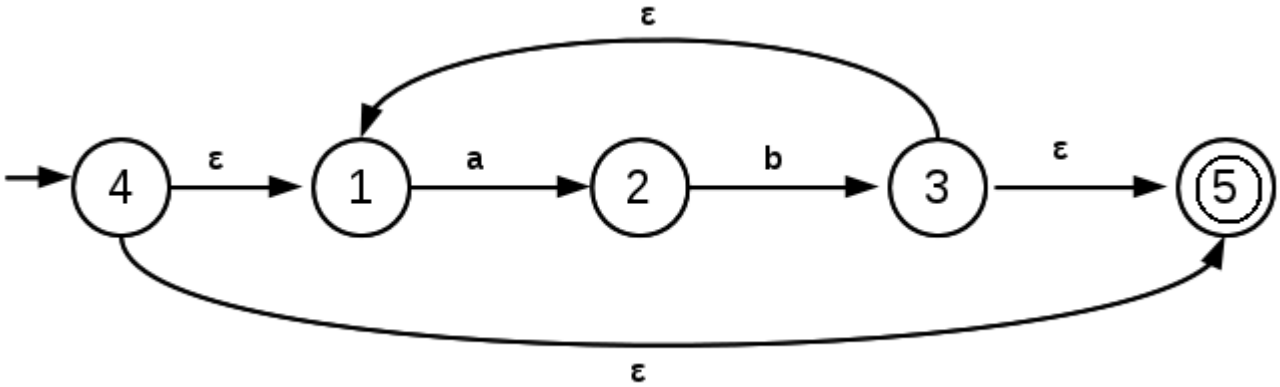
Example 2:

Construct a finite automation for the regular expression, $(ab)^*$.

The finite automation corresponding to ab is



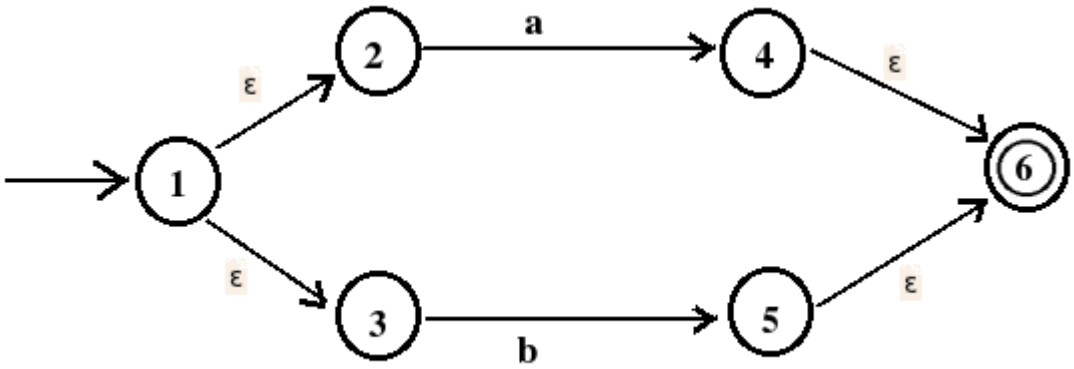
The finite automation corresponding to $(ab)^*$ is,



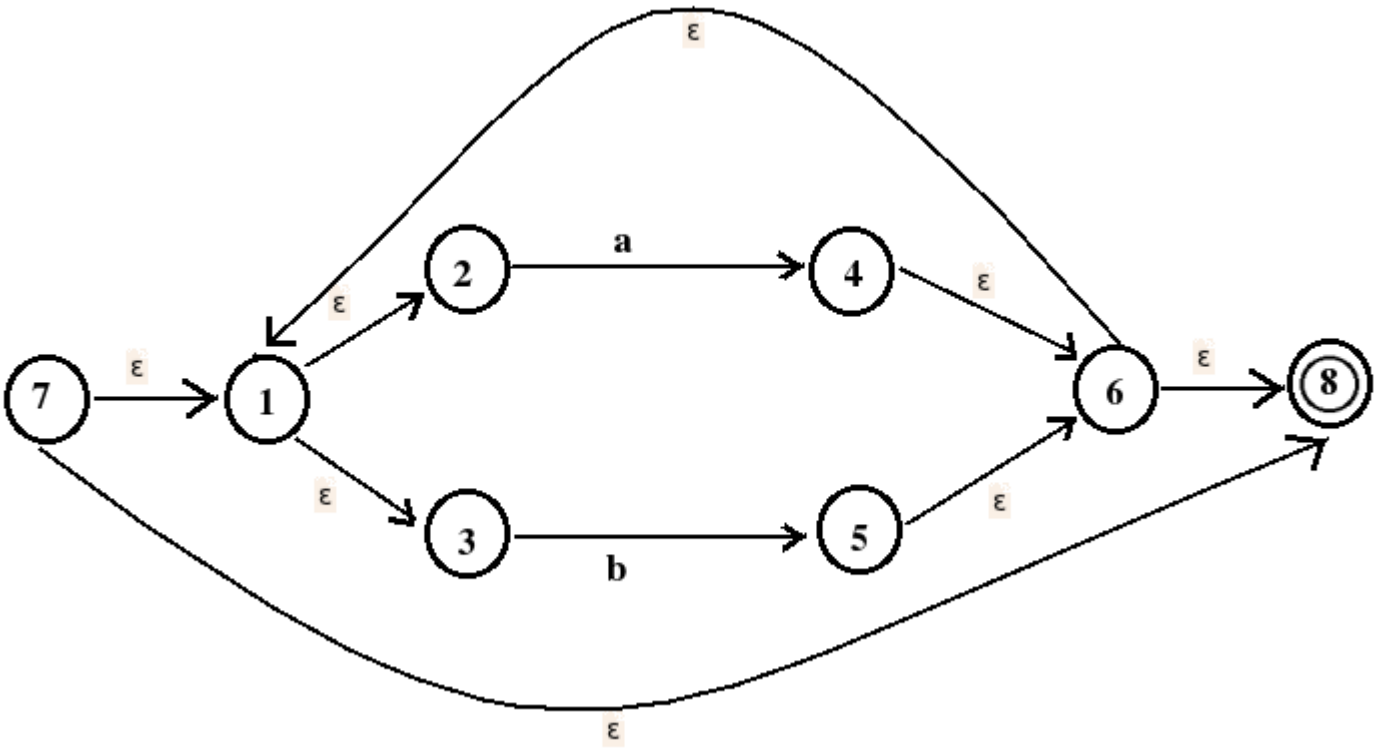
Example 3:

Find the NFA corresponding to regular expression $(a|b)^*a$.

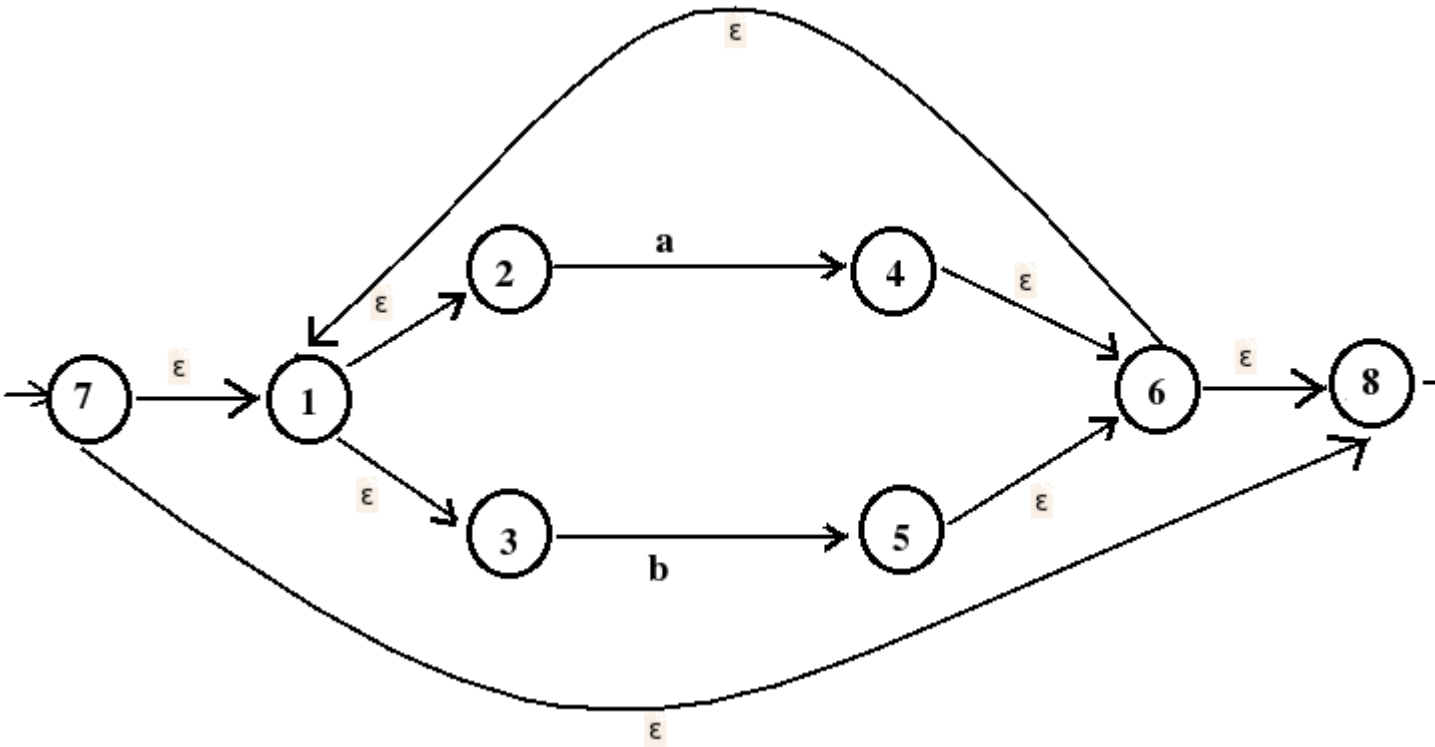
The NFA for a|b is



The NFA corresponding to $(a|b)^*$is



The NFA corresponding to $(a|b)^*a$ is



9 DFA to Regular Expression Conversion

Note: In this section, we use + symbol instead of / operator. For example, a/b is written as a+b.

We can find out the regular expression for every deterministic finite automata (DFA).

Arden's Theorem

This theorem is used to find the regular expression for the given DFA.

Let P and Q are two regular expressions,

then the equation, $R = Q + RP$ has the solution,

$R = QP^*$.

Proof:

$Q + RP = Q + (QP^*)P$,

since $R = QP^*$.

$= Q(\varepsilon + P^*P)$

$= Q[\varepsilon/(P^*P)]$

$= QP^*$

$= R$

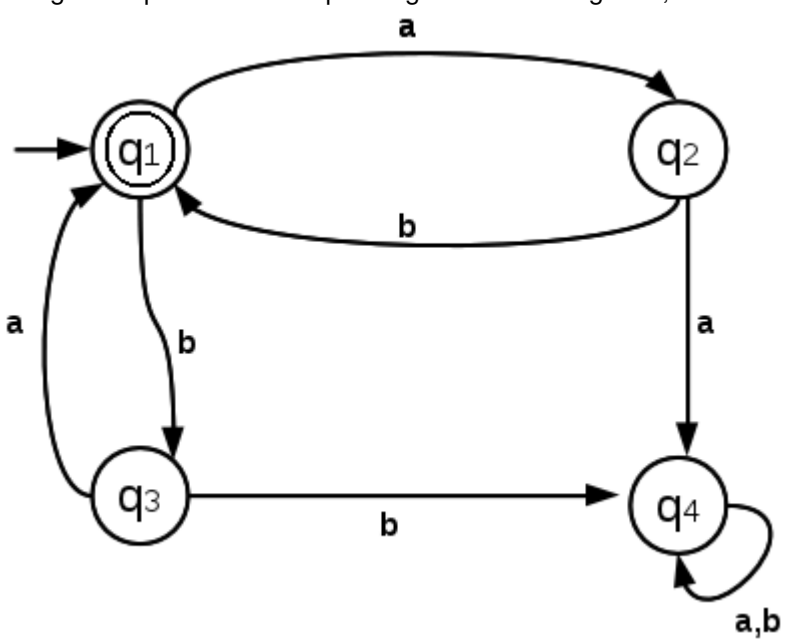
Thus if we are given, $R = Q + RP$, we may write it as,

$R = QP^*$

The process is demonstrated using the following examples:

Example 1:

Find the regular expression corresponding to the following DFA,



Here we write equations for every state.

We write,

$q_1 = q_2b + q_3a + \varepsilon$

The term q_2b because there is an arrow from q_2 to $q_1$on input symbol b.

The term q_3a because there is an arrow from q_3 to $q_1$on input symbol a.

The term ε because $q_1$is the start state.

$q_2 = q_1 a$

The term $q_1 a$ because there is an arrow from q_1 to $q_2$on input symbol a.

$q_3 = q_1 b$

The term $q_1 b$ because there is an arrow from q_1 to $q_3$on input symbol b.

$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$

The term $q_2 a$ because there is an arrow from q_2 to $q_4$on input symbol a.

The term $q_3 b$ because there is an arrow from q_3 to $q_4$on input symbol b.

The term $q_4 b$ because there is an arrow from q_4 to $q_4$on input symbol b.

The final state is q_1.

Putting $q_2$and q_3 in the first equation (corresponding to the final state), we get,

$q_1 = q_1 ab + q_1 ba + \varepsilon$

$q_1 = q_1 (ab + ba) + \varepsilon$

$q_1 = \varepsilon + q_1 (ab + ba)$

From Arden's theorem,

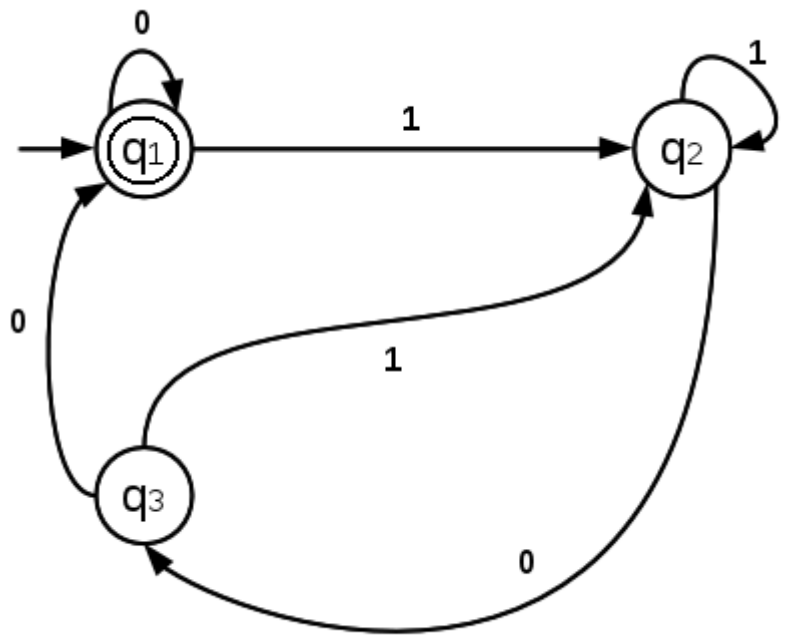
$q_1 = \varepsilon (ab + ba)^*$

$q_1 = (ab + ba)^*$

So the regular expression is, $((ab)/(ba))^*$

Example 2;

Find the regular expression corresponding to the following DFA,



Let us write the equations

$q_1 = q_1 0 + q_3 0 + \varepsilon$

$q_2 = q_1 1 + q_2 1 + q_3 1$

$q_3 = q_2 0$

$q_2 = q_1 1 + q_2 1 + q_3 1$

$q_2 = q_1 1 + q_2 1 + (q_2 0)1$

$q_2 = q_1 1 + q_2 (1 + 01)$

$q_2 = q_1 1(1 + 01)^*$ (From Arden's theorem)

Consider the equation corresponding to final state,

$q_1 = q_1 0 + q_3 0 + \varepsilon$

$q_1 = q_1 0 + (q_2 0)0 + \varepsilon$

$q_1 = q_1 0 + (q_1 1(1 + 01)^*)0)0 + \varepsilon$

$q_1 = q_1 (0 + 1(1 + 01)^*)00) + \varepsilon$

$q_1 = \varepsilon + q_1 (0 + 1(1 + 01)^*)00)$

$q_1 = \varepsilon(0 + 1(1 + 01)^*)00)^*$

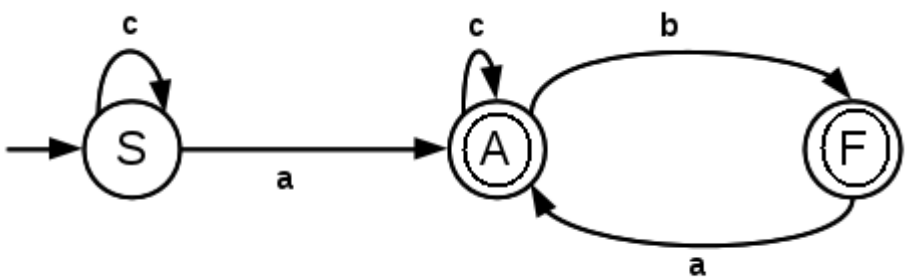
$q_1 = (0 + 1(1 + 01)^*)00)^*$

Since q_1 is a final state, the regular expression is,

$(0/(1(1 + 01)^*)00))^*$

Example 3:

Find the regular expression corresponding to the following DFA,



The equations are,

$S = Sc + \varepsilon$

$A = Ac + Fa + Sa$

$F = Ab$

$S = Sc + \varepsilon$

$S = \varepsilon + Sc$

$S = \varepsilon c^*$

$S = c^*$

$$A = Ac + Fa + Sa$$

$$A = Ac + Fa + c^*a$$

$$A = Ac + Aba + c^*a$$

$$A = A(c + ba) + c^*a$$

$$A = c^*a + A(c + ba)$$

$$A = c^*a(c + ba)^*$$

$$F = Ab$$

$$F = (c^*a(c + ba)^*)b$$

There are two final states in the DFA, they are A and F.

The regular expression corrresponding to A is $c^*a(c + ba)^*$

The regular expression corrresponding to F is $(c^*a(c + ba)^*)b$

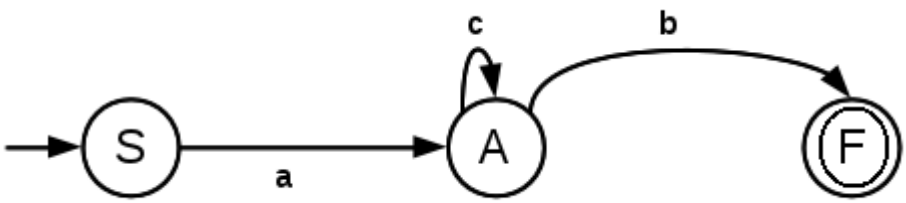
Then the regular expression for the DFA is

$$c^*a(c + ba)^* + (c^*a(c + ba)^*)b$$

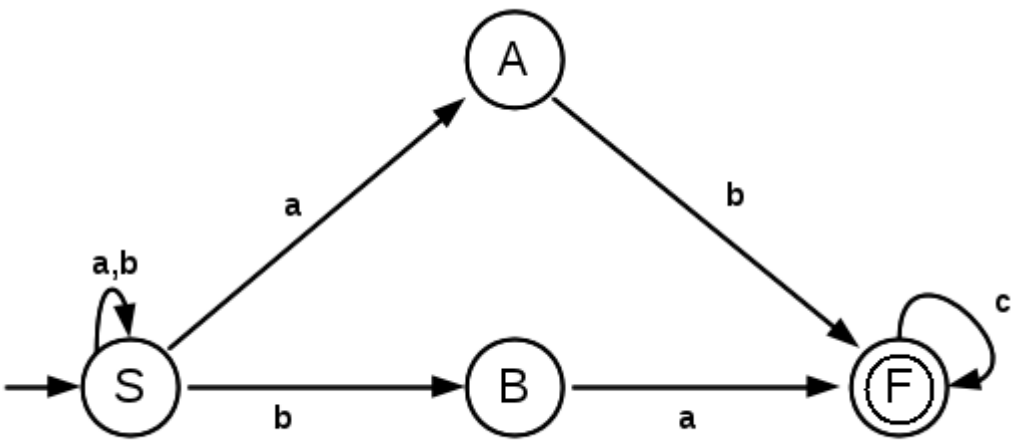
That is, $[c^*a(c/(ba))^*]/[(c^*a(c/(ba))^*)b]$

Exercises:

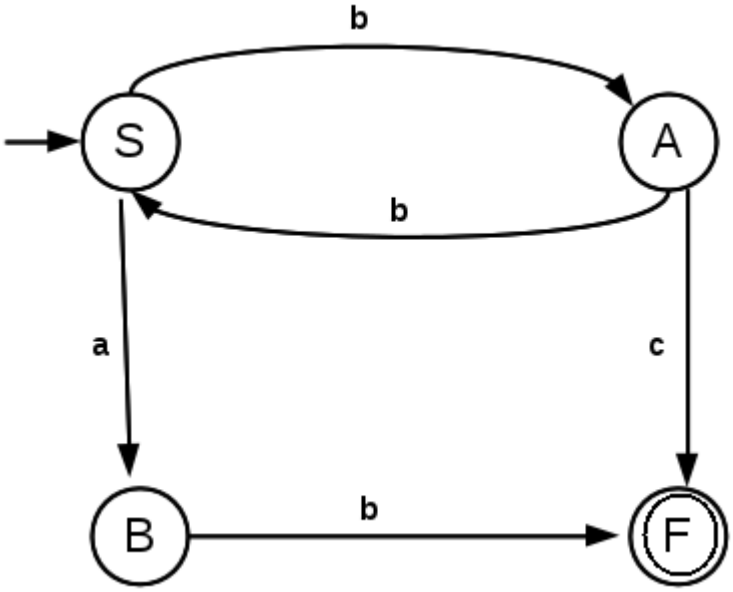
1. Find the regular expression corresponding to the following DFA,



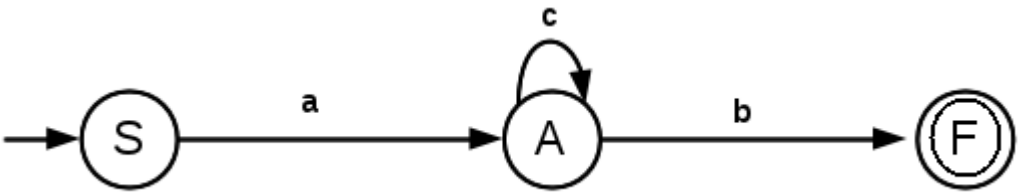
2. Find the regular expression corresponding to the following DFA,



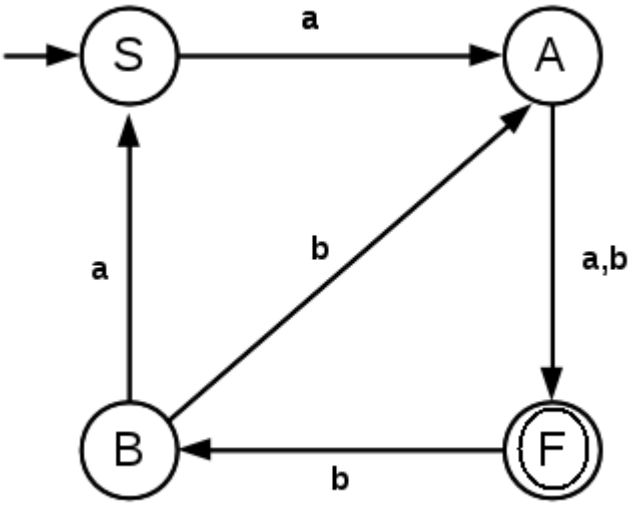
3. Find the regular expression corresponding to the following DFA,



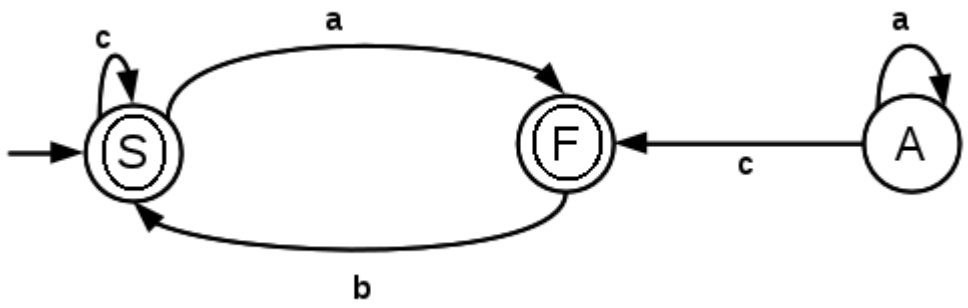
4. Find the regular expression corresponding to the following DFA,



5. Find the regular expression corresponding to the following DFA,

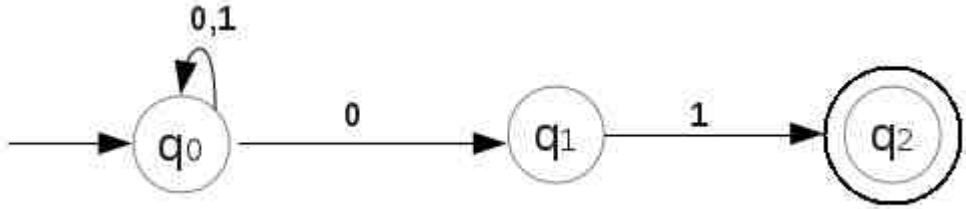


6. Find the regular expression corresponding to the following DFA,


c
a
a
S
F
A
c
b

Part VI. Automata with Output

Consider the following finite automation,



Check whether the string 01101 is accepted by the above automation.

$\delta(q_0, \underline{0}1101) = q_0$

$\delta(q_0, 0\underline{1}101) = q_0$

$\delta(q_0, 01\underline{1}01) = q_0$

$\delta(q_0, 011\underline{0}1) = q_1$

$\delta(q_1, 0110\underline{1}) = q_2$

q_2 is a final state. So the string 01101 is accepted by the above NFA.

Here note the output we obtained from the automation. THe output is that the string 01101 is accepted by the automation.

Thus in our previous discussion on finite automata, there are only two possible outputs, ie, accept or reject.

Thus here the task done by the machine is simply recognize a language. But machines can do more than this.

So here we learn finite automata with output capabilities.

We will learn two models of finite automata with output capabilities. They are,

Moore Machine, and

Mealy Machine.

10 Moore Machine

Moore machine is a finite automation.

In this finite automation, output is associated with every state.

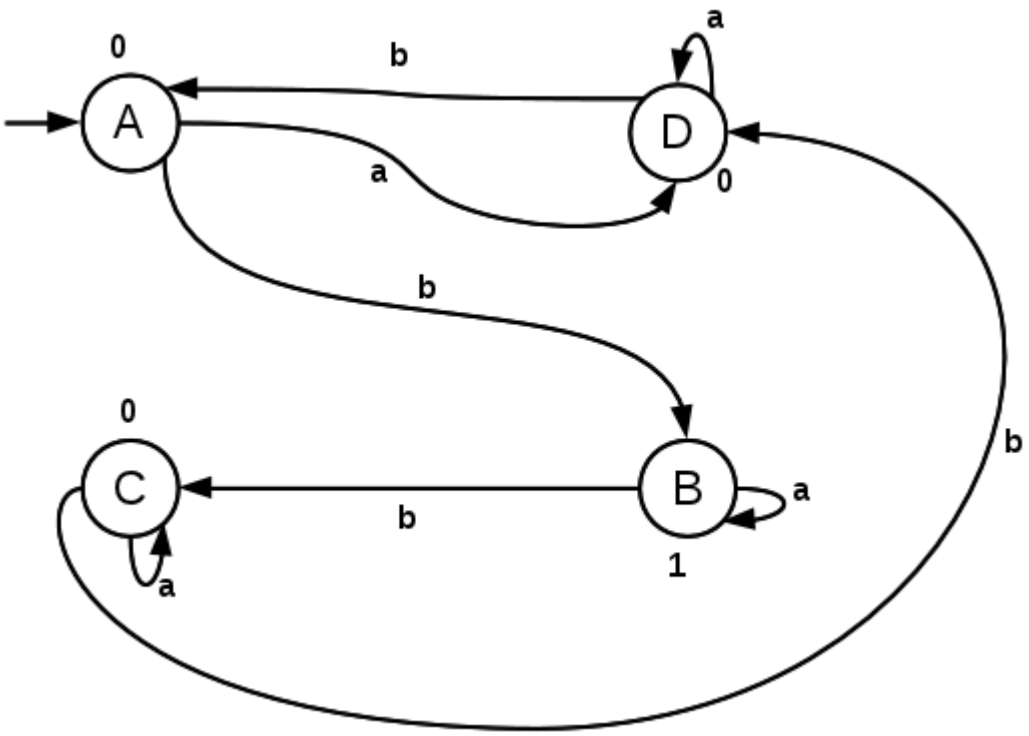
In Moore machine, every state has a fixed output.

Example,

The following is an example transition table for a Moore Machine.

	Input Symbol		
Current State	a	b	Output
$\longrightarrow$A	D	B	0
B	B	C	1
C	C	D	0
D	D	A	0

The transition diagram corresponding to this is ,



A is the start state.

There is no final state in a Moore machine.

Output is shown above every state.

Definition of a Moore Machine

Moore Machine is a six tuple machine and is defined as,

$$M_0\,(\,Q,\,\sum,\,\Delta,\,\delta,\,\lambda',\,q_0\,)$$

where M_0 is the Moore Machine,

Q is a finite set of states,

$\sum$ is a set of input symbols,

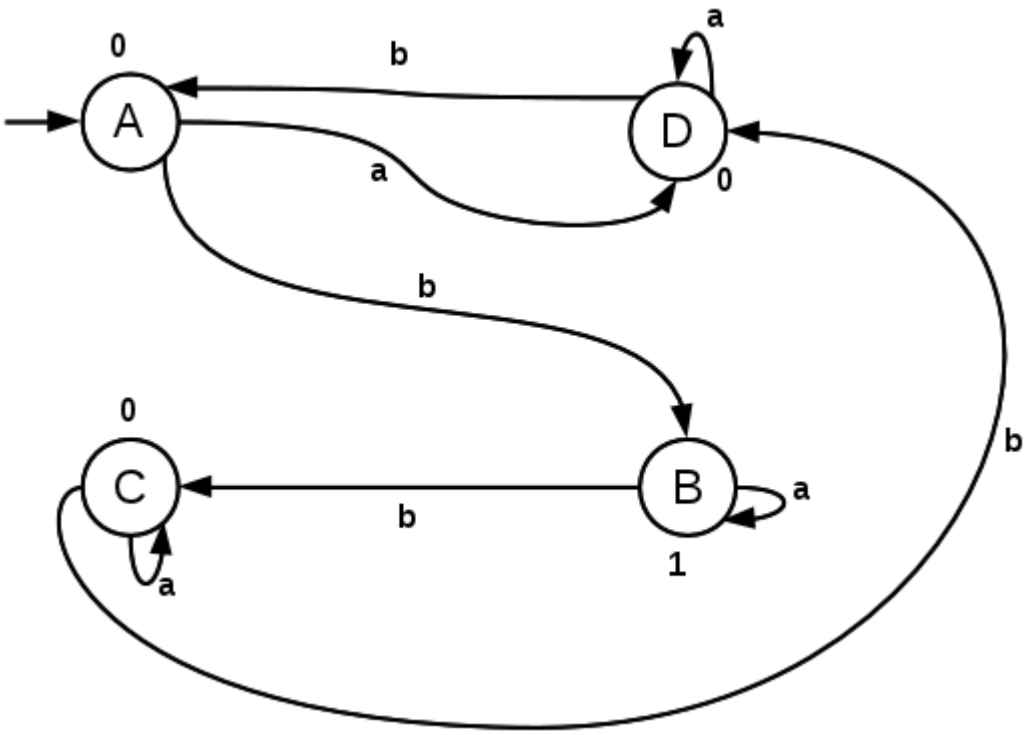
Δ is set of outputs,

δ is the set of transition functions,

λ' is the mapping function which maps a state to an output,

q_0 is the start state.

Consider the following Moore Machine,



In the above Moore Machine,

The set of states,

$Q = \{A, B, C, D\}$

The set of input symbols,

$\sum = \{a, b\}$

The set of outputs,

$\Delta = \{0, 1\}$

The set of transitions,

$\delta(A, a) = D$

$\delta(B, b) = C$

$\delta(C, a) = C$ and so on.

$\lambda'(A) = 0$

$\lambda'(B) = 1$

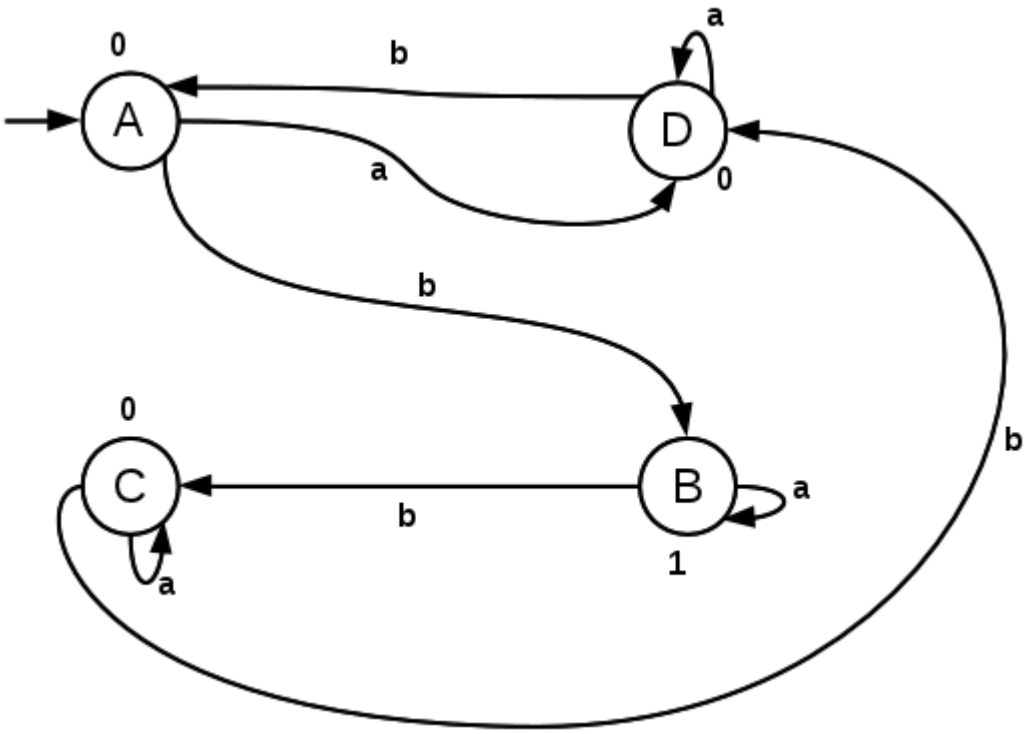
$\lambda'(C) = 0$

$\lambda'(D) = 0$

String Processing through Moore Machine

Example 1:

Consider the Moore Machine given below:



Process the string $abbb$ using the Moore Machine and find the output string.

Begin from the start symbol, A,

$$\delta(A, \underline{a}bbb) = D \qquad \lambda'(A) = 0$$

$$\delta(D, a\underline{b}bb) = A \qquad \lambda'(D) = 0$$

$$\delta(A, ab\underline{b}b) = B \qquad \lambda'(A) = 0$$

$$\delta(B, abb\underline{b}) = C \qquad \lambda'(B) = 1$$

$$\lambda'(C) = 0$$

The output string we got is 00010.

Example 2:

Consider the following Moore Machine,

Current State	Input Symbol a	Input Symbol b	Output
$\longrightarrow$A	B	C	0
B	C	D	0
C	D	E	0
D	E	B	0
E	B	C	1

Process the string aabbba through the Moore Machine and find the output.

Note that here transition table corresponding to the Moore Machine is given.

Begin from the start symbol, A,

$$\delta(A, \underline{a}abbba) = B \qquad \lambda'(A) = 0$$

$$\delta(B, a\underline{a}bbba) = C \qquad \lambda'(B) = 0$$

$$\delta(C, aa\underline{b}bba) = E \qquad \lambda'(C) = 0$$

$$\delta(E, aab\underline{b}ba) = C \qquad \lambda'(E) = 1$$

$$\delta(C, aabb\underline{b}a) = E \qquad \lambda'(C) = 0$$

$$\delta(E, aabbb\underline{a}) = B \qquad \lambda'(E) = 1$$

$$\lambda'(B) = 0$$

The output string we got is 0001010.

11 Mealy Machine

Mealy machine is a finite automation.

In this finite automation, output is associated with every transition.

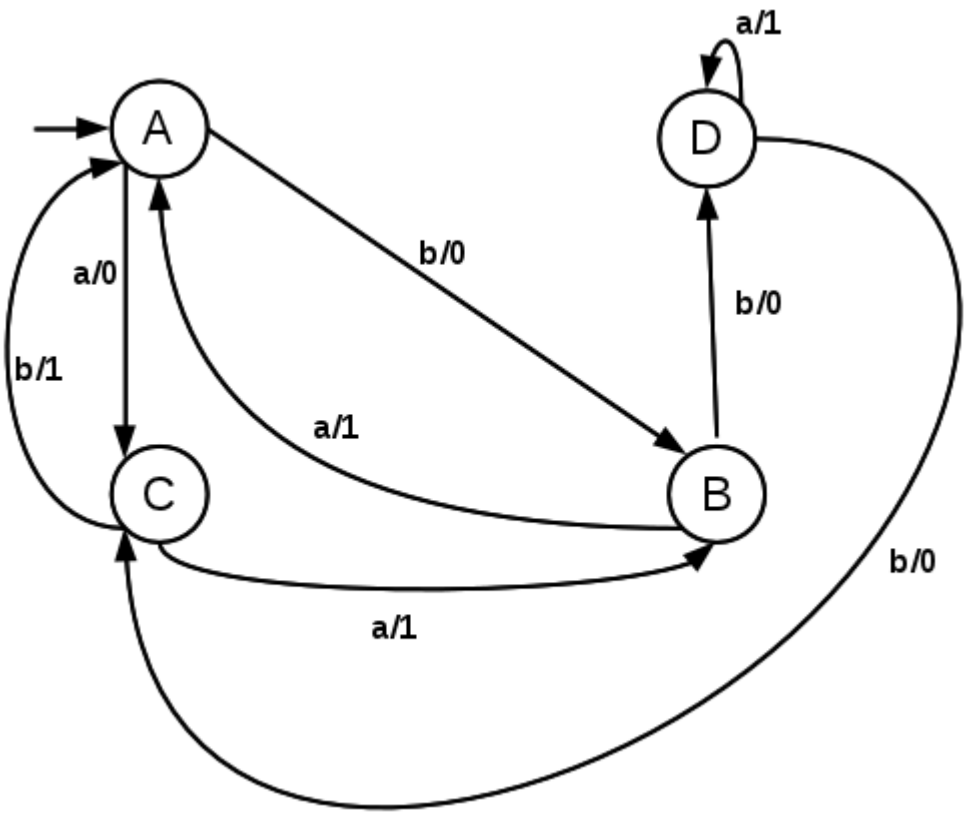
In Mealy machine, every transition for a particular input symbol has a fixed output.

Example,

The following is an example transition table for a Mealy Machine.

	For	Input = a	For	Input=b
Current State	State	Output	State	Output
$\longrightarrow$A	C	0	B	0
B	A	1	D	0
C	B	1	A	1
D	D	1	C	0

The transition diagram corresponding to this Mealy Machine is



A is the start state.

There is no final state in a Mealy machine.

Output is shown with every transition. (a/1 means a is the input symbol and output is 1).

Definition of a Mealy Machine

Mealy Machine is a six tuple machine and is defined as,

$$M_e\,(\,Q,\,\sum,\,\Delta,\,\delta,\,\lambda',\,q_0\,)$$

where M_e is the Mealy Machine,

Q is a finite set of states,

$\sum$ is a set of input symbols,

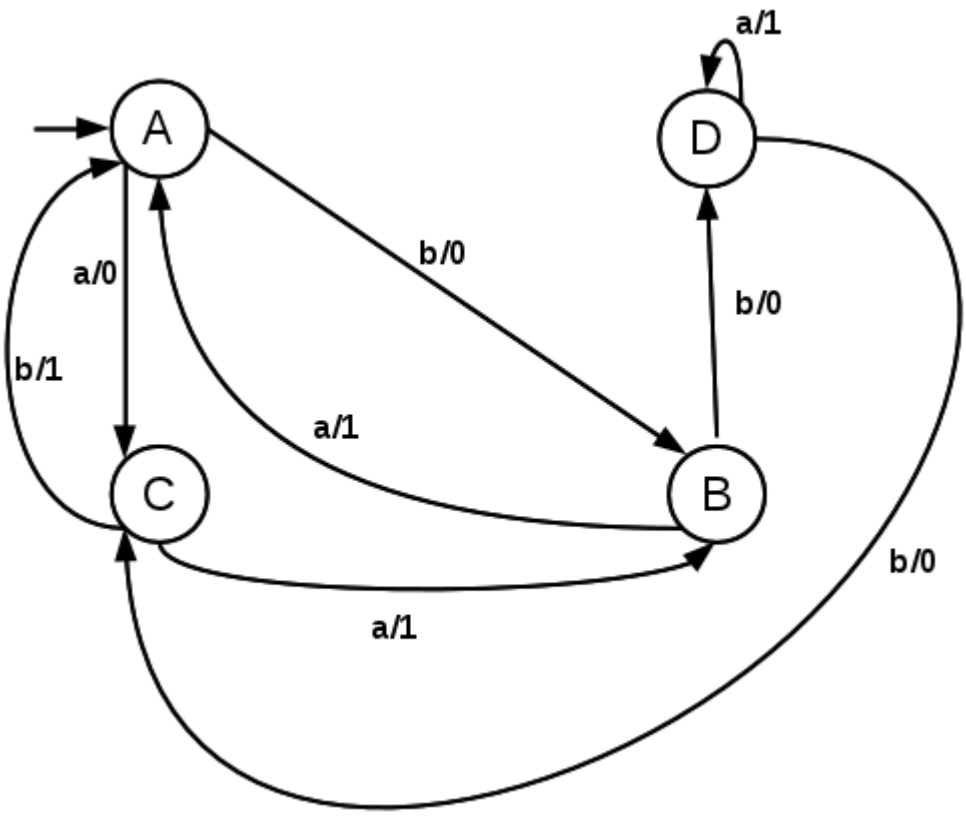
Δ is set of outputs,

δ is the set of transition functions,

λ' is the mapping function which maps a state and an input symbol to an output,

q_0 is the start state.

Consider the following Mealy Machine,



In the above Moore Machine,

The set of states,

$Q = \{A, B, C, D\}$

The set of input symbols,

$\sum = \{a, b\}$

The set of outputs,

$\Delta = \{0, 1\}$

The set of transitions,

$\delta(A, a) = C$

$\delta(B, b) = D$

$\delta(C, a) = B$ and so on.

$\lambda'(A, a) = 0$

$\lambda'(B, a) = 1$

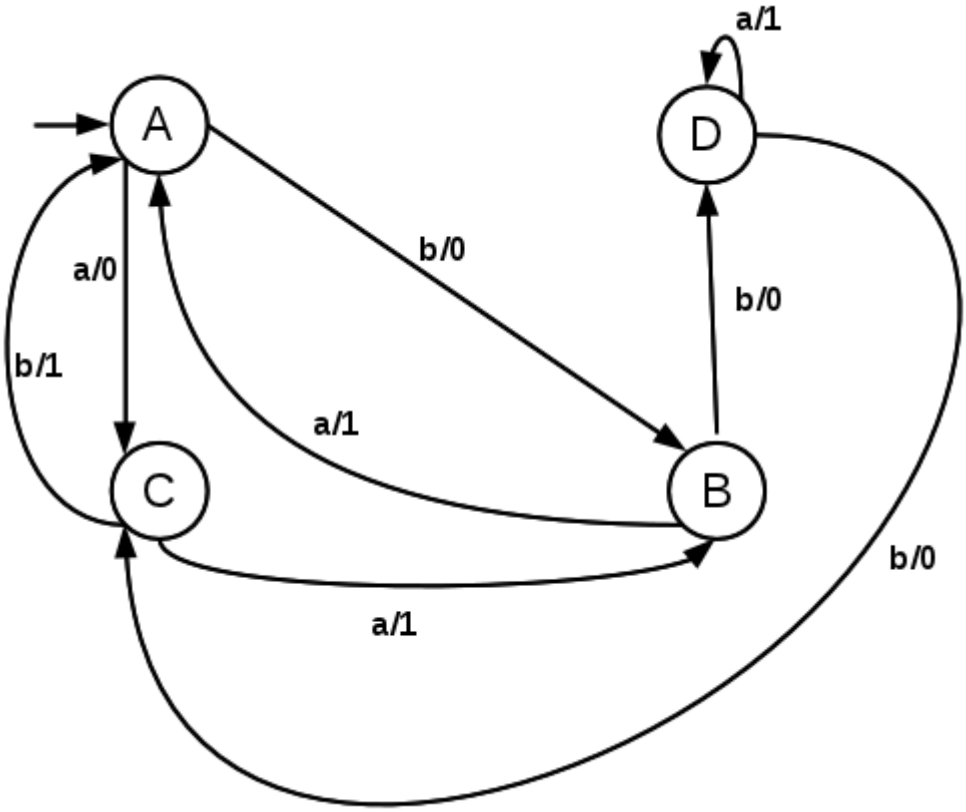
$\lambda'(C, a) = 1$

$\lambda'(D, b) = 0$ and so on.

String Processing through Mealy Machine

Example 1:

Consider the Mealy Machine given below:



Process the string abbb through the above Mealy Machine and find out the output.

Begin from the start symbol, A,

$\delta(A, \underline{a}bbb) = C$ $\qquad$ $\lambda'(A, a) = 0$

$\delta(C, a\underline{b}bb) = A$ $\qquad$ $\lambda'(C, b) = 1$

$\delta(A, ab\underline{b}b) = B$ $\qquad$ $\lambda'(A, b) = 0$

$\delta(B, abb\underline{b}) = D$ $\qquad$ $\lambda'(B, b) = 0$

The output string we got is 0100.

12 Moore Machine to Mealy Machine Conversion

Example 1:

Consider the Moore Machine given below:

	Input Symbol		
Current State	a	b	Output
$\longrightarrow$A	D	C	0
B	A	B	1
C	C	A	1
D	C	B	0

Convert this Moore Machine to a Mealy Machine.

If we look at the above transition table, we can see that the machine has output 0 for the states A and D.

The machine has output 1 for the states B and C.

To convert this to a Mealy Machine, the output is taken as 0, whenever there is a transition to A or D.

Also, the output is taken as 1, whenver there is a transition to B or C.

The Mealy Machine is as follows:

	For	Input = a	For	Input=b
Current State	State	Output	State	Output
$\longrightarrow$A	D	0	C	1
B	A	0	B	1
C	C	1	A	0
D	C	1	B	1

Example 2:

Convert the Moore Machine given below to a Mealy Machine:

	Input	Symbol	
Current State	a	b	Output
$\longrightarrow$A	B	C	0
B	A	C	1
C	C	B	1

If we look at the above transition table, we can see that the machine has output 0 for the state A.

The machine has output 1 for the states B and C.

To convert this to a Mealy Machine, the output is taken as 0, whenever there is a transition to A.

Also, the output is taken as 1, whenver there is a transition to B or C.

The Mealy Machine is as follows:

	For	Input = a	For	Input=b
Current State	State	Output	State	Output
$\longrightarrow$A	B	1	C	1
B	A	0	C	1
C	C	1	B	1

13 Mealy Machine to Moore Machine Conversion

Example 1:

Consider the Mealy Machine given below:

	For	Input = a	For	Input=b
Current State	State	Output	State	Output
$\longrightarrow$A	B	1	C	1
B	C	0	A	1
C	A	1	B	0

Conver this to a Moore Machine.

Here, State A has output 1 at two places.

State B has output 0 at one place and 1 at another place. So this state B is decomposed into two states, B0 and B1. In the new Moore Machine, state B0 has output 0 and B1 has output 1.

State C has output 0 at one place and output 1 at another place. So this state is decomposed into two states, C0 and C1. In the new Moore Machine, state C0 has output 0 and C1 has output 1.

The Moore Machine is given below;

	Input	Symbol	
Current State	a	b	Output
$\longrightarrow$A	B1	C1	1
B0	C0	A	0
B1	C0	A	1
C0	A	B0	0
C1	A	B0	1

Example 2:

Convert the following Mealy Machine to Moore Machine.

	For	Input = a	For	Input=b
Current State	State	Output	State	Output
$\longrightarrow$A	B	1	C	1
B	C	0	A	1
C	A	1	D	0
D	D	0	B	1

Here, State A has output 1 at two places.

State B has output 1 at two places.

State C has output 0 at one place and output 1 at another place. So this state is decomposed into two states, C0 and C1. In the new Moore Machine, state C0 has output 0 and C1 has output 1.

State D has output 0 at both places.

The Moore Machine is given below;

Current State	Input Symbol a	Input Symbol b	Output
$\longrightarrow$A	B	C1	1
B	C0	A	1
C0	A	D	0
C1	A	D	1
D	D	B	0

Part VII. Applications of Finite Automata

Finite automata has several applications in many areas such as

compiler design,

special purpose hardware design,

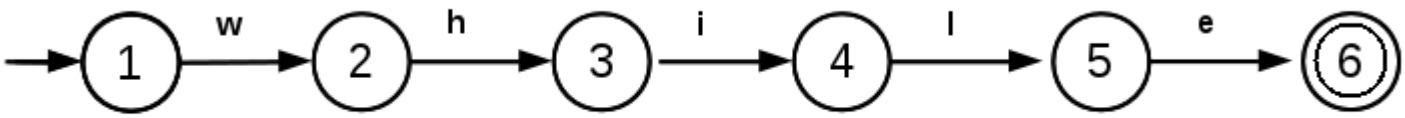
protocol specification etc..

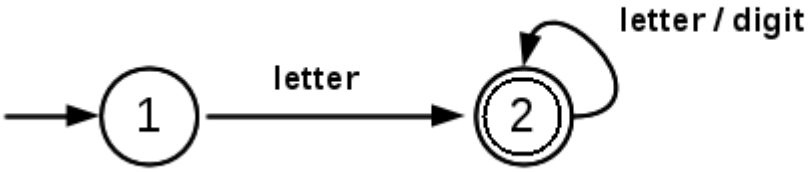
Some of the applications are explained below:

1. Compiler Design

Lexical analysis or scanning is an important phase of a compiler. In lexical analysis, a program such as a C program is scanned and the different tokens(constructs such as variables, keywords, numbers) in the program are identified. A DFA is used for this operation.

For example, finite automation to recognize the tokens, 'while' keyword and variables are shown below:





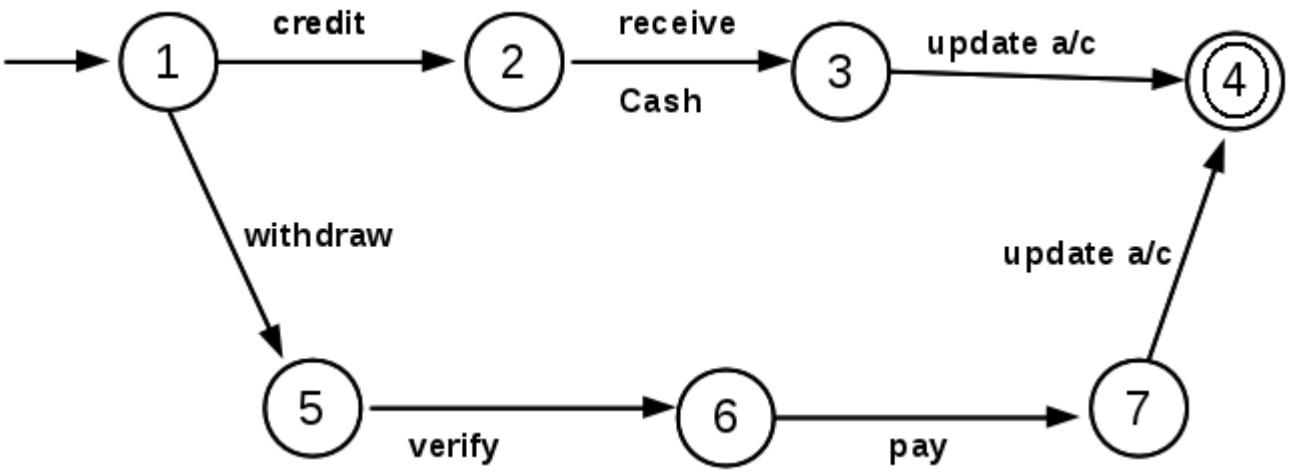
The well known lexical analyser tool, Lex uses DFA to perform lexical analysis.

2. Hardware Design

In the design of computers, finite automation is used to design control unit of a computer. A typical sequence of operations of a computer consists of a repretition of instructions and every instruction involves the actions fetch, decode, fetch the operand, and execute. Every state of a finite automation represents a specific stage of instruction cycle.

3. Protocol Specification

A system is composed of an interconnected set of subsystems. A protocol is a set of rules for proper coordination of activities of a system. Such a protocol can be described using finite automations. An example of a bank is ahown below:



State 1:

State 1 represents the entry of a customer.

State 2:

After he wishes to credit some cash to his account, system is in state 2.

State 3:

Then the cashier receives cash. It reaches state 3.

State 4:

Then the cashier updates the acoount and reach the final state 4.

State 5:

If the customer wants to withdraw cash, he submits the slip ans system reaches state 5.

State 6:

Then the slip is verified to confirm that he has sufficient balance and reaches state 6.

State 7:

Then the customer is paid cash and reaches state 7.

State 8:

Then the customer account is updated and system reaches final state 4 of the transaction.

Part VIII. Pumping Lemma for Regular Languages

Regular Languages

Regular languages are those classes of languages accepted by DFA's and by NFAs. These languages can be defined using regular expressions.

Not every language is regular.

For example,

Language defined by, $L = \{a^n b^n \ for \ n = 0, 1, 2, 3, ...\}$ is not regular.

Language defined by, $L = \{a^n b a^n \ for \ n = 0, 1, 2, 3, ...\}$ is not regular.

Language defined by, $L = \{a^n b^n a a b^{n+1} \ for \ n = 1, 2, 3, ...\}$ is not regular.

A powerful technique, known as Pumping Lemma can be used to show that certain languages are not regular.

Pumping Lemma for Regular Languages

Definition

Let L be a regular language. Then, there exists a constant n such that

for every string w in L and $|w| \geq n$,

w can be broken into three parts, x , y and z such that

$w = xyz$, $y \notin \varepsilon$, and $|xy| \leq n$.

Then for all $i \geq 0$, the string $xy^i z$ also belongs to L.

That is, we always find a non-empty string y not too far from the beginning of w that can be "pumped"; that is, repeating y any number of times, keeps the resulting string in the language.

A Simple Definition

Let L be a language and w be a string in the language. Break w into three parts. Then write, $w = xyz$.

Then $xy^i z$ must also be a string in the language. That is, strings $xy^0 z$, $xy^1 z$, $xy^2 z$, $xy^3 z$, $xy^4 z$ ..... must be in the language.

Then L is a context free language.

For example,

Example 1:

Consider the language $L = \{a^n | n \geq 1\}$. This language is regular.

The set of strings in this language are,

{a, aa, aaa, aaaa, aaaaa, ...}

Consider one string in this language, aaaa.

Let w=aaaa.

Break w into three parts, x, y and z.

Let w=aaaa=xyz

That is

w =	a	aa	a
	x	y	z

Then,

xy^iz is

Let i=1,

we have, xy^1z

w =	a	aa	a
	x	y^1	z

xy^1z=aaaa =a^4 is in the above language.

Let i=2,

we have, xy^2z

w =	a	aaaa	a
	x	y^2	z

xy^2z=aaaaaa =a^6 is in the above language.

Let i=3,

we have, xy^3z

w =	a	aaaaaa	a
	x	y^3	z

xy^3z=aaaaaaaa =a^8 is in the above language and so on.

Since all the strings of the form xy^iz , are in the above language, the language a^n is a regular language.

Example 2:

Consider the language $L = \{a^nb^m|n, m \geq 1\}$. This language is regular.

The set of strings in this language are,

{ab, abb, abbb, aab, aabb,aaaabbbbbbbb .....}

Consider one string in this language, aaabb.

Let w=aaabb.

Break w into three parts, x, y and z.

Let w=aaabb=xyz

That is

w =	a	aa	bb
	x	y	z

Then,

xy^iz is

Let i=1,

we have, xy^1z

w =	a	aa	bb
	x	y^1	z

xy^1z=aaabb =a^3b^2 is in the above language.

Let i=2,

we have, xy^2z

w =	a	aaaa	bb
	x	y^2	z

xy^2z=aaaaabb =a^5b^2 is in the above language and so on

Since all the strings of the form xy^iz , are in the above language, the language a^nb^m is regular.

Example 3:

Consider the language $L = \{a^nb^n | n \geq 1\}$. This language is not regular.

The set of strings in this language are,

{ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, ...}

Consider one string in this language, aaabbb.

Let w=aaabbb.

Break w into three parts, x, y and z.

Let w=aaabbb=xyz

That is

w =	a	aa	bbb
	x	y	z

Then,

xy^iz is

Let i=1,

we have, xy^1z

w =	a	aa	bbb
	x	y^1	z

xy^1z=aaabbb =a^3b^3 is in the above language.

Let i=2,

we have, xy^2z

w =	a	aaaa	bbb
	x	y^2	z

xy^2z=aaaaabbb =a^5b^3 is not in the above language.

Since some of the strings of the form xy^iz , are not in the above language, the language a^nb^n is not regular.

Proof

Let L is regular. Then a DFA exists for L.

Let that DFA has n states.

Consider a string w of length n or more, let $w = a_1, a_2, a_3.....a_m$, where m≥n and each a_i is an input symbol

By the Pigeonhole principle, it is not possible for the n+1 different states (P_i's) for i=0,1,2,3....n to be distinct, since there are only n states.

Thus we can find two different integers i and j such that $P_i = P_j$.

Now we can break $w = xyz$ as follows:

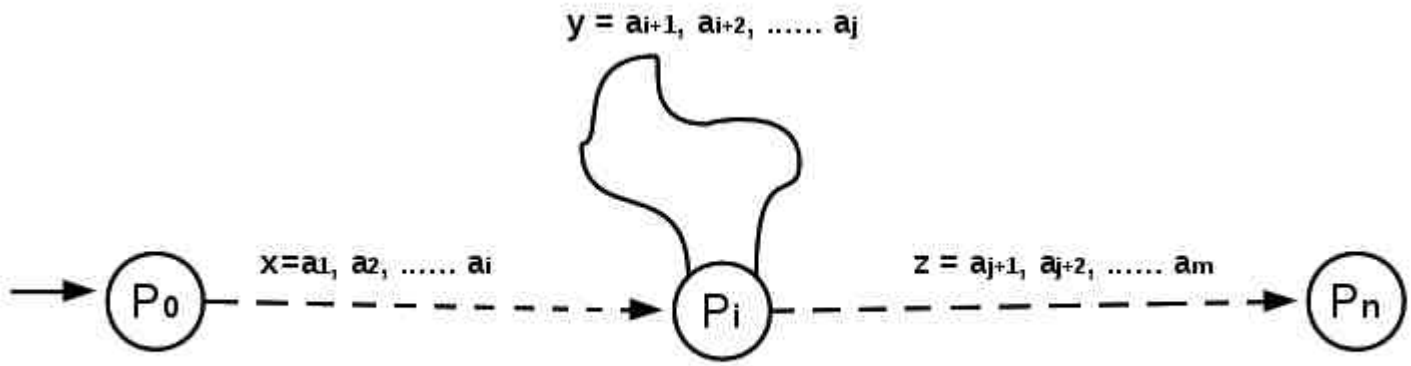
1. $x = a_1a_2, ...a_i$
2. $y = a_{i+1}a_{i+2}.....a_j$
3. $z = a_{j+1}a_{j+2}.....a_m$

That is x takes us to P_i once,

y takes us from P_i back to P_i (since P_i is also P_j), and

z is balance of w.

This is shown in the following diagram.



Note that x may be empty, when i=0.

Also, z may be empty, if j = n = m.

y cannot be empty, since i is strictly less than j.

Suppose the above automation receives xy^iz for an $i \geq 0$.

If i = 0, then the string is xz,

automation goes from the start state P_0 to P_i on input x.

Since P_i is also P_j, it must be that M goes from P_i to the accepting state on input z.

Thus xz is accepted by the automation.

That is,

$\hat{\delta}(P_0, x) = \hat{\delta}(P_i, x) = P_i$

$\hat{\delta}(P_i, z) = P_n$

If i>0, then

automation goes from P_0 to P_i on input string x,

circles from P_i to P_i, i times on input y_i, and then

goes to the accepting state on input z.

Thus xy^iz is accepted by the automation.

That is,

$\hat{\delta}(P_0, x) = P_i$

$\hat{\delta}(P_i, y) = P_i$

$\hat{\delta}(P_i, y^i) = P_i$

$\hat{\delta}(P_i, z) = P_n$

Pumping lemma is useful for proving that certain languages are not regular.

Proving Non-regularity of certain Languages using Pumping Lemma

To prove that certain languages are not regular, following are the steps:

Step 1:

Assume that L is regular. Let n be the number of steps in the corresponding finite automation.

Step 2:

Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

Step 3:

Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

The important part of proof is Step 3. There, we need to find i such that $xy^iz \notin L$.

In some cases, we prove $xy^iz \notin L$ by considering $|xy^iz|$.

In some cases, we may have to use the structure of strings in L.

Example 1:

Prove that language, $L = \{0^i1^i$ for $i \geq 1$ } is not regular.

Step 1: Assume that L is regular. Let n be the number of steps in the corresponding finite automation.

Step 2: Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

Let $w = 0^n1^n$.

Then $|w| = 2n > n$.

By pumping lemma, we can write $w = xyz$, with $|xy| \leq n$ and $|y| \neq 0$.

Step 3: Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

The string y can be any one of the following forms:

Case 1: y has only 0's, ie. $y = 0^k$ for some $k \geq 1$.

In this case, we can take i=0. As $xyz = 0^n1^n$, $xz = 0^{n-k}1^n$.

As $k \geq 1$, $n - k \neq n$. So, $xz \notin L$.

Case 2: y has only 1's, ie. $y = 1^m$, for some $m \geq 1$.

In this case, take i=0.

As before, xz is 0^n1^{n-m} and $n \neq n - m$. So, $xz \notin L$.

Case 3: y has both 0's and 1's. ie. $y = 0^k1^j$, for some k, $j \geq 1$

In this case, take i=2.

As $xyz = 0^{n-k}0^k1^j1^{n-j}$,$xy^2z = 0^{n-k}0^k1^j0^k1^j1^{n-j}$,

As xy^2z is not of the form 0^i1^i, $xy^2z \notin L$.

In all three cases, we get a contradiction. So L is not regular.

Example 2:

Prove that $L = \{a^{i^2} | i \geq 1\}$ is not regular.

Proof:

Step 1: Assume that L is regular. Let n be the number of steps in the corresponding finite automation.

Step 2: Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

Let $w = a^{n^2}$. Then $|w| = n^2 > n$.

By pumping lemma, we can write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$.

Step 3: Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

Let us choose, i=2.

Then consider xy^2z.

$|xy^2z| = |x| + 2|y| + |z| > |x| + |y| + |z|$ as $|y| > 0$.

This means $n^2 = |xyz| = |x| + |y| + |z| < |xy^2z|$.

As $|xy| \leq n$, we have $|y| \leq n$.

Therefore,

$$|xy^2z| = |x| + 2|y| + |z| \leq n^2 + n$$

That is,

$$n^2 < |xy^2z| \leq n^2 + n < n^2 + n + n + 1$$

That is,

$$n^2 < |xy^2z| \leq (n+1)^2$$

Thus $|xy^2z|$ strictly lies between n^2 and $(n+1)^2$, but is not equal to any one of them.

Thus $|xy^2z|$ is not a perfect square and so $xy^2z \notin L$.

But by pumping lemma, $xy^2z \in L$. This is a contradiction.

Example 3:

Prove that $L = \{a^p | p \text{ is a prime}\}$ is not regular.

Proof:

Step 1: Assume that L is regular. Let n be the number of steps in the corresponding finite automation.

Step 2: Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

Let p be a prime number greater than n.

Let $w = a^p$.

By pumping lemma, we can write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

x, y, z are simply strings of a's.

So, $y = a^m$, for some $m \geq 1$ (and $\leq n$).

Step 3: Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

Let $i = p + 1$.

Then $|xy^iz| = |xyz| + |y^{i-1}| = p + (i-1)m = p + pm$.

By pumping lemma, $xy^iz \in L$.

But $|xy^iz| = p + pm = p(1+m)$.

$p(1+m)$ is not prime.

So $xy^iz \notin L$.

This is a contradiction.

Thus L is not regular.

Example 4:

Show that $L = \{ww | w \in \{a,b\}^*\}$ is not regular.

Proof:

Step 1: Assume that L is regular. Let n be the number of steps in the corresponding finite automation.

Step 2: Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$, with $|xy| \leq n$ and $|y| > 0$.

Let us consider $ww = a^nba^nb$ in L.

$|ww| = 2(n+1) > n$

We can apply pumping lemma to write

$ww = xyz$ with $|y| \neq 0$, $|xy| \leq n$.

Step 3: Find a suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

We want to find i so that $xy^iz \notin L$ for getting a contradiction.

The string y can be in only one of the following forms:

Case 1: y has no b's. ie. $y = a^k$ for some $k \geq 1$.

Case 2: y has only one b.

We may note that y cannot have two b's. If so, $|y| \geq n+2$. But $|y| \leq |xy| \leq n$.

In case 1, we can take $i = 0$. Then $xy^0z = xz$ is of the form a^mba^nb, where $m = n - k < n$. We cannot write xz in the form uu with $u \in \{a,b\}^*$, and so $xz \notin L$.

In case 2 also, we can take $i = 0$. Then $xy^0z = xz$ has only one b (as one b is removed from xyz, b being in y). So $xz \notin L$ as any element in L should have an even number of a's and an even number of b's.

Thus in both cases, we have a contradiction. So, L is not regular.

Questions (from Old syllabus of S7CS TOC)

MGU/Nov2011

1. State the formal definition of an E-NFA (4marks).
2. Design a DFA that accepts the language:

   L = {w/w starts with 01} from the alphabet {0,1}. (4marks)

3a. Explain the different applications of finite automata.

OR

b. Prove that for every NFA there is an equivalent DFA (12marks).

MGU/April2011

1. Define regular expression (4marks).
2. Draw a DFA for the regular expression $[(a^* + b^*)^*]a^*$. (4marks)
3. Differentiate a DFA and NFA (4marks).

4a. i) If 'r' is a regular expression , then show that there is an NFA that accepts L(r).

ii) Design a minimised FSA that recognize $(1110/100)^*0^*$.

MGU/Nov2010

1. Draw a DFA for the regular expression $(a^*b^*)^*$. (4marks)
2. Define pumping lemma for regular language (4marks).

3a. i) Construct a DFA to accept the language $L = [(a/b)^*/(ab)^*]^*$.

OR

b. Find a minimal DFA for the language $L = \{a^n b^n : n \geq 2, m \geq 1\}$. (12marks)

MGU/May2010

1. State the mathematical definition of DFA (4marks).
2. a. Give regular set for the following expressions : $1(01)^*(10)^*1$.

b. What is the difference between DFa and NFA. (4marks)

3a. Prove that for every non-deterministic finite automation there is an equivalent deterministic finite automation (12marks).

4a. Construct a DFA equivalent to non-deterministic automata given below:

b. Construct a DFA for the language given by:

$L = [(a + b)^*ab(a + b)^*] \cap L[(ab)^*]$. (12marks)

MGU/Nov2009

1. Find minimal DFA's for the language $L = \{a^n b^m, n \geq 2; m \geq 1\}$. (4marks)

2a. Prove the equivalence of NFA and DFA.

OR

b. Explain in detail with an example the conversion of NDFA to DFA (12marks).

MGU/Nov2008

1. Describe a finite automation (4marks).
2. Construct automata to accept $1(1 + 0)^* + a(a + b)^*$. (4marks)
3. Explain pumping lemma for regular languages (4marks).

4a. i) Design a minimum state FSA to recognize the expression $(111/000)^*0$.

ii. Construct automation that accepts language $S \longrightarrow aA, A \longrightarrow abB/b, B \longrightarrow aA/a$.

OR

b. Differentiate between deterministic and non-deterministic finite automation (12marks).

MGU/May2008

1. Differentiate deterministic and non-deterministic automata (4marks).

2. Construct automata to accept $1(1+0)^* + a(a+b)^*$. (4marks)

3a. explain the algorithm for the minimization of DFA.

OR

b. i) Construct DFA for the language given by

$L = [(a+b)^*ab(a+b)^*] \cap L^1[(ab)^*]$

ii) Design a minimum state FSA to recognize the expression $(111/000)^*0$. (12marks)

MGU/JDec2007

1. Write regular expression of set of strings with even number of a's followed by odd number of b's (4marks).

2. Show that the class of languages accepted by finite automata is clossed under union (4marks).

3. What are useless symbols and how they are removed (4marks)?

4a. i) Construct NFA for $(01^* + 1$. (12marks)

5a. i) Construct an automation accepting language generated by grammar $S \longrightarrow aA/a, A \longrightarrow abB,$ and $B \longrightarrow bS$.

ii. State and prove pumping lemma for regular languages.

OR

b. Construct DFA for language L over the $\sum = \{0,1\}$ and α is set of strings ending with "00". (12marks)

MGU/July2007

1. Define regular expression (4marks).

2. How will you use pumping lemma to show that certain languages are not regular? Give the general steps (4marks).

3a. i. Prove that $L = \{ww^R | w\epsilon\{a,b\}^*\}$ is not regular.

ii. Prove that $L = \{a^p | P$ is a prime number } is not regular.

OR

b. i. Give the regular expression for the language $L = \{a^n b^m | n \geq 4, m \leq 3\}$ and its component L.

ii. Define DFA and construct a DFA that accepts $L = \{w\epsilon\{a,b\}^*|$no. of a's in w is even and no. of b's in w is odd}. (12marks)

MGU/Jan2007

1. Define regular expression. (4marks)

2. Construct a DFA (transition diagram) accepting the language $L = \{w\epsilon(a,b)^*|$w has abab as substring (4marks).

3a. i. Prove that $L = a^n b^n | n > 0\}$ is not regular.

OR

b. i. State and prove pumping lemma for regular languages.

ii. Construct a DFA accepting $L = \{w\epsilon(a,b)^*$

MGU/July2006

1. Construct a DFA for $(a/b)^*abb$. (4marks)

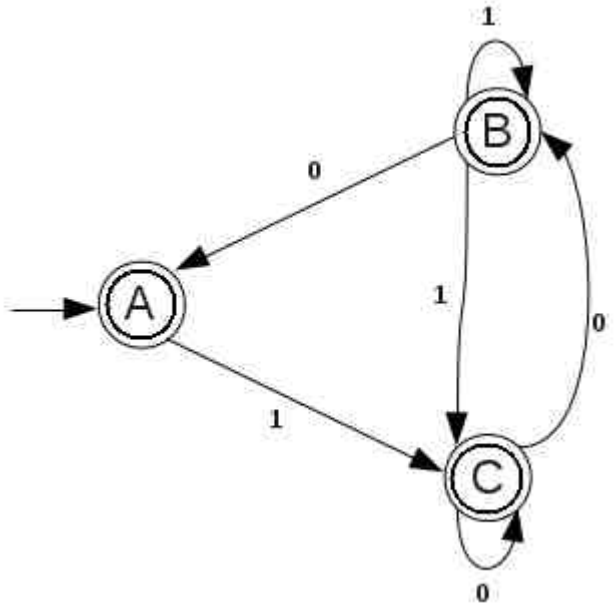
2. Define a NDFA. (4marks)

3a. For the regular expression, $a^*(a/b)^*b$, draw the NFA. Obtain DFA from NFA.

OR

b. i. Design an algorithm for minimising the states of DFA.

ii. Construct the regular expression equivalent to the state diagram given below:



MGU/Nov2005

1. What are the strings in the regular sets denoted by the regular expression ab^* and $(ab)^*$? (4marks)

2. Define a NDFA. (4marks)

3a. If L is a language accepted by NDFA, then prove that there exists a DFA that accepts L.

OR

b. Discuss briefly any two applications of finite state automata (12marks).

References

.

Pandey, A, K (2006). An Introduction to automata Theory and Formal Languages. Kataria & Sons.

Mishra, K, L, P; Chandrasekaran, N (2009). Theory of Computer Science. PHI.

Nagpal, C, K (2011). Formal Languages and Automata Theory. Oxford University Press.

Murthy, B, N, S (2006). Formal Languages and Automata Theory. Sanguine.

Kavitha,S; Balasubramanian,V (2004). Theory of Computation. Charulatha Pub.

Linz, P (2006). An Introduction to Formal Languages and Automata. Narosa.

Hopcroft, J, E; Motwani, J; Ullman, J, D (2002). Introduction to Automata Theory, Languages and Computation. Pearson Education.

St. Joseph's College of Engineering & Technology Palai

Department of Computer Science & Engineering

S4 CS

CS010 406 Theory of Computation

Module 3

Theory of Computation - Module 3

Syllabus

Context Free Grammar –Simplification of CFG-

Normal forms-Chomsky Normal form and Greibach Normal form-

pumping lemma for Context free languages-

Applications of PDA -

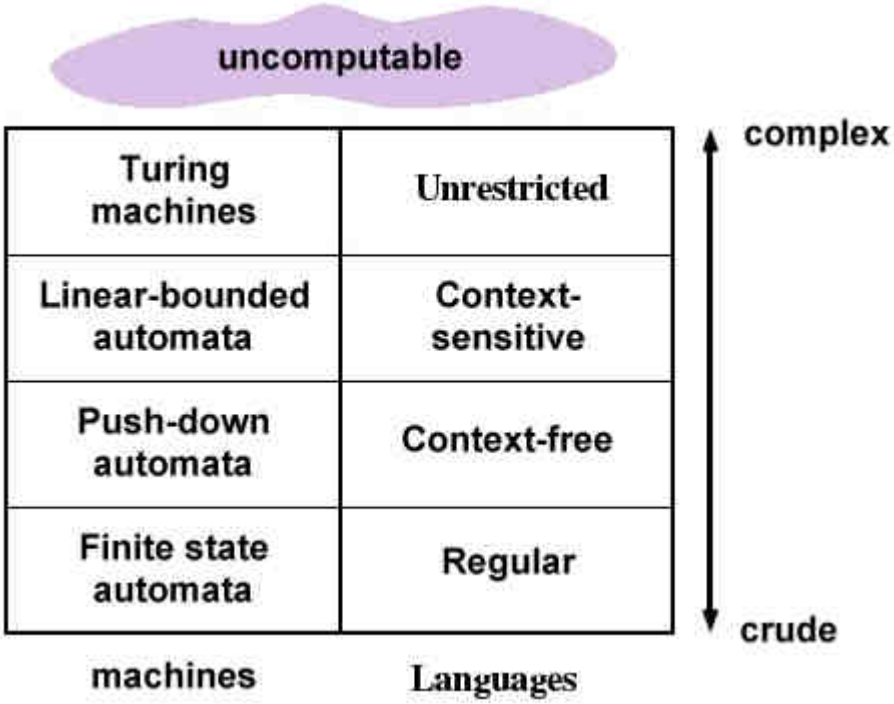
Pushdown Automata – Formal definition – Language acceptability by PDA through empty stack and final state –

Deterministic and nondeterministic PDA – designing of PDA-

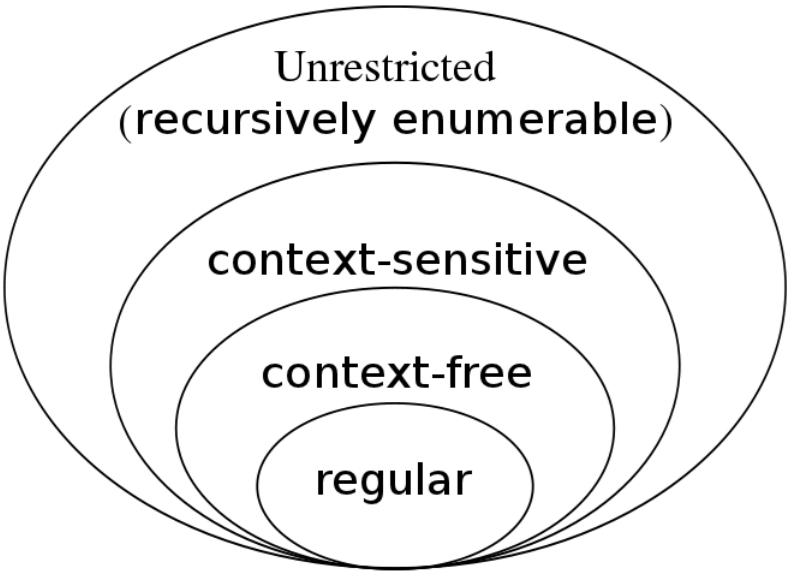
Contents

Context Free Grammars and Languages

As we learned in first module, according to Chomsky classification, Type- 2 languages are also called context free languages. They are generated using context free grammars. They are recognised using push down automata.



Also regular languages are a subset of context free languages.



Part I. Context Free Grammars (CFG)

Context free grammar, G is defined as,

$$G = (V, \sum, P, S)$$

where

V is a set of non-terminals,

$\sum$ is a set of terminals,

S is the start symbol,

P is a set of productions.

A grammar, G is context free, if productions are of the form,

$$\alpha \longrightarrow \beta$$

where α is a single non terminal.

Every regular grammar is context free.

Example 1:

An example for context free grammar is shown below:

$$S \longrightarrow ASA|BSB|a|b$$

$$A \longrightarrow a$$

$$B \longrightarrow b$$

where S is the start symbol.

From the definition of CFG,

$$G = \{S, A, B\}$$

$$\sum = \{a, b\}$$

$$P = \{S \longrightarrow ASA|BSB|a|b,\ A \longrightarrow a,\ B \longrightarrow b\}$$

$$S = S$$

The production,

$$S \longrightarrow ASA|BSB|a|b$$

can also be written as,

$$S \longrightarrow ASA$$

$$S \longrightarrow BSB$$

$$S \longrightarrow a$$

$$S \longrightarrow b$$

Example 2:

The following is an example for context free grammar.

Stmt $\longrightarrow$ id = Expr;

Stmt $\longrightarrow$ if (Expr) Stmt

Stmt $\longrightarrow$ if (Expr) Stmt else Stmt

Stmt $\longrightarrow$ while (Expr) Stmt

Stmt $\longrightarrow$ do Stmt while (Expr);

Stmt $\longrightarrow$ { Stmts }

Stmts $\longrightarrow$ Stmts Stmt| ε

where $Stmts$ is the start symbol.

From the definition of CFG,

$G = \{Stmts, stmt, Expr\}$

$\sum = \{id, =, ;, (,), if, else, while, do\}$

P is the set of productions given above

$S = Stmts$

Note that above CFG corresponds to some statements in C programming language.

1 Language of Context Free Grammar (Context Free Language)

A language, L of G is a set of strings that can be derived from G.

Example 1:

Consider the following context free grammar,

$S \longrightarrow aA|bB$

$A \longrightarrow x$

$B \longrightarrow y$

The language corresponding to the above CFG is {ax,by}

A string w belongs to a context free language (CFL), L, if it can be derived from the CFG associated with L.

That is,

$L(G) = \{w \in \sum \mid S \overset{*}{\underset{G}{\Longrightarrow}} w\}$

There are two approaches to check whether a string belongs to a CFL. They are,

Derivations, and

Reductions.

Derivations

It is a mechanism to check whether a string w belongs to a context free language (CFL).

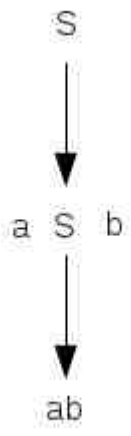
Example 1:

Consider the following CFG,

$S \longrightarrow aSb \mid ab$

where S is the start symbol.

Check whether the string $aabb$ belongs to the CFL corresponding to above CFG.



Above diagram is called a derivation tree (parse tree).

Here, by performing two derivations, we got a^2b^2. So a^2b^2 belongs to the above CFL.

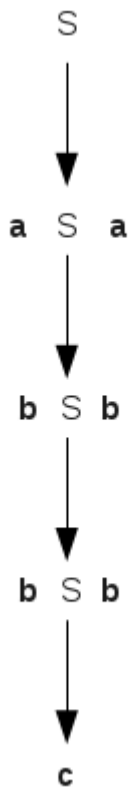
Example 2:

Consider the following CFG,

$$S \longrightarrow aSa \,|\, bSb \,|c$$

where S is the start symbol.

Check whether the string abbcbba belongs to the language corresponding to the above grammar.



From the above derivation tree, the string abbcbba is derived using the given CFG. So the string abbcbba belongs to the above language.

Example 3:

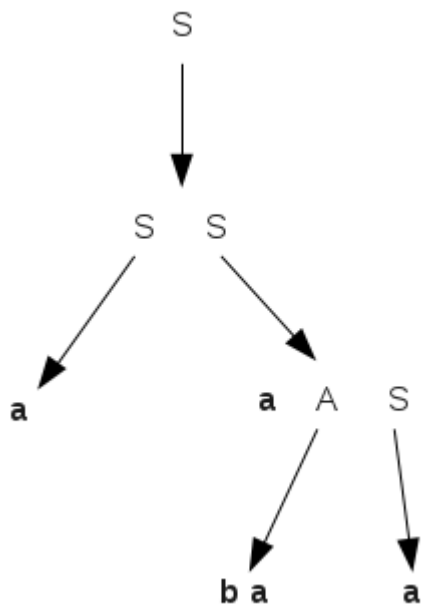
Consider the CFG,

$S \longrightarrow aAS|a|SS$

$A \longrightarrow SbA|ba$

where S is the start symbol.

Check whether the string aabaa can be derived using the above grammar.



Example 4:

Consider the CFG,

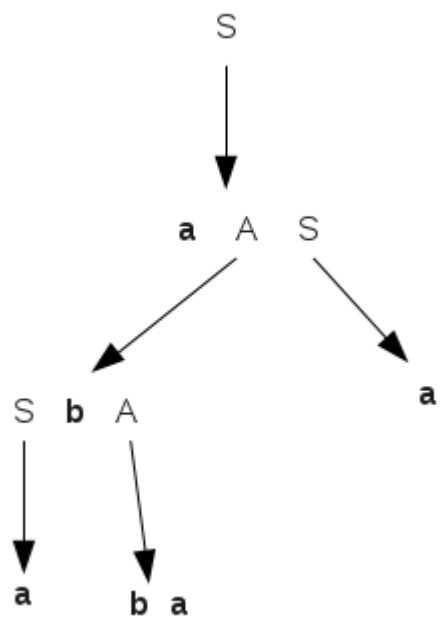
$S \longrightarrow aAS|a$

$A \longrightarrow SbA|SS|ba$

where S is the start symbol.

Check whether the string aabbaa belongs to this language.

Following is the derivation tree:



Thus the above string $aabbaa$ or $a^2b^2a^2$ belongs to the above CFL.

Reductions

This approach can also be used to check whether a string belongs to a context free language. Here, the string w is taken and an inverted tree is made. this is done by performing a number of reductions starting from the given string using the productions of the grammar.

Example 1:

Consider the grammar,

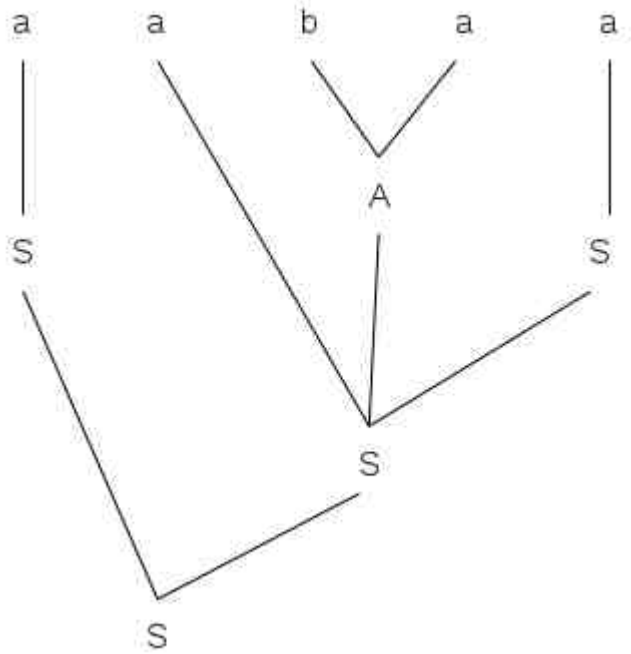
$S \longrightarrow aAS|a|SS$

$A \longrightarrow SbA|ba$

where S is the start symbol.

Check whether the string aabaa belongs to the CFL associated with the above grammar.

The reduction tree is shown below:



Thus by performing a number of reductions, we finally reached the start symbol. So the string aabaa belongs to the above CFL.

2 CFG corresponding to a CFL

Example 1:

Design a CFG for the language,

$$L = \{a^n b^n |\, n \geq 0\}$$

At n=0, ε is a valid string for the above language. The production for this is,

$$S \longrightarrow \varepsilon$$

Every string is of length 2n. The strings are ab, abab, aaabbb, ... etc..

The corresponding production is,

$$S \longrightarrow aSb.$$

Hence the grammar is,

$$S \longrightarrow aSb \,|\, \varepsilon$$

where S is the start symbol.

Example 2:

Design a CFG for the language,

$$L = \{a^n b^{2n} |\, n \geq 0\}$$

At n=0, ε is a valid string for the above language. The production for this is,

$$S \longrightarrow \varepsilon$$

It can be seen that there are double the number of b's as a's. The production is,

$S \longrightarrow aSbb$

So the grammar is,

$S \longrightarrow aSbb \mid \varepsilon$

where S is the start symbol.

Example 3:

Design a CFG for the language, L over {0,1} to generate all possible strings of even length.

Consider number 0 as even. The string of length 0 is ε. The corresponding production is,

$S \longrightarrow \varepsilon$

An even length is of length 2n contains one or more repetitions of 00, 01, 10 or 11. The corresponding production is,

$S \longrightarrow 00S|01S|10S|11S$

Thus the grammar is,

$S \longrightarrow 00S|01S|10S|11S|\varepsilon$

Example 4:

Design a CFG for the language, L over {0,1} to generate all the strings having alternate sequence of 0 and 1.

The minimum string length in L is 2. The strings are 01 or 10.

If a string begins with 01, then it should follow a repetition of 01 and terminate with either 01 or 0.

If a string begins with 10, then it should follow a repetition of 10 and terminate with either 10 or 1.

The grammar will be,

$S \longrightarrow 01|10|01A|10B$

$A \longrightarrow 01A|0|01$

$B \longrightarrow 10B|1|10$

Example 6:

Write a CFG for the regular expression,

$a^*b(a|b)^*$

On analysing this regular expression, we can see that the strings start with any number of a's, followed by a 'b' and may end with a combination of a's and b's.

The CFG can be written as,

$S \longrightarrow AbB$

$A \longrightarrow aA|\varepsilon$

$B \longrightarrow aB|bB|\varepsilon$

For example, using the above productions

$$S \Longrightarrow AbB$$
$$\Longrightarrow aAbB$$
$$\Longrightarrow aAbaB$$
$$\Longrightarrow aaAbaB$$
$$\Longrightarrow aaAbabB$$
$$\Longrightarrow aabab$$

aabab is string belonging to the above CFL.

Example 7:

Write a CFG which generates strings of equal number of a's and b's.

The CFG is

$$S \longrightarrow aSbS|bSaS|\varepsilon$$

For example, using the above productions,

$$S \Longrightarrow bSaS$$
$$\Longrightarrow bbSaSaS$$
$$\Longrightarrow bbaSbSaSaS$$
$$\Longrightarrow bbaSbaSbSaSaS$$
$$\Longrightarrow bbababaa$$

Note that above string contains equal number of a's and b's.

Example 8:

Design a CFG for the regular expression,

$$(a|b)^*$$

The CFG is,

$$S \longrightarrow aS$$
$$S \longrightarrow bS$$
$$S \longrightarrow \varepsilon$$

That is,

$$S \longrightarrow aS|bS|\varepsilon$$

where S is the start symbol.

Example 9;

Design a CFG for the language,

$$L(G) = \{ww^R|w \in \{0,1\}^*\}$$

The CFG is,

$$S \longrightarrow 0S0|1S1|\varepsilon$$

where S is the start symbol.

[In the above, w^R means reverse of string w.]

For example, using the above productions,

$$S \Longrightarrow 0S0$$
$$\Longrightarrow 00S00$$
$$\Longrightarrow 001S100$$
$$\Longrightarrow 001100$$

Example 10:

Design a CFG for the language,

$$L = \{a^n b^m |\, n \neq m\}$$

Case 1: For n>m,

The language is

$$L = \{a^n b^m |\, n > m\}$$

The CFG is,

$$S_1 \longrightarrow AS_1$$
$$S_1 \longrightarrow aS_1b|\varepsilon$$
$$A \longrightarrow aA|a$$

Case 2: For n<m,

The language is

$$L = \{a^n b^m |\, n < m\}$$

The CFG is,

$$S_2 \longrightarrow S_2B$$
$$S_2 \longrightarrow aS_2b|\varepsilon$$
$$B \longrightarrow bB|b$$

Combining Case 1 and Case 2, we get,

$$S \longrightarrow S_1|S_2$$

Thus the CFG is,

$$S \longrightarrow S_1|S_2$$
$$S_1 \longrightarrow AS_1$$
$$S_1 \longrightarrow aS_1b|\varepsilon$$
$$A \longrightarrow aA|a$$
$$S_2 \longrightarrow S_2B$$

$S_2 \longrightarrow aS_2b|\varepsilon$

$B \longrightarrow bB|b$

where S is the start symbol.

Example 11:

Design a CFG for the language,

$L = \{\, (0^n1^n|\, n \geq 0) \,\cup\, (1^n0^n|\, n \geq 0)\,\}$

We can write it as,

$L = L_1 \cup L_2$

Case 1:

Consider L_1,

$L_1 = \{\, 0^n1^n|\, n \geq 0\,\}$

The CFG is,

$S_1 \longrightarrow 0S_11|\varepsilon$

Case 2:

Consider L_2,

$L_2 = \{\, 1^n0^n|\, n \geq 0\,\}$

The CFG is,

$S_2 \longrightarrow 1S_20|\varepsilon$

Then we have $L = L_1 \cup L_2$

Combining above two cases, we get the CFG as,

$S \longrightarrow S_1|S_2$

Thus the complete CFG is,

$S \longrightarrow S_1|S_2$

$S_1 \longrightarrow 0S_11|\varepsilon$

$S_2 \longrightarrow 1S_20|\varepsilon$

where S is the start symbol.

Example 12:

Design a CFG for the language, L

$L = \{\, a^nb^{2n}|\, n \geq 0\,\}$

The CFG is,

$S \longrightarrow aSbb|\varepsilon$

Example 13;

Write a CFG for the language,

$L = \{\, a^{2n}b^m |\, n > 0,\, m \geq 0 \,\}$

From the above we can say that L is the set of strings starting with even number of a's followed by any number of b's.

There is no 'a' in the string after first 'b' is encountered. Also there is no 'b' before any 'a' in the string.

The CFG is,

$S \longrightarrow aaAB$

$A \longrightarrow aaA|\varepsilon$

$B \longrightarrow bB|\varepsilon$

where S is the start symbol.

Example 13:

Write a CFG for the language,

$L = \{a^n b^{2n} c^m |\, n, m \geq 0\}$

This means strings start with 'a' or 'c', but not with a 'b'.

If the string starts with 'a', then number of a's must follow b's, and the number of b's is twice than number of a's.

If the string starts with 'c', it is followed by any number of c's.

There is no 'a' after any 'b' or any 'c'.

There is no 'b' after any 'c'.

There is no 'b' or 'c' after any 'a'.

There is no 'c' after any 'b'.

Thus the CFG is,

$S \longrightarrow AB$

$A \longrightarrow aAbb|\varepsilon$

$B \longrightarrow cB|\varepsilon$

where S is the start symbol.

Example 14:

Design a CFG for the language,

$L = \{a^n b^m c^{2m} |\, n, m \geq 0\}$

The CFG is

$S \longrightarrow AB$

$A \longrightarrow aA|\varepsilon$

$B \longrightarrow bBcc|\varepsilon$

Example 15:

Design a CFG for the language, $L = \{wcw^R | w \in (a, b)^*\}$

The CFG is

$$S \longrightarrow aSa$$

$$S \longrightarrow bSb$$

$$S \longrightarrow c$$

where S is the start symbol.

Part II. Simplification of Context Free Grammars

3 Simplification of CFGs

We can simplify a CFG and produce an equivalent reduced CFG. This is done by,

a. Eliminating useless symbols,

b. Eliminating ε productions,

c. Eliminating unit productions.

3.1 Eliminating Useless Symbols

Useless symbols are those non-terminals or terminals that do not appear in any derivation of a string.

A symbol, Y is said to be useful if:

a. $Y \stackrel{*}{\Longrightarrow} w$,

that is Y should lead to a set of terminals. Here Y is said to be 'generating'.

b. If there is a derivation,

$$S \stackrel{*}{\Longrightarrow} \alpha Y \beta \stackrel{*}{\Longrightarrow} w,$$

then Y is said to be 'reachable'.

Thus a symbol is useful, if it is 'generating' and 'useful'.

Thus a symbol is useless, if it is not 'generating' and not 'reachable'.

Example 1:

Consider the CFG,

$$S \longrightarrow AB|a$$

$$A \longrightarrow b$$

where S is the start symbol.

Eliminate uselss symbols from this grammar.

By observing the above CFG, it is clear that B is a non generating symbol. Since A derives 'b', S derives 'a' but B does not derive any string 'w'.

So we can eliminate $S \longrightarrow AB$ from the CFG.

Now CFG becomes,

$S \longrightarrow a$

$A \longrightarrow b$

Here A is a non reachable symbol, since it cannot be reached from the start symbol S.

So we can eliminate the production, $A \longrightarrow b$ from the CFG.

Now the reduced grammar is,

$S \longrightarrow a$

This grammar does not contain any useless symbols.

Example 2:

Consider the CFG,

$S \longrightarrow aB|bX$

$A \longrightarrow BAd|bSX|a$

$B \longrightarrow aSB|bBX$

$X \longrightarrow SBD|aBx|ad$

where S is the start symbol.

Eliminate uselss symbols from this grammar.

First we choose those non terminals which derive to the strings of terminals.

The non terminals,

A derives to the terminal 'a';

X derives to the terminals 'ad'.

So A and X are useful symbols.

Since,

$S \longrightarrow bX$, and X is a useful symbol, S is also a useful symbol.

From the production, $B \longrightarrow aSB|bBX$,

B does not derive any terminal string, B is a non-generating symbol. So eliminate those productions containing B.

Grammar becomes,

$S \longrightarrow bX$

$A \longrightarrow bSX|a$

$X \longrightarrow ad$

From the above CFG, A is a non- reachable symbol, since A cannot be reached from the start symbol, S.

So eliminate the production, $A \longrightarrow bSX|a$.

Now the CFG is,

$$S \longrightarrow bX$$

$$X \longrightarrow ad$$

This grammar does not contain any useless symbols.

Example 3:

Consider the CFG,

$$A \longrightarrow xyz|Xyzz$$

$$X \longrightarrow Xz|xYz$$

$$Y \longrightarrow yYy|Xz$$

$$Z \longrightarrow Zy|z$$

where A is the start symbol.

Eliminate uselss symbols from this grammar.

From the productions,

$$A \longrightarrow xyz,\ Z \longrightarrow z,$$

A and Z derive to the strings of terminals. So A and Z are useful symbols.

X and Y do not lead to a string of terminals; that means X and Y are useless symbols.

Eliminating the productions of X and Y, we get,

$$A \longrightarrow xyz$$

$$Z \longrightarrow Zy|z$$

From the above, Z is not reachable, since it cannot be reached from the start symbol, A. So eliminate the production corresponding to Z.

The CFG is,

$$A \longrightarrow xyz.$$

This grammar does not contain any useless symbols.

Example 4:

Consider the CFG,

$$S \longrightarrow aC|SB$$

$$A \longrightarrow bSCa$$

$$B \longrightarrow aSB|bBC$$

$$C \longrightarrow aBC|ad$$

where S is the start symbol.

Eliminate uselss symbols from this grammar.

Since, $C \longrightarrow ad$,

C is a generating symbol.

Since, $S \longrightarrow aC$,

S is also a useful symbol.

Since, $A \longrightarrow bSCa$,

A is also a useful symbol.

The RHS of $B \longrightarrow aSB|bBC$ contains B. B is not terminating. B is not generating.

So B is a useless symbol. Eliminate those productions containing B.

We get,

$$S \longrightarrow aC$$

$$A \longrightarrow bSCa$$

$$C \longrightarrow ad$$

From the above, A is not reachable, since it cannot be reached from the start symbol, S.

So eliminate the production corresponding to A.

We get the CFG as,

$$S \longrightarrow aC$$

$$C \longrightarrow ad$$

where S is the start symbol.

This grammar does not contain any useless symbols.

3.2 Removal of Unit Productions

A unit production is defined as,

$$A \longrightarrow B$$

where A is a non-terminal, and

B is a non-terminal.

Thus both LHS and RHS contain single non-terminals.

Following examples show how unit productions are removed from a CFG.

Example 1:

Consider the CFG,

$$S \longrightarrow AB$$

$A \longrightarrow a$

$B \longrightarrow C|b$

$C \longrightarrow D$

$D \longrightarrow E$

$E \longrightarrow a$

where S is the start symbol.

Eliminate unit productions from this grammar.

From the above CFG, it is seen that,

$B \longrightarrow C$

$C \longrightarrow D$

$D \longrightarrow E$

are unit productions.

To remove the production, $B \longrightarrow C$, check whether there exists a production whose LHS is C and RHS is a terminal. No such production exists.

To remove the production, $C \longrightarrow D$, check whether there exists a production whose LHS is D and RHS is a terminal. No such production exists.

To remove the production, $D \longrightarrow E$, check whether there exists a production whose LHS is E and RHS is a terminal.

There is a production, $E \longrightarrow a$. So remove, $D \longrightarrow E$, and add the production, $D \longrightarrow a$, CFG becomes,

$S \longrightarrow AB$

$A \longrightarrow a$

$B \longrightarrow C|b$

$C \longrightarrow D$

$D \longrightarrow a$

$E \longrightarrow a$

Now remove the production, $C \longrightarrow D$, and add the production, $C \longrightarrow a$, we get,

$S \longrightarrow AB$

$A \longrightarrow a$

$B \longrightarrow C|b$

$C \longrightarrow a$

$D \longrightarrow a$

$E \longrightarrow a$

Now remove the production, $B \longrightarrow C$, and add the production, $B \longrightarrow a$, we get,

$S \longrightarrow AB$

$A \longrightarrow a$

$B \longrightarrow a|b$

$C \longrightarrow a$

$D \longrightarrow a$

$E \longrightarrow a$

where S is the start symbol.

Now the grammar contains no unit productions.

From the above CFG, it is seen that the productions, $C \longrightarrow a$, $D \longrightarrow a$, $E \longrightarrow a$ are useless becuse the symbols C, D and E cannot be reached from the start symbol, S.

By eliminating these productions, we get the CFG as,

$S \longrightarrow AB$

$A \longrightarrow a$

$B \longrightarrow a|b$

Above is a completely reduced grammar.

Example 2:

Consider the CFG,

$S \longrightarrow a|b|Sa|Sb|S0|S1$

$F \longrightarrow S|(E)$

$T \longrightarrow F|T * F$

$E \longrightarrow T|E + T$

where E is the start symbol.

Eliminate unit productions from this grammar.

From the above CFG,

$F \longrightarrow S$

$T \longrightarrow F$

$E \longrightarrow T$

are unit productions.

The unit production $F \longrightarrow S$ can be removed by rewriting it as, $F \longrightarrow a|b|Sa|Sb|S0|S1$

Now the CFG is,

$S \longrightarrow a|b|Sa|Sb|S0|S1$

$F \longrightarrow a|b|Sa|Sb|S0|S1|(E)$

$T \longrightarrow F|T * F$

$E \longrightarrow T|E + T$

The unit production $T \longrightarrow F$ can be removed by rewriting it as, $T \longrightarrow a|b|Sa|Sb|S0|S1|(E)$

Now the CFG is,

$$S \longrightarrow a|b|Sa|Sb|S0|S1$$

$$F \longrightarrow a|b|Sa|Sb|S0|S1|(E)$$

$$T \longrightarrow a|b|Sa|Sb|S0|S1|(E)|T * F$$

$$E \longrightarrow T|E + T$$

The unit production $E \longrightarrow T$ can be removed by rewriting it as, $E \longrightarrow a|b|Sa|Sb|S0|S1|(E)|T * F$

Now the CFG is,

$$S \longrightarrow a|b|Sa|Sb|S0|S1$$

$$F \longrightarrow a|b|Sa|Sb|S0|S1|(E)$$

$$T \longrightarrow a|b|Sa|Sb|S0|S1|(E)|T * F$$

$$E \longrightarrow a|b|Sa|Sb|S0|S1|(E)|T * F|E + T$$

This is the CFG that does not contain any unit productions.

Example 1:

Consider the CFG,

$$S \longrightarrow A|bb$$

$$A \longrightarrow B|b$$

$$B \longrightarrow S|a$$

where S is the start symbol.

Eliminate unit productions from this grammar.

Here unit productions are,

$$S \longrightarrow A$$

$$A \longrightarrow B$$

$$B \longrightarrow S$$

Here,

$S \longrightarrow A$ generates $S \longrightarrow b$

$S \longrightarrow A \longrightarrow B$ generates $S \longrightarrow a$

$A \longrightarrow B$ generates $A \longrightarrow a$

$A \longrightarrow B \longrightarrow S$ generates $A \longrightarrow bb$

$B \longrightarrow S$ generates $B \longrightarrow bb$

$B \longrightarrow S \longrightarrow A$ generates $B \longrightarrow b$

The new CFG is,

$$S \longrightarrow b|a|bb$$

$$A \longrightarrow a|bb|b$$

$$B \longrightarrow bb|b|a$$

which contains no unit productions.

3.3 Removal of ε- Productions

Productions of the form, $A \longrightarrow \varepsilon$ are called ε -productions.

We can eliminate ε -productions from a grammar in the following way:

If $A \longrightarrow \varepsilon$ is a production to be eliminated, then we look for all productions, whose RHS contains A, and replace every occurrence of A in each of these productions to obtain non ε -productions. The resultant non ε -productions are added to the grammar.

Example 1:

Consider the CFG,

$$S \longrightarrow aA$$

$$A \longrightarrow b|\varepsilon$$

where S is the start symbol.

Eliminate epsilon productions from this grammar.

Here, $A \longrightarrow \varepsilon$ is an epsilon production.

By following the above procedure, put ε in place of A, at the RHS of productions, we get,

The production, $S \longrightarrow aA$ becomes, $S \longrightarrow a$.

Then the CFG is,

$$S \longrightarrow aA$$

$$S \longrightarrow a$$

$$A \longrightarrow b$$

OR

$$S \longrightarrow aA|a$$

$$A \longrightarrow b$$

Thus this CFG does not contain any epsilon productions.

Example 2:

Consider the CFG,

$$S \longrightarrow ABAC$$

$$A \longrightarrow aA|\varepsilon$$

$$B \longrightarrow bB|\varepsilon$$

$$C \longrightarrow c$$

where S is the start symbol.

Eliminate epsilon productions from this grammar.

This CFG contains the epsilon productions, $A \longrightarrow \varepsilon$, $B \longrightarrow \varepsilon$.

To eliminate $A \longrightarrow \varepsilon$, replace A with epsion in the RHS of the productions, $S \longrightarrow ABAC$, $A \longrightarrow aA$,

For the production, $S \longrightarrow ABAC$, replace A with epsilon one by one as,

we get,

$S \longrightarrow BAC$

$S \longrightarrow ABC$

$S \longrightarrow BC$

For the production, $A \longrightarrow aA$, we get,

$A \longrightarrow a$,

Now the grammar becomes,

$S \longrightarrow ABAC|ABC|BAC|BC$

$A \longrightarrow aA|a$

$B \longrightarrow bB|\varepsilon$

$C \longrightarrow c$

To eliminate $B \longrightarrow \varepsilon$, replace A with epsion in the RHS of the productions, $S \longrightarrow ABAC$, $B \longrightarrow bB$,

For the production, $S \longrightarrow ABAC|ABC|BAC|BC$, replace B with epsilon as,

we get,

$S \longrightarrow AAC|AC|C$

For the production, $B \longrightarrow bB$, we get,

$B \longrightarrow b$,

Now the grammar becomes,

$S \longrightarrow ABAC|ABC|BAC|BC|AAC|AC|C$

$A \longrightarrow aA|a$

$B \longrightarrow bB|b$

$C \longrightarrow c$

which does not contain any epsilon production.

Example 3:

Consider the CFG,

$S \longrightarrow aSa$

$$S \longrightarrow bSb|\varepsilon$$

where S is the start symbol.

Eliminate epsilon productions from this grammar.

Here, $S \longrightarrow \varepsilon$ is an epsilon production. Replace the occurrence of S by epsilon, we get,

$$S \longrightarrow aa,$$

$$S \longrightarrow bb$$

Now the CFG becomes,

$$S \longrightarrow aSa|bSb|aa|bb$$

This does not contain epsilon productions.

Example 4:

Consider the CFG,

$$S \longrightarrow a|Xb|aYa$$

$$X \longrightarrow Y|\varepsilon$$

$$Y \longrightarrow b|X$$

where S is the start symbol.

Eliminate epsilon productions from this grammar.

Here, $X \longrightarrow \varepsilon$ is an epsilon production. Replace the occurrence of X by epsilon, we get,

$$S \longrightarrow a|b$$

$$X \longrightarrow Y$$

$$Y \longrightarrow \varepsilon$$

Now the grammar becomes,

$$S \longrightarrow a|Xb|aYa|a|b$$

$$X \longrightarrow Y$$

$$Y \longrightarrow b|X|\varepsilon$$

In the above CFG, $Y \longrightarrow \varepsilon$ is an epsilon production. Replace the ocurrence of Y by epsilon, we get,

$$S \longrightarrow aa$$

$$X \longrightarrow \varepsilon$$

Now the grammar becomes,

$$S \longrightarrow a|Xb|aYa|a|b|aa$$

$$X \longrightarrow Y$$

$$Y \longrightarrow b|X$$

This grammar does not contain epsilon productions.

Exercises:

Simplify the following context free grammars:

1.

$S \longrightarrow AB$

$A \longrightarrow a$

$B \longrightarrow b$

$B \longrightarrow C$

$E \longrightarrow c|\varepsilon$

where S is the start symbol.

2.

$S \longrightarrow aAa$

$A \longrightarrow Sb|bCC|DaA$

$C \longrightarrow abb|DD$

$E \longrightarrow aC$

$D \longrightarrow aDA$

where S is the start symbol.

3.

$S \longrightarrow bS|bA|\varepsilon$

$A \longrightarrow \varepsilon$

where S is the start symbol.

4.

$S \longrightarrow bS|AB$

$A \longrightarrow \varepsilon$

$B \longrightarrow \varepsilon$

$D \longrightarrow a$

where S is the start symbol.

5.

$S \longrightarrow A$

$A \longrightarrow B$

$B \longrightarrow C$

$C \longrightarrow a$

where S is the start symbol.

6.

$S \longrightarrow AB$

$A \longrightarrow a$

$B \longrightarrow C|b$

$C \longrightarrow D$

$D \longrightarrow E$

$E \longrightarrow a$

where S is the start symbol.

7.

$S \longrightarrow ABCD$

$A \longrightarrow a$

$B \longrightarrow C|b$

$C \longrightarrow D$

$D \longrightarrow c$

where S is the start symbol.

8.

$S \longrightarrow aS|AC|AB$

$A \longrightarrow \varepsilon$

$C \longrightarrow \varepsilon$

$B \longrightarrow bB|bb$

$D \longrightarrow c$

where S is the start symbol.

9.

$S \longrightarrow ABC|A0A$

$A \longrightarrow 0A|B0C|000|B$

$B \longrightarrow 1B1|0|D$

$C \longrightarrow CA|AC$

$D \longrightarrow \varepsilon$

where S is the start symbol.

10.

$S \longrightarrow AAA|B$

$A \longrightarrow 0A|B$

$B \longrightarrow \varepsilon$ where S is the start symbol.

Part III. Normal Forms for CFGs

Standard form or normal forms have been developed for context free grammars. Any CFG can be converted to these forms.

The advantages of representing a CFG in this form are,

1. Complexity of the CFG will be less, and
2. It will be easier to implement the CFG.

Two commonly used normal forms for CFGs are,

1. Chomsky normal form (CNF), and
2. Greibach normal form (GNF).

4 Chomsky Normal Form (CNF)

A context free grammar, G is in Chomsky Normal Form, if every production is of the form,

$A \longrightarrow a$,

$A \longrightarrow BC$,

$S \longrightarrow \varepsilon$; for this production, S is the start symbol.

Thus the RHS of a production can be a single terminal, or two non-terminals. Also it can be ε, if LHS is the start symbol.

Example 1:

Following CFG is in Chomsky normal form (CNF):

$S \longrightarrow AB|b|\varepsilon$,

$A \longrightarrow BC|a$,

$B \longrightarrow b$,

$C \longrightarrow c$

where S is the start symbol.

Example 2:

Following CFG is not in Chomsky normal form (CNF):

$S \longrightarrow ASB|AB|b$,

$A \longrightarrow BC|a$,

$A \longrightarrow a$,

$B \longrightarrow bb$

where S is the start symbol.

The above is not in CNF because the productions,

$S \longrightarrow ASB$,

$B \longrightarrow bb$

do not satisfy the conditions of CNF.

4.1 Conversion to CNF

Every CFG can be reduced to CNF. Following examples show how this is done.

Example 1:

Consider the following CFG,

$S \longrightarrow bA|aB,$

$A \longrightarrow bAA|aS|a,$

$B \longrightarrow aBB|bS|b$

where S is the start symbol.

Convert this CFG to CNF.

Consider the production,

$S \longrightarrow bA|aB$

This is not in CNF. So replace 'b' by non-terminal, P and 'a' by non-terminal, Q. Thsi is by adding the productions, $P \longrightarrow b$, $Q \longrightarrow a$.

,Then we get,

$S \longrightarrow PA|QB$

$P \longrightarrow b$

$Q \longrightarrow a$

Now the CFG becomes,

$S \longrightarrow PA|QB$

$P \longrightarrow b$

$Q \longrightarrow a$

$A \longrightarrow PAA|QS|a,$

$B \longrightarrow QBB|PS|b$

In the above the productions,

$A \longrightarrow PAA,$

$B \longrightarrow QBB$

are not in CNF. So replace AA by R and BB by T. this is done by adding the productions, $R \longrightarrow AA$, $T \longrightarrow BB$.

Then we get,

$A \longrightarrow PR,$

$B \longrightarrow QT$

$R \longrightarrow AA,$

$T \longrightarrow BB$

Now the CFG becomes,

$S \longrightarrow PA|QB$

$P \longrightarrow b$

$Q \longrightarrow a$

$A \longrightarrow PR|QS|a,$

$B \longrightarrow QT|PS|b$

$R \longrightarrow AA,$

$T \longrightarrow BB$

where S is the start symbol.

Note that above CFG is in CNF.

Example 2:

Consider the following CFG,

$S \longrightarrow 1A|0B,$

$A \longrightarrow 1AA|0S|0,$

$B \longrightarrow 0BB|1$

where S is the start symbol.

Convert this CFG to CNF.

Consider the production, $S \longrightarrow 1A|0B,$

This is not in CNF. So replace 1 by P and 0 by Q, by adding the productions, $P \longrightarrow 1, Q \longrightarrow 0,$

we get,

$S \longrightarrow PA|QB$

$P \longrightarrow 1$

$Q \longrightarrow 0$

Now the CFG is,

$S \longrightarrow PA|QB$

$P \longrightarrow 1$

$Q \longrightarrow 0$

$A \longrightarrow PAA|QS|0,$

$B \longrightarrow QBB|1$

In the above CFG, the productions,

$A \longrightarrow PAA$

$B \longrightarrow QBB$

are not in CNF.

So replace AA by R and BB by T. Then we get,

$$A \longrightarrow PR$$

$$B \longrightarrow QT$$

$$R \longrightarrow AA$$

$$T \longrightarrow BB$$

Now the CFG is,

$$S \longrightarrow PA|QB$$

$$P \longrightarrow 1$$

$$Q \longrightarrow 0$$

$$A \longrightarrow PR|QS|0,$$

$$B \longrightarrow QT|1$$

$$R \longrightarrow AA$$

$$T \longrightarrow BB$$

where S is the start symbol.

Above CFG is in CNF.

Example 3:

Consider the following CFG,

$$S \longrightarrow a|b|CSS,$$

where S is the start symbol.

Convert this CFG to CNF.

In the above, the production, $S \longrightarrow CSS$,

is not in CNF. So replace SS by P, we get,

$$S \longrightarrow CP$$

$$P \longrightarrow SS$$

Now the CFG becomes,

$$S \longrightarrow a|b|CP$$

$$P \longrightarrow SS$$

where S is the start symbol.

Above CFG is in CNF.

Example 4:

Consider the following CFG,

$$S \longrightarrow abSb|a|aAb,$$

$A \longrightarrow bS|aAAb,$

where S is the start symbol.

Convert this CFG to CNF.

Consider the production, $S \longrightarrow abSb|aAb$ is not in CNF.

So replace Ab by P , we get,

$S \longrightarrow abSb|aP$

$P \longrightarrow Ab$

Now the grammar becomes,

$S \longrightarrow abSb|a|aP$

$P \longrightarrow Ab$

$A \longrightarrow bS|aAP$

Now replace Sb by R, we get,

$S \longrightarrow abR|a|aP$

$P \longrightarrow Ab$

$A \longrightarrow bS|aAP$

$R \longrightarrow Sb$

Now replace bR with T, we get

$S \longrightarrow aT|a|aP$

$P \longrightarrow Ab$

$A \longrightarrow bS|aAP$

$R \longrightarrow Sb$

$T \longrightarrow bR$

Now replace aA with U, we get,

$S \longrightarrow aT|a|aP$

$P \longrightarrow Ab$

$A \longrightarrow bS|UP$

$R \longrightarrow Sb$

$T \longrightarrow bR$

$U \longrightarrow aA$

Now replace a with V and b with W, we get,

$S \longrightarrow VT|a|VP$

$P \longrightarrow AW$

$$A \longrightarrow WS|UP$$

$$R \longrightarrow SW$$

$$T \longrightarrow WR$$

$$U \longrightarrow VA$$

$$V \longrightarrow a$$

$$W \longrightarrow b$$

where S is the start symbol.

The above CFG is in CNF.

Exercises:

Convert the following CFGs to CNF.

1.

$$S \longrightarrow bA|aB$$

$$A \longrightarrow bAA|aS|a$$

$$B \longrightarrow aBB|bS|b$$

where S is the start symbol.

2.

$$S \longrightarrow aAD$$

$$A \longrightarrow aB|bAB$$

$$B \longrightarrow b$$

$$D \longrightarrow d$$

where S is the start symbol.

3.

$$S \longrightarrow aAbB$$

$$A \longrightarrow aA|a$$

$$B \longrightarrow bB|b$$

where S is the start symbol.

4.

$$S \longrightarrow aAbB$$

$$A \longrightarrow Ab|b$$

$$B \longrightarrow Ba|a$$

where S is the start symbol.

5.

$S \longrightarrow aA|bB$

$A \longrightarrow bAA|a$

$B \longrightarrow BBa|b$

where S is the start symbol.

5.

$S \longrightarrow abSb|ab$

where S is the start symbol.

5 Greibach Normal Form (GNF)

A context free grammar, G is in Greibach nromal form, if every production is of the form,

$A \longrightarrow aX$

$S \longrightarrow \varepsilon$, for this production, S should be start symbol.

where a is a single terminal,

X is a set of 0 or more non-terminals.

Example 1:

The following CFG is in Greibach normal form (GNF):

$S \longrightarrow aAB|\varepsilon$

$A \longrightarrow bC$

$B \longrightarrow b$

$C \longrightarrow c$

where S is the start symbol.

Example 2:

The following CFG is not in Greibach normal form (GNF):

$S \longrightarrow aAB$

$A \longrightarrow BbC$

$B \longrightarrow b|\varepsilon$

$C \longrightarrow c$

where S is the start symbol.

The above is not in GNF becuause the productions, $A \longrightarrow BbC$, $B \longrightarrow \varepsilon$ do not satisfy the conditions of GNF.

5.1 Conversion to GNF

Every CFG can be converted to Greibach normal form (GNF).

Following are the steps:

Step 1: Convert the given CFG to CNF.

Rename all non-terminals by $A_1, A_2, ...A_n$, where A_1 is the start symbol.

Step 2: Modify the grammar so that every production is of the form,

$A_i \longrightarrow a\gamma$, or

$A_i \longrightarrow A_j\gamma$, where j>i, and

γ is a set of 0 or more non-terminals.

This can be done by performing a number of substitutions.

Now if a production is of the form,

$A \longrightarrow A\gamma$, it is in left recursive form.

This left recursion can be eliminated by using a new variable, Z.

This is done as follows:

Let there be a production, $A \longrightarrow ADF|Aa|a|b|c$

This can be written as,

$$A \longrightarrow a|b|c|aZ|bZ|cZ$$

$$Z \longrightarrow DF|a|DFZ|aZ$$

In a production leftmost symbol of RHS must be a terminal as,

$A_n \longrightarrow \alpha\gamma$,

otherwise if it is of the form,

$A_n \longrightarrow A_m\gamma$

then replace A_m on RHS of production of A_{m-1} by replacement rule.

Thus in short, to convert a grammar to GNF, first start with grammar in CNF and then apply substitution rule and eliminate left recursion.

Example 1:

Consider the CFG,

$S \longrightarrow AB|BC$

$A \longrightarrow aB|bA|a$

$B \longrightarrow bB|cC|b$

$C \longrightarrow c$

Convert this grammar to Greibach normal form (GNF).

The production, $S \longrightarrow AB|BC$ is not in GNF.

On applying the substitution rule we get,

$S \longrightarrow aBB|bAB|aB|bBC|cCC|bC$

Now the CFG becomes,

$S \longrightarrow aBB|bAB|aB|bBC|cCC|bC$

$A \longrightarrow aB|bA|a$

$B \longrightarrow bB|cC|b$

$C \longrightarrow c$

Example 2:

Consider the CFG,

$S \longrightarrow abaSa|aba$

Convert this grammar to Greibach normal form (GNF).

Replace a by A using the production, $A \longrightarrow a$,

Replace b by B using the production, $B \longrightarrow b$,

we get,

$S \longrightarrow aBASA|aBA$

$A \longrightarrow a$

$B \longrightarrow b$

Now the above CFG is in GNF.

Example 3:

Consider the CFG,

$S \longrightarrow AB$

$A \longrightarrow aA|bB|b$

$B \longrightarrow b$

Convert this grammar to Greibach normal form (GNF).

In the above CFG, the production,

$S \longrightarrow AB$

is not in GNF.

So replace this by,

$S \longrightarrow aAB|bBB|bB$

Now the CFG becomes,

$S \longrightarrow aAB|bBB|bB$

$A \longrightarrow aA|bB|b$

$B \longrightarrow b$

The above CFG is in Greibach normal form.

Example 4:

Consider the CFG,

$S \longrightarrow abSb|aa$

Convert this grammar to Greibach normal form (GNF).

The above production is not in GNF.

So introduce two productions,

$A \longrightarrow a$

$B \longrightarrow b$

Now the CFG becomes,

$S \longrightarrow aBSB|aA$

$A \longrightarrow a$

$B \longrightarrow b$

Above CFG is in GNF.

Example 5:

Consider the CFG,

$S \longrightarrow aSa|bSb|aa|bb$

Convert this grammar to Greibach normal form (GNF).

Introduce two productions,

$A \longrightarrow a$

$B \longrightarrow b$

Then the grammar becomes,

$S \longrightarrow aSP|bSQ|aP|bQ$

$A \longrightarrow a$

$B \longrightarrow b$

Above CFG is in GNF.

Example 6:

Consider the CFG,

$S \longrightarrow AA|a$

$A \longrightarrow SS|b$

Convert this grammar to Greibach normal form (GNF).

Step 1:

Above CFG is in CNF.

Rename the variables, that is,

Rename S, A by A_1, A_2.

Then the grammar becomes,

$A_1 \longrightarrow A_2A_2|a$

$A_2 \longrightarrow A_1A_1|b$

Step 2: In this, productions must be of the form that

the RHS of a production must start with a terminal followed by nonterminals, or

the RHS of a production starts with a higher numbered variable.

The productions,

$A_1 \longrightarrow A_2A_2|a$

$A_2 \longrightarrow b$

satisfy this criterion.

Consider the production,

$A_2 \longrightarrow A_1A_1$

Replace the first A_1 on the RHS by its production.

Then,

$A_2 \longrightarrow A_2A_2A_1|aA_1$

Now the CFG is,

$A_1 \longrightarrow A_2A_2|a$

$A_2 \longrightarrow b$

$A_2 \longrightarrow A_2A_2A_1|aA_1$

That is,

$A_1 \longrightarrow A_2A_2|a$

$A_2 \longrightarrow A_2A_2A_1|aA_1|b$

The production,$A_2 \longrightarrow A_2A_2A_1|aA_1|b$ contains left recursion.

To eliminate left recursion, this is written as,

$A_2 \longrightarrow aA_1|b|aA_1Z|bZ$

$Z \longrightarrow A_2A_1|A_2A_1Z$

Now the CFG is,

$A_1 \longrightarrow A_2A_2|a$

$A_2 \longrightarrow aA_1|b|aA_1Z|bZ$

$Z \longrightarrow A_2A_1|A_2A_1Z$

Consider the production, $A_1 \longrightarrow A_2A_2$

Replace first A_2 on the RHS with its production, we get,

$A_1 \longrightarrow aA_1A_2|bA_2|aA_1ZA_2|bZA_2$

Now the CFG is

$A_1 \longrightarrow aA_1A_2|bA_2|aA_1ZA_2|bZA_2$

$A_2 \longrightarrow aA_1|b|aA_1Z|bZ$

$Z \longrightarrow A_2A_1|A_2A_1Z$

Consider the production, $Z \longrightarrow A_2A_1|A_2A_1Z$

Replace first A_2 on the RHS with its production,

$Z \longrightarrow aA_1A_1|aA_1A_1Z|bA_1|bA_1Z|aA_1ZA_1|aA_1ZA_1Z|bZA_1|bZA_1Z$

Now the CFG is,

$A_1 \longrightarrow aA_1A_2|bA_2|aA_1ZA_2|bZA_2$

$A_2 \longrightarrow aA_1|b|aA_1Z|bZ$

$Z \longrightarrow aA_1A_1|aA_1A_1Z|bA_1|bA_1Z|aA_1ZA_1|aA_1ZA_1Z|bZA_1|bZA_1Z$

Above CFG is in GNF.

Example 8:

Consider the CFG,

$S \longrightarrow AB$

$A \longrightarrow BS|a$

$B \longrightarrow SA|b$

Convert this grammar to Greibach normal form (GNF).

Step 1: The given grammar is in Chomsky normal form.

Rename the variables, that is,

Rename S, A, B by A_1, A_2, A_3.

Then the grammar becomes,

$A_1 \longrightarrow A_2A_3$

$A_2 \longrightarrow A_3A_1|a$

$A_3 \longrightarrow A_1A_2|b$

Step 2: In this, productions must be of the form that

the RHS of a production must start with a terminal followed by nonterminals, or

the RHS of a production starts with a higher numbered variable.

The productions,

$A_1 \longrightarrow A_2A_3$

$A_2 \longrightarrow A_3A_1|a$

$A_3 \longrightarrow b$

satisfy this criterion.

But the production,

$A_3 \longrightarrow A_1A_2$ does not satisfy this.

By applying substitution rule, substitute for A_1 as

$A_3 \longrightarrow A_2A_3A_2$

substitute for A_2 as

$A_3 \longrightarrow A_3A_1A_3A_2|aA_3A_2$

Now the CFG is,

$A_1 \longrightarrow A_2A_3$

$A_2 \longrightarrow A_3A_1|a$

$A_3 \longrightarrow A_3A_1A_3A_2|aA_3A_2|b$

Consider the production,

$A_3 \longrightarrow A_3A_1A_3A_2|aA_3A_2|b$

Eliminating left recursion from the above production, we get,

$A_3 \longrightarrow aA_3A_2B_3|bB_3|aA_3A_2|b$

$B_3 \longrightarrow A_1A_3A_2B_3|A_1A_3A_2$

Now we have the productions,

$A_1 \longrightarrow A_2A_3$

$A_2 \longrightarrow A_3A_1|a$

$A_3 \longrightarrow aA_3A_2B_3|bB_3|aA_3A_2|b$

$B_3 \longrightarrow A_1A_3A_2B_3|A_1A_3A_2$

Now all the A_3 productions start with terminals.

Using substitution rule, replace A_3 in the RHS of the production for A_2.

$A_2 \longrightarrow A_3A_1|a$ becomes,

$A_2 \longrightarrow aA_3A_2B_3A_1|bB_3A_1|aA_3A_2A_1|bA_1|a$

Now the CFG is,

$A_1 \longrightarrow A_2A_3$

$A_2 \longrightarrow aA_3A_2B_3A_1|bB_3A_1|aA_3A_2A_1|bA_1|a$

$A_3 \longrightarrow aA_3A_2B_3|bB_3|aA_3A_2|b$

$B_3 \longrightarrow A_1A_3A_2B_3|A_1A_3A_2$

Using substitution rule, replace A_2 in the RHS of the production for A_1.

$A_1 \longrightarrow aA_3A_2B_3A_1A_3|bB_3A_1A_3|aA_3A_2A_1A_3|bA_1A_3|aA_3$

Now the CFG is,

$A_1 \longrightarrow aA_3A_2B_3A_1A_3|bB_3A_1A_3|aA_3A_2A_1A_3|bA_1A_3|aA_3$

$A_2 \longrightarrow aA_3A_2B_3A_1|bB_3A_1|aA_3A_2A_1|bA_1|a$

$A_3 \longrightarrow aA_3A_2B_3|bB_3|aA_3A_2|b$

$B_3 \longrightarrow A_1A_3A_2B_3|A_1A_3A_2$

Using substitution rule, replace A_1 in the RHS of the production for B_3.

$B_3 \longrightarrow aA_3A_2B_3A_1A_3A_3A_2B_3|aA_3A_2B_3A_1A_3A_3A_2|bB_3A_1A_3A_3A_2B_3|bB_3A_1A_3A_3A_2|aA_3A_2A_1A_3A_3A_2B_3|aA_3A_2$

Now the CFG is,

$A_1 \longrightarrow aA_3A_2B_3A_1A_3|bB_3A_1A_3|aA_3A_2A_1A_3|bA_1A_3|aA_3$

$A_2 \longrightarrow aA_3A_2B_3A_1|bB_3A_1|aA_3A_2A_1|bA_1|a$

$A_3 \longrightarrow aA_3A_2B_3|bB_3|aA_3A_2|b$

$$B_3 \longrightarrow aA_3A_2B_3A_1A_3A_3A_2B_3|aA_3A_2B_3A_1A_3A_3A_2|bB_3A_1A_3A_3A_2B_3|bB_3A_1A_3A_3A_2|aA_3A_2A_1A_3A_3A_2B_3|aA_3A_2$$

Above CFG is in Greibach normal form.

Exercises:

1.

Consider the CFG,

$$A \longrightarrow aBD|bDB|c|AB|AD$$

Convert this grammar to Greibach normal form (GNF).

2.

Consider the CFG,

$$S \longrightarrow AB$$

$$A \longrightarrow BS|b$$

$$B \longrightarrow SA|a$$

Convert this grammar to Greibach normal form (GNF).

3.

Consider the CFG,

$$E \longrightarrow E + T|T$$

$$T \longrightarrow T * F|F$$

$$F \longrightarrow (E)|a$$

Convert this grammar to Greibach normal form (GNF).

4.

Consider the CFG,

$$S \longrightarrow YY|0$$

$$Y \longrightarrow SS|1$$

Convert this grammar to Greibach normal form (GNF).

5.

Consider the CFG,

$$S \longrightarrow XY$$

$$X \longrightarrow YS|1$$

$$Y \longrightarrow SX|0$$

Convert this grammar to Greibach normal form (GNF).

6.

Consider the CFG,

$$S \longrightarrow XY$$

$$X \longrightarrow 0X|1Y|1$$

$$Y \longrightarrow 1$$

Convert this grammar to Greibach normal form (GNF).

7.

Consider the CFG,

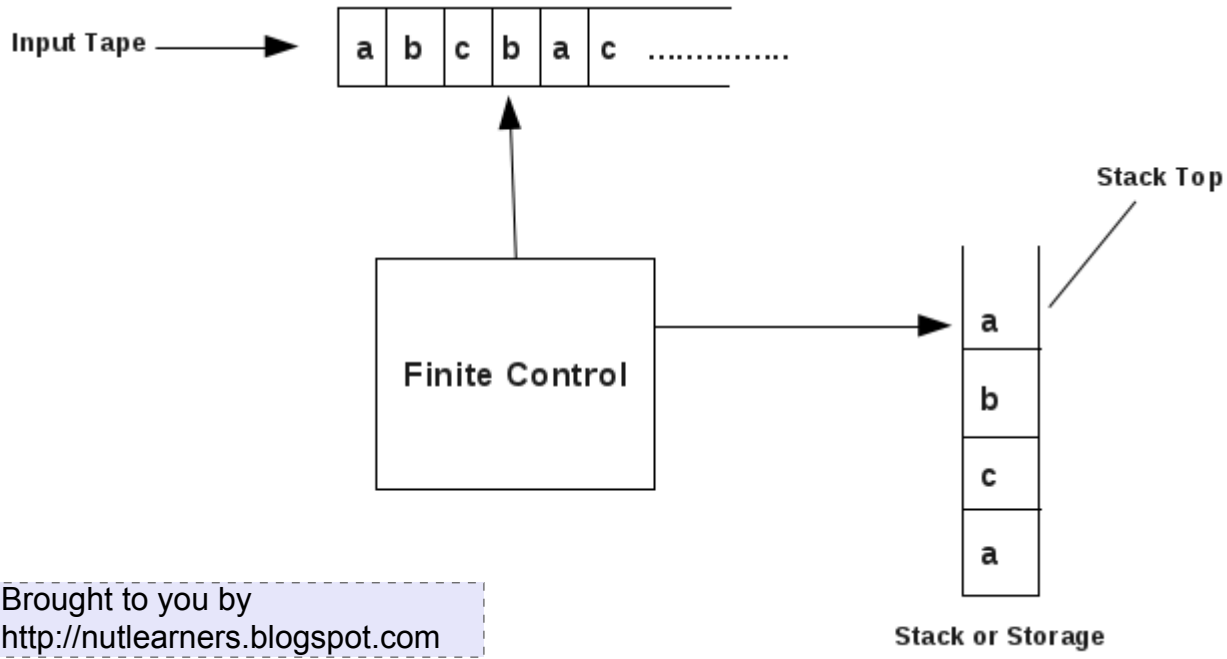
$$S \longrightarrow 01S1|11$$

Convert this grammar to Greibach normal form (GNF).

Part IV. Pushdown Automata (PDA)

As we learned, context free languages are generated using context free grammars. Context free languages are recognised using pushdown automata.

Following diagram shows a pushdown automation.



The PDA has three components:

An input tape,

A control mechanism, and

A stack.

From the input tape, the finite control reads the input, and also

the finite control reads a symbol from the stack top.

6 Definition of PDA

A pushdown automation, P is a system,

$$P = (Q, \sum, \Gamma, \delta, q_0, F)$$

where

Q is a set of states,

$\sum$ is a set of input symbols,

Γ is a set of stack symbols (pushdown symbols),

q_0 is the start state,

F is a set of final states,

δ is a transition function which maps,

$$(Q \times \sum^* \times \Gamma^*) \longrightarrow (Q \times \Gamma^*)$$

The meaning of δ is explained below:

Consider a transition function, δ in a PDA defined by,

$$\delta = (p, a, \beta) \longrightarrow (q, \gamma)$$

This means that

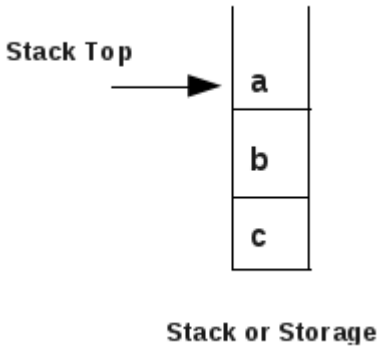
read a from the input tape,

pop the string β from stack top,

move from state p to state q,

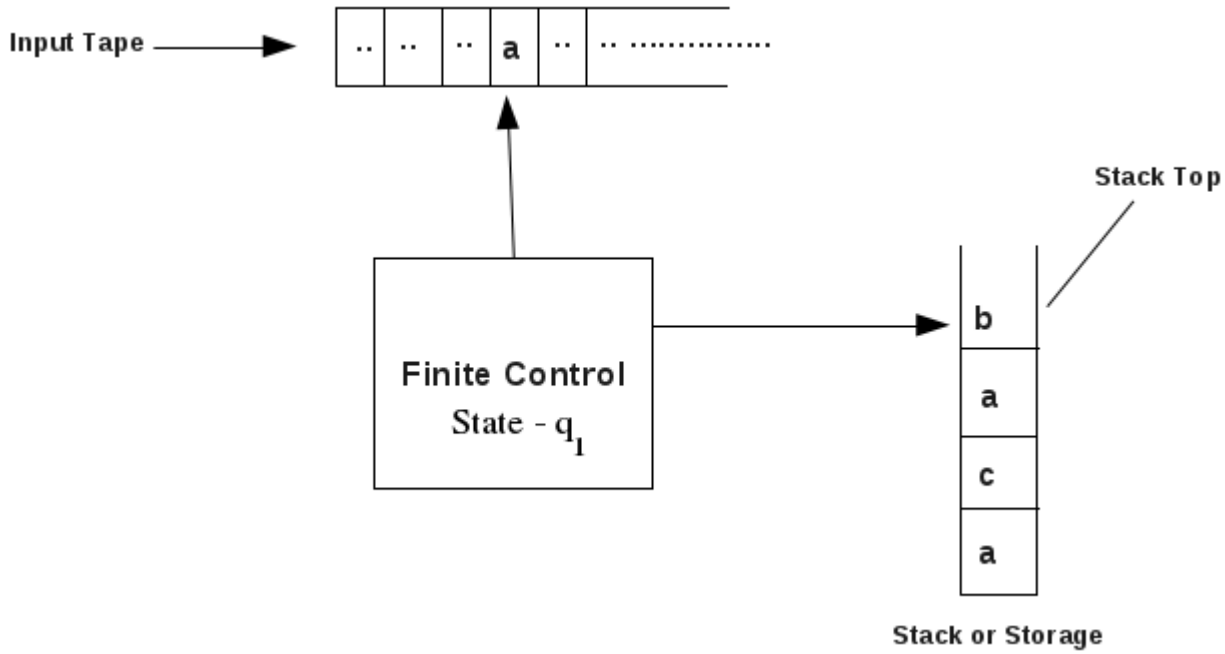
push string γ on to the stack top.

Pushing the string abc on to the stack means,



Example:

Consider the following PDA,



Suppose PDA is currently in state q_1 and the finite control points to state a and the stack contents are as shown above.

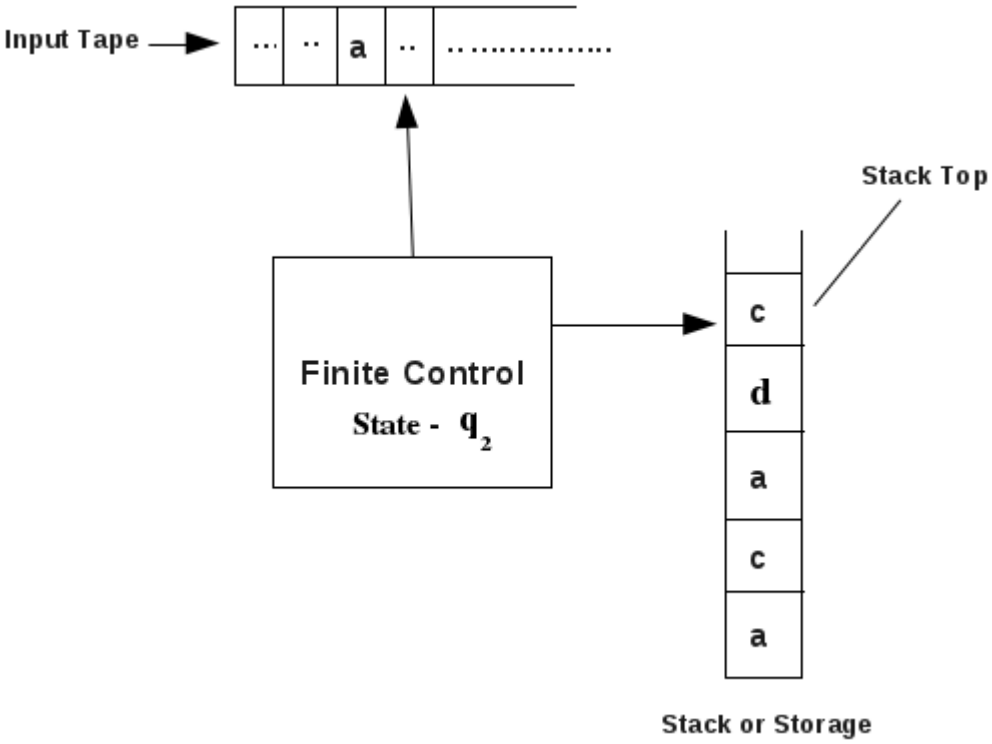
Let a transition is given as,

$$\delta = (q_1, a, b) \longrightarrow (q_2, cd)$$

This means PDA currently in state q_1, on reading the input symbol, a and popping b from the stack top changes to state

q_2 and pushes cd on to the stack top.

The new PDA is shown below:



7 Language Acceptability by PDA

Example 1:

Consider a PDA,

$$P = (Q, \sum, \Gamma, \delta, q_0, F)$$

where

$Q = \{s, q, f\}$

$\sum = \{a, b\}$

$\Gamma = \{a, b, c\}$

$q_0 = \{s\}$

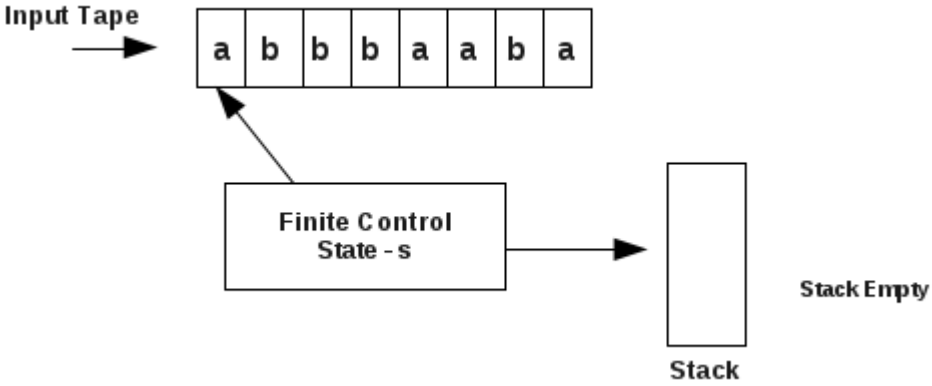
$F = \{f\}$

δ is given as follows:

1. $(s, a, \varepsilon) \longrightarrow (q, a)$
2. $(s, b, \varepsilon) \longrightarrow (q, b)$
3. $(q, a, a) \longrightarrow (q, aa)$
4. $(q, b, b) \longrightarrow (q, bb)$
5. $(q, a, b) \longrightarrow (q, \varepsilon)$
6. $(q, b, a) \longrightarrow (q, \varepsilon)$
7. $(q, b, \varepsilon) \longrightarrow (q, b)$
8. $(q, \varepsilon, \varepsilon) \longrightarrow (f, \varepsilon)$

Check whether the string abbbaaba is accepted by the above pushdown automation.

From the above, stack is initially empty, the start state of the PDA is s and PDA points to symbol, a as shown below:



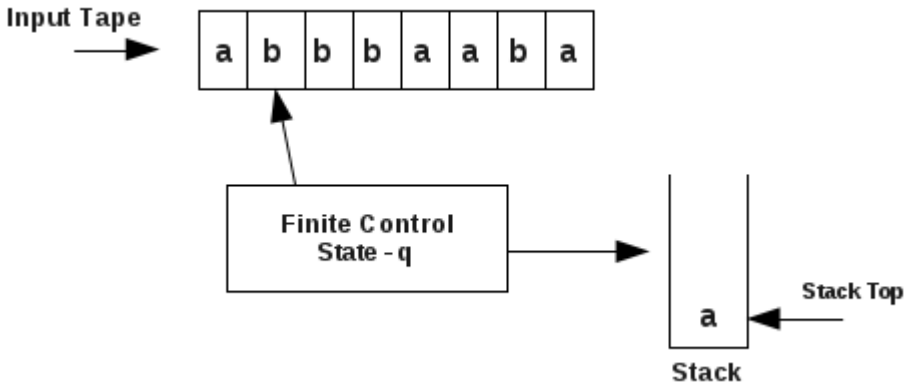
1.

The PDA is in state s, stack top contains symbol ε. Consider the transition,

1. $(s, a, \varepsilon) \longrightarrow (q, a)$

This means PDA in state s, reads a from the tape, pops nothing from stack top, moves to state q and pushes a onto the stack.

PDA now is,



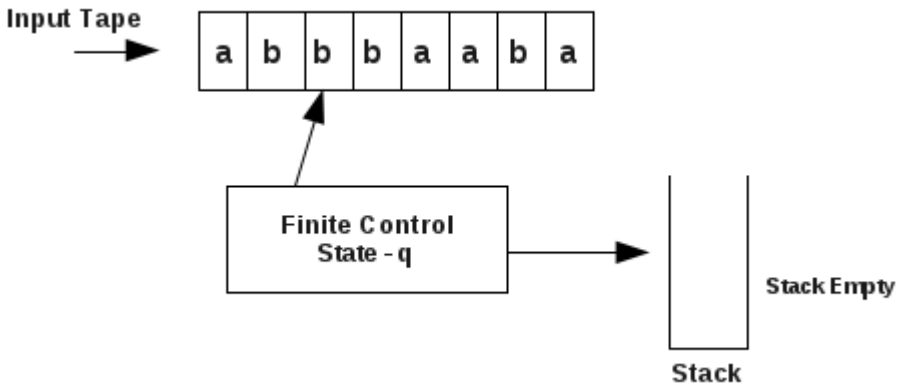
2.

The PDA is in state q, finite control points to symbol b in the input tape, stack top contains symbol a. Consider the transition,

6. $(q, b, a) \longrightarrow (q, \varepsilon)$

This means PDA is in state q, reads b from the input tape, pops a from stack top, moves to state q and pushes nothing onto the stack.

PDA now is,



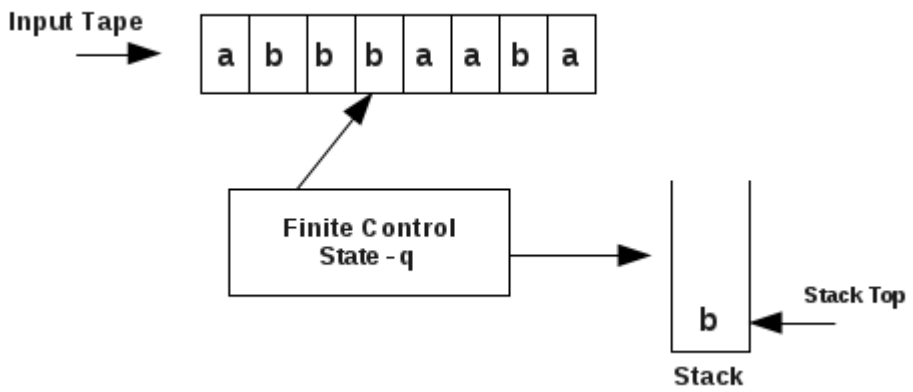
3.

The PDA is in state q, finite control points to symbol b in the input tape, stack top contains symbol ε. Consider the transition,

7. $(q, b, \varepsilon) \longrightarrow (q, b)$

This means PDA is in state q, reads b from the input tape, pops ε from stack top, moves to state q and pushes b onto the stack.

PDA now is,



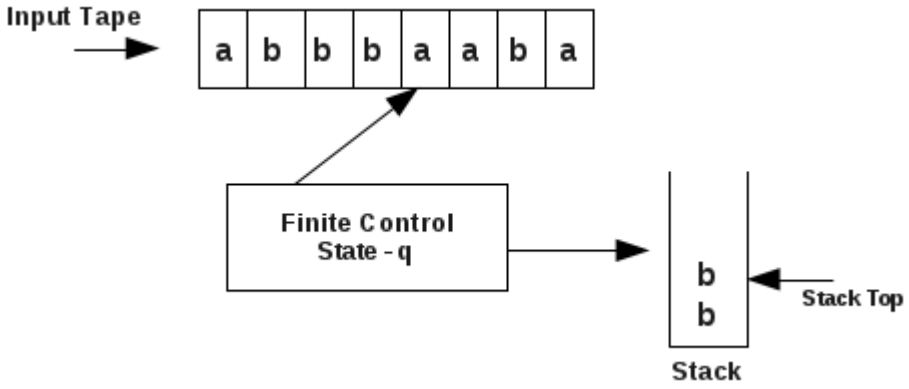
4.

The PDA is in state q, finite control points to symbol b in the input tape, stack top contains symbol b. Consider the transition,

4. $(q, b, b) \longrightarrow (q, bb)$

This means PDA is in state q, reads b from the input tape, pops b from stack top, moves to state q and pushes bb onto the stack.

PDA now is,



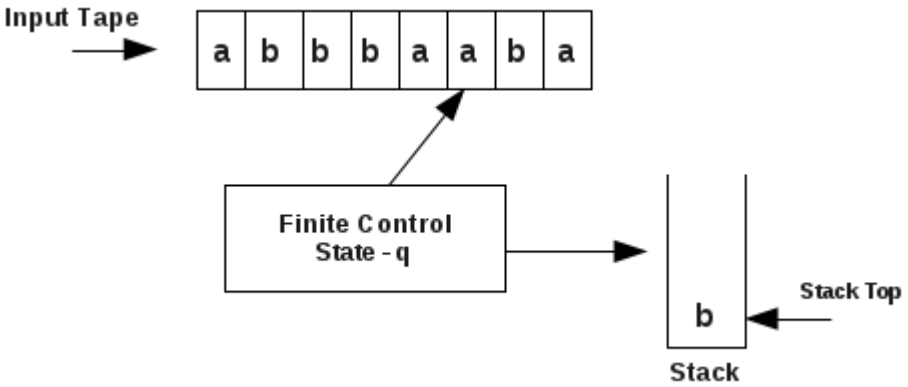
5.

The PDA is in state q, finite control points to symbol a in the input tape, stack top contains symbol b. Consider the transition,

5. $(q, a, b) \longrightarrow (q, \varepsilon)$

This means PDA is in state q, reads a from the input tape, pops b from stack top, moves to state q and pushes ε onto the stack.

PDA now is,



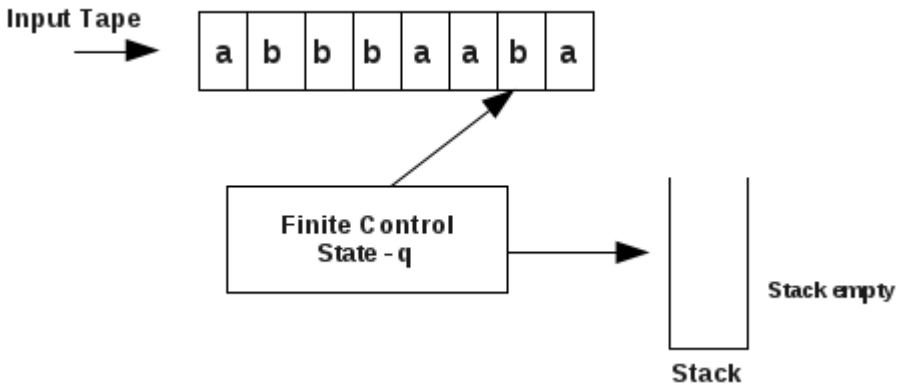
6.

The PDA is in state q, finite control points to symbol a in the input tape, stack top contains symbol b. Consider the transition,

5. $(q, a, b) \longrightarrow (q, \varepsilon)$

This means PDA is in state q, reads a from the input tape, pops b from stack top, moves to state q and pushes nothing onto the stack.

PDA now is,



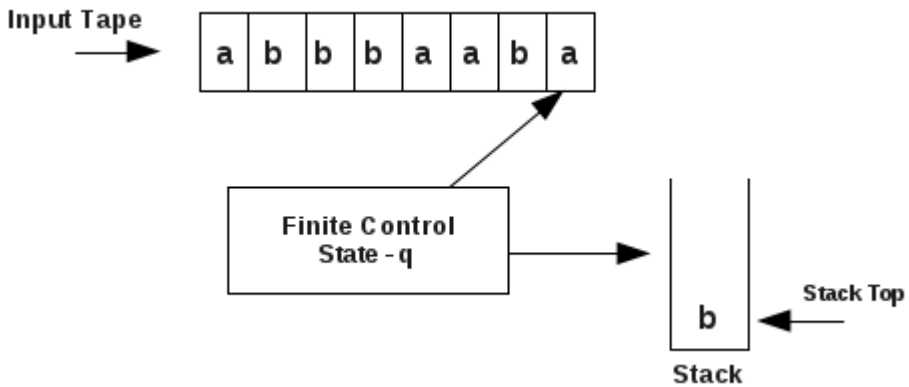
7.

The PDA is in state q, finite control points to symbol b in the input tape, stack top contains symbol ε. Consider the transition,

7. $(q, b, \varepsilon) \longrightarrow (q, b)$

This means PDA is in state q, reads b from the input tape, pops ε from stack top, moves to state q and pushes b onto the stack.

PDA now is,



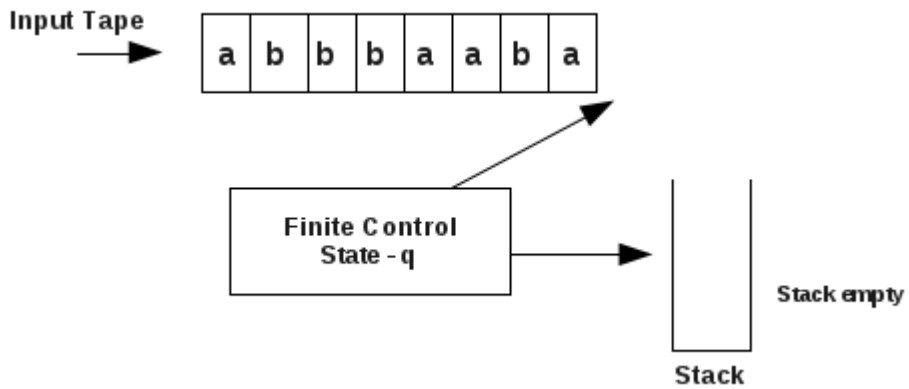
8.

The PDA is in state q, finite control points to symbol a in the input tape, stack top contains symbol b. Consider the transition,

5. $(q, a, b) \longrightarrow (q, \varepsilon)$

This means PDA is in state q, reads a from the input tape, pops b from stack top, moves to state q and pushes ε onto the stack.

PDA now is,



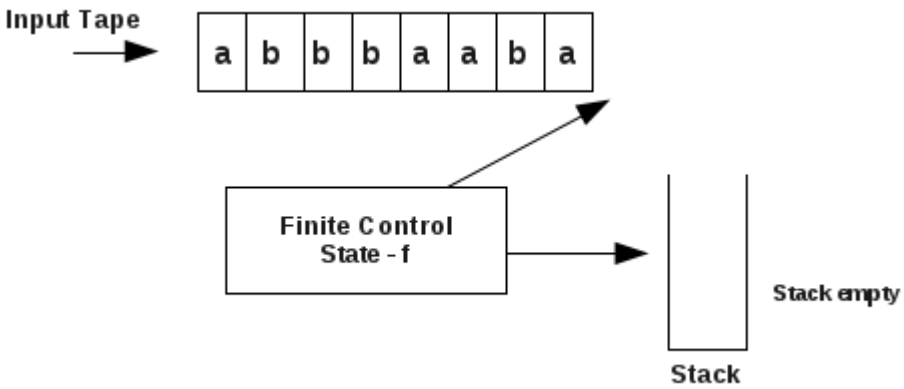
8.

The PDA is in state q, finite control points to symbol ε in the input tape, stack top contains symbol ε. Consider the transition,

8. $(q, \varepsilon, \varepsilon) \longrightarrow (f, \varepsilon)$

This means PDA is in state q, reads ε from the input tape, pops ε from stack top, moves to state f and pushes nothing onto the stack.

PDA now is,



Now PDA is in final state, f, and stack is empty. There are no more symbols in the input tape.

So the string $abbbaaba$ is accepted by the above pushdown automation.

Example 2:

1.

Consider a PDA,

$$P = (Q, \sum, \Gamma, \delta, q_0, F)$$

where

$Q = \{s, f\}$

$\sum = \{a, b, c\}$

$\Gamma = \{a, b\}$

$q_0 = \{s\}$

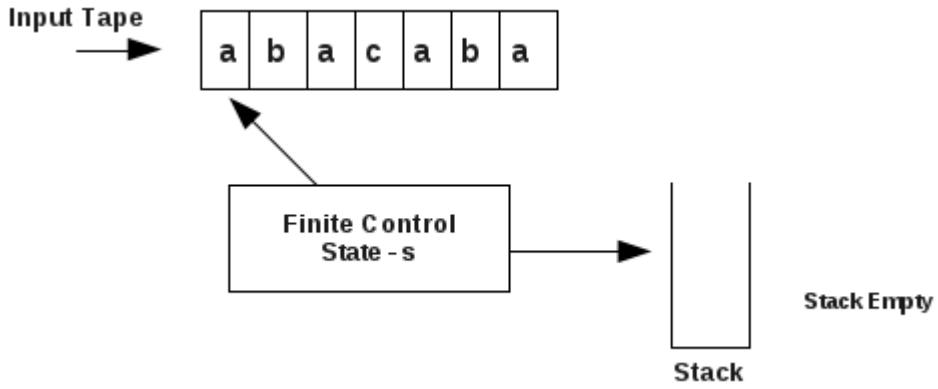
$F = \{f\}$

δ is given as follpws:

1. $(s, a, \varepsilon) \longrightarrow (s, a)$
2. $(s, b, \varepsilon) \longrightarrow (s, b)$
3. $(s, c, \varepsilon) \longrightarrow (f, \varepsilon)$
4. $(f, a, a) \longrightarrow (f, \varepsilon)$
5. $(f, b, b) \longrightarrow (f, \varepsilon)$

Check whether the string $abacaba$ is accpeted by the above pushdown automation.

From the above, stack initially is empty, the start state of the PDA is s as shown below:



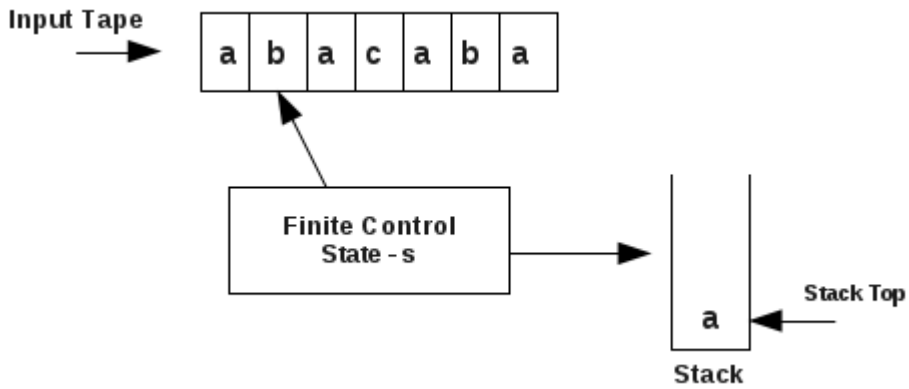
1.

The PDA is in state s, finite control points to symbol a in the input tape, stack top contains symbol ε. Consider the transition,

1. $(s, a, \varepsilon) \longrightarrow (s, a)$

This means PDA is in state s, reads a from the input tape, pops nothing from stack top, moves to state s and pushes a onto the stack.

PDA now is,



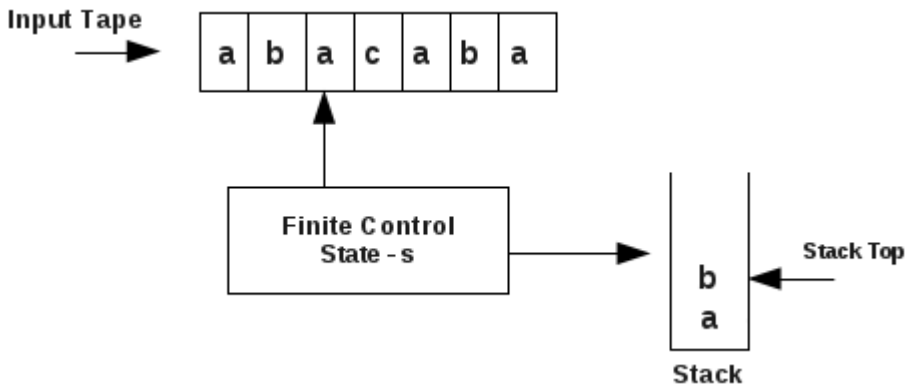
2.

The PDA is in state s, finite control points to symbol b in the input tape, stack top contains symbol a. Consider the transition,

2. $(s, b, \varepsilon) \longrightarrow (s, b)$

This means PDA is in state s, reads b from the input tape, pops nothing from stack top, moves to state s and pushes b onto the stack.

PDA now is,



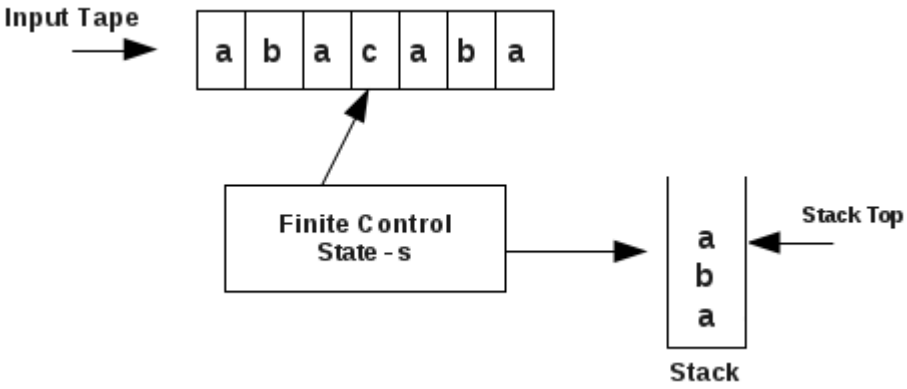
3.

The PDA is in state s, finite control points to symbol a in the input tape, stack top contains symbol b. Consider the transition,

1. $(s, a, \varepsilon) \longrightarrow (s, a)$

This means PDA is in state s, reads a from the input tape, pops nothing from stack top, moves to state s and pushes a onto the stack.

PDA now is,



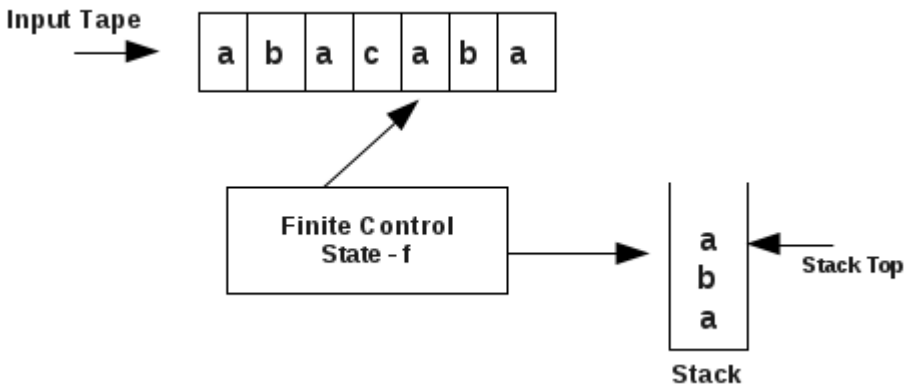
4.

The PDA is in state s, finite control points to symbol c in the input tape, stack top contains symbol a. Consider the transition,

3. $(s, c, \varepsilon) \longrightarrow (f, \varepsilon)$

This means PDA is in state s, reads c from the input tape, pops nothing from stack top, moves to state f and pushes nothing onto the stack.

PDA now is,



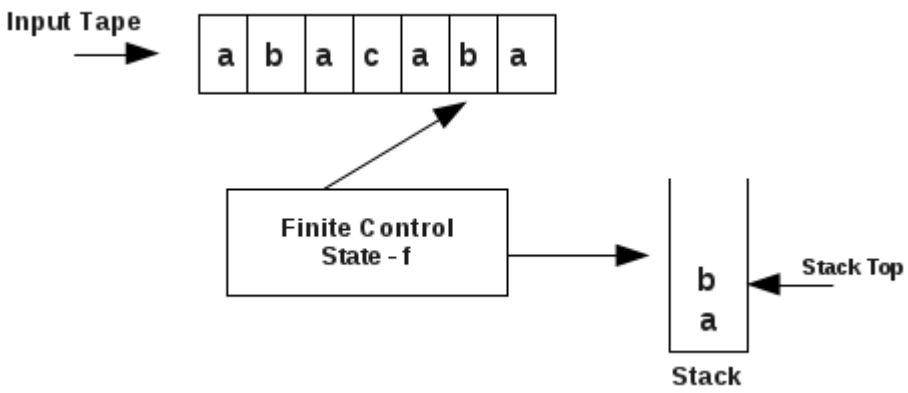
5.

The PDA is in state f, finite control points to symbol a in the input tape, stack top contains symbol a. Consider the transition,

4. $(f, a, a) \longrightarrow (f, \varepsilon)$

This means PDA is in state f, reads a from the input tape, pops a from stack top, moves to state f and pushes nothing onto the stack.

PDA now is,



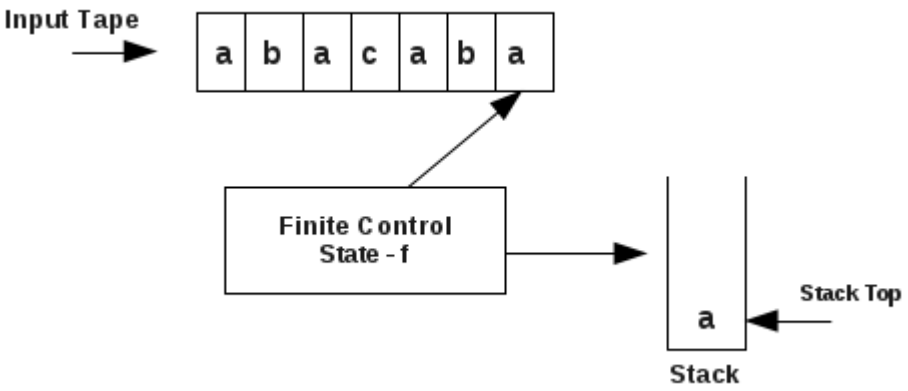
6.

The PDA is in state f, finite control points to symbol b in the input tape, stack top contains symbol b. Consider the transition,

5. $(f, b, b) \longrightarrow (f, \varepsilon)$

This means PDA is in state f, reads b from the input tape, pops b from stack top, moves to state f and pushes nothing onto the stack.

PDA now is,



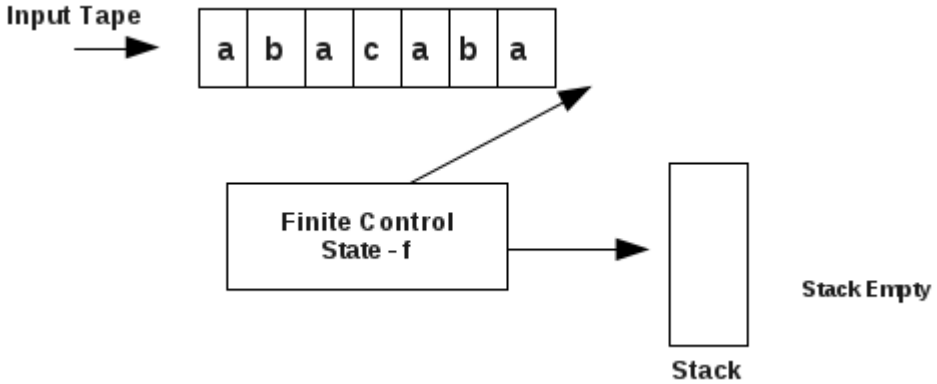
7.

The PDA is in state f, finite control points to symbol a in the input tape, stack top contains symbol a. Consider the transition,

4. $(f, a, a) \longrightarrow (f, \varepsilon)$

This means PDA is in state f, reads a from the input tape, pops a from stack top, moves to state f and pushes nothing onto the stack.

PDA now is,



Now there are no more symbols in the input string, stack is empty. PDA is in final state, f.

So the string $abacaba$ is accepted by the above pushdown automation.

Exercises:

1.

Consider a PDA,

$$P = (Q, \sum, \Gamma, \delta, q_0, F)$$

where

$Q = \{q_0, q_1, q_2, q_3\}$

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

$\Gamma = \{0, 1\}$

$q_0 = \{q_0\}$

$F = \{q_3\}$

δ is given as follpws:

1. $(q_0, 0, \varepsilon) \longrightarrow (q_1, 0)$
2. $(q_1, 0, 0) \longrightarrow (q_1, 00)$
3. $(q_1, 1, 0) \longrightarrow (q_2, \varepsilon)$
4. $(q_2, 1, 0) \longrightarrow (q_2, \varepsilon)$
5. $(q_2, \varepsilon, \varepsilon) \longrightarrow (q_3, \varepsilon)$

Check whether the string 000111 is accpeted by the above pushdown automation.

2.

Consider a PDA,

$$P = (Q, \sum, \Gamma, \delta, q_0, F)$$

where

$Q = \{q_0, q_1, q_2, q_3\}$

$\sum = \{a, b\}$

$\Gamma = \{a, b\}$

$q_0 = \{q_0\}$

$F = \{q_3\}$

δ is given as follpws:

1. $(q_0, a, \varepsilon) \longrightarrow (q_1, a)$
2. $(q_1, a, a) \longrightarrow (q_1, aa)$
3. $(q_1, b, a) \longrightarrow (q_2, a)$
4. $(q_2, a, a) \longrightarrow (q_2, \varepsilon)$
5. $(q_2, \varepsilon, \varepsilon) \longrightarrow (q_3, \varepsilon)$

Check whether the string $aabaa$ is accpeted by the above pushdown automation.

3.

Consider a PDA,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{q_0, q_1, q_2, q_3\}$

$\sum = \{a, b\}$

$\Gamma = \{a, b\}$

$q_0 = \{q_0\}$

$F = \{q_3\}$

δ is given as follpws:

1. $(q_0, a, \varepsilon) \longrightarrow (q_1, a)$
2. $(q_1, a, a) \longrightarrow (q_1, aa)$
3. $(q_1, b, a) \longrightarrow (q_2, a)$
4. $(q_2, b, a) \longrightarrow (q_3, \varepsilon)$
5. $(q_3, b, a) \longrightarrow (q_2, a)$
5. $(q_3, \varepsilon, \varepsilon) \longrightarrow (q_4, \varepsilon)$

Check whether the string $aabbbb$ is accpeted by the above pushdown automation.

2.

Consider a PDA,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{s, q, f\}$

$\sum = \{a, b\}$

$\Gamma = \{a, b\}$

$q_0 = \{s\}$

$F = \{f\}$

δ is given as follpws:

1. $(s, a, \varepsilon) \longrightarrow (s, a)$

2. $(s, b, \varepsilon) \longrightarrow (s, b)$
3. $(s, \varepsilon, \varepsilon) \longrightarrow (q, \varepsilon)$
4. $(q, a, a) \longrightarrow (q, \varepsilon)$
4. $(q, b, b) \longrightarrow (q, \varepsilon)$
5. $(q, \varepsilon, \varepsilon) \longrightarrow (f, \varepsilon)$

Check whether the string $aabbaa$ is accepted by the above pushdown automation.

8 Deterministic Pushdown Automata (DPDA)

A PDA is said to be deterministic, if

1. $\delta(q, a, b)$ contains at most one element, and
2. if $\delta(q, \varepsilon, b)$ is not empty, then

   $\delta(q, c, b)$ must be empty for every input symbol, c.

For example, consider the following PDA,

Example 1:

Consider the PDA,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{s, f\}$

$\sum = \{a, b, c\}$

$\Gamma = \{a, b\}$

$q_0 = \{s\}$

$F = \{f\}$

δ is given as follpws:

1. $(s, a, \varepsilon) \longrightarrow (s, a)$
2. $(s, b, \varepsilon) \longrightarrow (s, b)$
3. $(s, c, \varepsilon) \longrightarrow (f, \varepsilon)$
4. $(f, a, a) \longrightarrow (f, \varepsilon)$
5. $(f, b, b) \longrightarrow (f, \varepsilon)$

Above is a deterministic pushdown automata (DPDA) because it satisfies both conditions of DPDA.

Example 2:

Consider the PDA,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{s, f\}$

$\sum = \{a, b, c\}$

$\Gamma = \{a, b\}$

$q_0 = \{s\}$

$F = \{f\}$

δ is given as follpws:

1. $(s, a, \varepsilon) \longrightarrow (s, a)$
2. $(s, b, \varepsilon) \longrightarrow (s, b)$
3. $(s, c, \varepsilon) \longrightarrow (f, \varepsilon)$
4. $(f, a, a) \longrightarrow (f, \varepsilon)$
5. $(f, b, b) \longrightarrow (f, \varepsilon)$

Above is a deterministic pushdown automata (DPDA) because it satisfies both conditions of DPDA.

9 Non-Deterministic Pushdown Automata (NPDA)

A PDA is said to be non- deterministic, if

1. $\delta(q, a, b)$ may contain multiple elements, or
2. if $\delta(q, \varepsilon, b)$ is not empty, then

   $\delta(q, c, b)$ is not empty for some input symbol, c.

Example 1:

Consider the following PDA,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{q_0, q_1, q_2, q_3\}$

$\sum = \{a, b\}$

$\Gamma = \{0, 1\}$

$q_0 = \{q_0\}$

$F = \{q_3\}$

δ is given as follpws:

1. $(q_0, a, 0) \longrightarrow (q_1, 10),\ (q_3, \varepsilon)$
2. $(q_0, \varepsilon, 0) \longrightarrow (q_3, \varepsilon)$
3. $(q_1, a, 1) \longrightarrow (q_1, 11)$
4. $(q_1, b, 1) \longrightarrow (q_2, \varepsilon)$
5. $(q_2, b, 1) \longrightarrow (q_2, \varepsilon)$
5. $(q_2, \varepsilon, 0) \longrightarrow (q_3, \varepsilon)$

Above is a non deterministic pushdown automata (NPDA).

Consider the transition 1, ie.

1. $(q_0, a, 0) \longrightarrow (q_1, 10),\ (q_3, \varepsilon)$

Here $(q_0, a, 0)$ can go to q_1 or q_3. That this transition is not deterministic.

Example 2:

Consider the following PDA,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{q_0, q_f\}$

$\sum = \{a, b\}$

$\Gamma = \{0, 1\}$

$q_0 = \{q_0\}$

$F = \{q_f\}$

δ is given as follpws:

1. $(q_0, \varepsilon, \varepsilon) \longrightarrow (q_f, \varepsilon)$
2. $(q_0, a, \varepsilon) \longrightarrow (q_0, 0)$
3. $(q_0, b, \varepsilon) \longrightarrow (q_0, 1)$
4. $(q_0, a, 0) \longrightarrow (q_0, 00)$
5. $(q_0, b, 0) \longrightarrow (q_0, \varepsilon)$
6. $(q_0, a, 1) \longrightarrow (q_0, \varepsilon)$
7. $(q_0, b, 1) \longrightarrow (q_0, 11)$

Above is a non deterministic pushdown automata (NPDA).

Consider the transitions, 1, 2 and 3.

1. $(q_0, \varepsilon, \varepsilon) \longrightarrow (q_f, \varepsilon)$
2. $(q_0, a, \varepsilon) \longrightarrow (q_0, 0)$
3. $(q_0, b, \varepsilon) \longrightarrow (q_0, 1)$

Here $(q_0, \varepsilon, \varepsilon)$ is not empty, also,

(q_0, a, ε) and (q_0, b, ε) are not empty.

Example 3:

Consider the following PDA,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{s, p, q\}$

$\sum = \{a, b\}$

$\Gamma = \{a, b\}$

$q_0 = \{s\}$

$F = \{q\}$

δ is given as follpws:

1. $(s, a, \varepsilon) \longrightarrow (p, a)$
2. $(p, a, a) \longrightarrow (p, aa)$
3. $(p, a, \varepsilon) \longrightarrow (p, a)$
4. $(p, b, a) \longrightarrow (p, \varepsilon),\ (q, \varepsilon)$

Above is a non deterministic pushdown automata (NPDA).

This is due to the transition,

4. $(p, b, a) \longrightarrow (p, \varepsilon),\ (q, \varepsilon)$

Example 4:

Consider the following PDA,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{q_0, q_1, q_2, q_3, f\}$

$\sum = \{a, b\}$

$\Gamma = \{a, b\}$

$q_0 = \{q_0\}$

$F = \{f\}$

δ is given as follpws:

1. $(q_0, a, \varepsilon) \longrightarrow (q_1, a)$
2. $(q_1, a, a) \longrightarrow (q_1, a)$
3. $(q_1, b, a) \longrightarrow (q_2, \varepsilon)$
4. $(q_2, b, a) \longrightarrow (q_3, \varepsilon)$
5. $(q_3, b, a) \longrightarrow (q_2, \varepsilon)$
6. $(q_2, \varepsilon, a) \longrightarrow (q_2, \varepsilon)$
7. $(q_2, \varepsilon, \varepsilon) \longrightarrow (f, \varepsilon)$
8. $(q_1, \varepsilon, a) \longrightarrow (q_1, \varepsilon)$
9. $(q_1, \varepsilon, \varepsilon) \longrightarrow (f, \varepsilon)$

Above is a non deterministic pushdown automata (NPDA).

This is due to the transitions, 6 and 4, ie,

6. $(q_2, \varepsilon, a) \longrightarrow (q_2, \varepsilon)$
4. $(q_2, b, a) \longrightarrow (q_3, \varepsilon)$

Also due to the transitions, 8 and 2 and 3, ie,

8. $(q_1, \varepsilon, a) \longrightarrow (q_1, \varepsilon)$
2. $(q_1, a, a) \longrightarrow (q_1, a)$

3. $(q_1, b, a) \longrightarrow (q_2, \varepsilon)$

10 Design of Pushdown Automata

For every CFG, there exists a pushdown automation that accepts it.

To design a pushdown automation corresponding to a CFG, following are the steps:

Step 1:

Let the start symbol of the CFG is S. Then a transition of PDA is,

$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$

Step 2:

For a production of the form, $P \longrightarrow AaB$, a transition of PDA is,

$\delta(q, \varepsilon, P) \longrightarrow (q, AaB)$

For a production of the form, $P \longrightarrow a$, a transition of PDA is,

$\delta(q, \varepsilon, P) \longrightarrow (q, a)$

For a production of the form, $P \longrightarrow \varepsilon$, a transition of PDA is,

$\delta(q, \varepsilon, P) \longrightarrow (q, \varepsilon)$

Step 3:

For every terminal symbol, a in CFG, a transition of PDA is,

$\delta(q, a, a) \longrightarrow (q, \varepsilon)$

Thus the PDA is,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{p, q\}$

$\sum =$ set of terminal symbols in the CFG

$\Gamma =$set of terminals and non-terminals in the CFG

$q_0 = p$

$F = q$

δ is according to the above rules.

Example 1:

Consider the following CFG,

$S \longrightarrow aA$

$A \longrightarrow aABC|bB|a$

$B \longrightarrow b$

$C \longrightarrow c$

where S is the start symbol.

Design a pushdown automata corresponding to the above grammar.

Step 1:

Here start symbol of CFG is S. A transition is,

$$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$$

Step 3:

Consider the production, $S \longrightarrow aA$.

A transition is

$$\delta(q, \varepsilon, S) \longrightarrow (q, aA)$$

Consider the production, $A \longrightarrow aABC|bB|a$.

Transitions are

$$\delta(q, \varepsilon, A) \longrightarrow (q, aABC),$$
$$\delta(q, \varepsilon, A) \longrightarrow (q, bB),$$
$$\delta(q, \varepsilon, A) \longrightarrow (q, a).$$

Consider the production, $B \longrightarrow b$.

Corresponding transition is

$$\delta(q, \varepsilon, B) \longrightarrow (q, b)$$

Consider the production, $C \longrightarrow c$

Corresponding transition is

$$\delta(q, \varepsilon, C) \longrightarrow (q, c)$$

Step 3:

The terminal symbols in the CFG are, a, b, c.

Then the transitions are,

$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$
$$\delta(q, b, b) \longrightarrow (q, \varepsilon)$$
$$\delta(q, c, c) \longrightarrow (q, \varepsilon)$$

Thus the PDA is,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{p, q\}$

$\sum = \{a, b, c\}$

$\Gamma = \{S, A, B, C, a, b, c\}$

$q_0 = p$

$F = q$

δ is given as follows:

$$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, aA)$$
$$\delta(q, \varepsilon, A) \longrightarrow (q, aABC),$$
$$\delta(q, \varepsilon, A) \longrightarrow (q, bB),$$

$$\delta(q, \varepsilon, A) \longrightarrow (q, a).$$
$$\delta(q, \varepsilon, B) \longrightarrow (q, b)$$
$$\delta(q, \varepsilon, C) \longrightarrow (q, c)$$
$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$
$$\delta(q, b, b) \longrightarrow (q, \varepsilon)$$
$$\delta(q, c, c) \longrightarrow (q, \varepsilon)$$

Example 2:

Consider the CFG,

$S \longrightarrow AB|BC$

$A \longrightarrow aB|bA|a$

$B \longrightarrow bB|cC|b$

$C \longrightarrow c$

where S is the start symbol.

Step 1:

Here start symbol of CFG is S. A transition is,

$$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$$

Step 3:

Consider the production, $S \longrightarrow AB|BC$

Transitions are

$$\delta(q, \varepsilon, S) \longrightarrow (q, AB)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, BC$$

Consider the production, $A \longrightarrow aB|bA|a$

Transitions are

$$\delta(q, \varepsilon, A) \longrightarrow (q, aB),$$
$$\delta(q, \varepsilon, A) \longrightarrow (q, bA),$$
$$\delta(q, \varepsilon, A) \longrightarrow (q, a).$$

Consider the production, $B \longrightarrow bB|cC|b$

Transitions are

$$\delta(q, \varepsilon, B) \longrightarrow (q, bB)$$
$$\delta(q, \varepsilon, B) \longrightarrow (q, cC),$$
$$\delta(q, \varepsilon, B) \longrightarrow (q, b)$$

Consider the production, $C \longrightarrow c$

Transitions are

$$\delta(q, \varepsilon, C) \longrightarrow (q, c)$$

Step 3:

The terminal symbols in the CFG are, a, b, c.

Then the transitions are,

$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$
$$\delta(q, b, b) \longrightarrow (q, \varepsilon)$$
$$\delta(q, c, c) \longrightarrow (q, \varepsilon)$$

Thus the PDA is,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{p, q\}$

$\sum = \{a, b, c\}$

$\Gamma = \{S, A, B, C, a, b, c\}$

$q_0 = p$

$F = q$

δ is given as follows:

$$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, AB)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, BC)$$
$$\delta(q, \varepsilon, A) \longrightarrow (q, aB)$$
$$\delta(q, \varepsilon, A) \longrightarrow (q, bA)$$
$$\delta(q, \varepsilon, A) \longrightarrow (q, a)$$
$$\delta(q, \varepsilon, B) \longrightarrow (q, bB)$$
$$\delta(q, \varepsilon, B) \longrightarrow (q, cC)$$
$$\delta(q, \varepsilon, B) \longrightarrow (q, b)$$
$$\delta(q, \varepsilon, C) \longrightarrow (q, c)$$
$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$
$$\delta(q, b, b) \longrightarrow (q, \varepsilon)$$
$$\delta(q, c, c) \longrightarrow (q, \varepsilon)$$

Example 3:

Design a pushdown automata that accepts the language, $L = \{wcw^R | w \in (a, b)^*\}$

We learned earlier that the CFG corresponding to this language is,

$S \longrightarrow aSa$

$S \longrightarrow bSb$

$S \longrightarrow c$

where S is the start symbol.

Next we need to find the PDA corresponding to the above CFG.

Step 1:

Here start symbol of CFG is S. A transition is,

$$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$$

Step 3:

Consider the production, $S \longrightarrow aSa$

Transitions are

$$\delta(q, \varepsilon, S) \longrightarrow (q, aSa)$$

Consider the production, $S \longrightarrow bSb$

Transitions are

$$\delta(q, \varepsilon, S) \longrightarrow (q, bSb),$$

Consider the production, $S \longrightarrow c$

Transitions are

$$\delta(q, \varepsilon, S) \longrightarrow (q, c)$$

Step 3:

The terminal symbols in the CFG are, a, b, c.

Then the transitions are,

$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$
$$\delta(q, b, b) \longrightarrow (q, \varepsilon)$$
$$\delta(q, c, c) \longrightarrow (q, \varepsilon)$$

Thus the PDA is,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{p, q\}$

$\sum = \{a, b, c\}$

$\Gamma = \{S, a, b, c\}$

$q_0 = p$

$F = q$

δ is given as follows:

$$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, aSa)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, bSb)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, c)$$
$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$
$$\delta(q, b, b) \longrightarrow (q, \varepsilon)$$
$$\delta(q, c, c) \longrightarrow (q, \varepsilon)$$

Example 4:

Design a pushdown automata that accepts the language, $L = \{w|w$ contains equal number of a's and b's } from the input alphabet {a,b}.

First, we need to find the CFG corresponding to this language. We learned in a previous section, the CFG corresponding to this is,

$S \longrightarrow aSbS|bSaS|\varepsilon$

where S is the start symbol.

Next we need to find the PDA corresponding to the above CFG.

Step 1:

Here start symbol of CFG is S. A transition is,

$$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$$

Step 3:

Consider the production, $S \longrightarrow aSbS|bSaS|\varepsilon$

Transitions are

$$\delta(q, \varepsilon, S) \longrightarrow (q, aSbS)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, bSaS)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, \varepsilon)$$

Step 3:

The terminal symbols in the CFG are, a, b.

Then the transitions are,

$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$
$$\delta(q, b, b) \longrightarrow (q, \varepsilon)$$

Thus the PDA is,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{p, q\}$

$\sum = \{a, b\}$

$\Gamma = \{S, a, b\}$

$q_0 = p$

$F = q$

δ is given as follows:

$$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, aSbS)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, bSaS)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, \varepsilon)$$
$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$

$$\delta(q, b, b) \longrightarrow (q, \varepsilon)$$

Example 5:

Design a pushdown automata that accepts the language corresponding to the regular expression, $(a|b)^*$.

First, we need to find the CFG corresponding to this language. We learned in a previous section, the CFG corresponding to this is,

$S \longrightarrow aS|bS|a|b|\varepsilon$

where S is the start symbol.

Next we need to find the PDA corresponding to the above CFG.

Step 1:

Here start symbol of CFG is S. A transition is,

$$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$$

Step 3:

Consider the production, $S \longrightarrow aS|bS|a|b|\varepsilon$

Transitions are

$$\delta(q, \varepsilon, S) \longrightarrow (q, aS)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, bS)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, a)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, b)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, \varepsilon)$$

Step 3:

The terminal symbols in the CFG are, a, b.

Then the transitions are,

$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$
$$\delta(q, b, b) \longrightarrow (q, \varepsilon)$$

Thus the PDA is,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{p, q\}$

$\sum = \{a, b\}$

$\Gamma = \{S, a, b\}$

$q_0 = p$

$F = q$

δ is given as follows:

$$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$$

$$\delta(q, \varepsilon, S) \longrightarrow (q, aS)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, bS)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, a)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, b)$$
$$\delta(q, \varepsilon, S) \longrightarrow (q, \varepsilon)$$
$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$
$$\delta(q, b, b) \longrightarrow (q, \varepsilon)$$

Exercises:

1. Design a pushdown automation to accept the language, $L = \{ww^R | w \in \{a, b\}^*\}$.
2. Design a pushdown automation to accept the language, $L = \{a^n b^n | n > 0\}$.
3. Design a pushdown automation to accept the language, $L = \{a^n b^{2n} | n > 0\}$.
4. Design a pushdown automation for the regular expression, $r = 0^*1^*$.
5. Design a pushdown automation to accept the language, $L = \{a^n b^{n+1} | n = 1, 2, 3, ...\}$.
6. Design a pushdown automation to accept the language, $L = \{a^n b^m | n > m \geq 0\}$.
7. Construct PDA equivalent to the grammar $S \longrightarrow aAA,\ A \longrightarrow aS/bS/a$.
8. Construct PDA for the language $L = \{x \in (a, b)^* / n_a(x) > n_b(n)\}$.
9. Construct a PDA that accepts the language $L = \{a^n b a^m / n,\ m \geq 1\}$ by empty stack.

11 Applications of Pushdown Automata

An application of PDA is in parsing.

Parsing is the process of checking whether the syntax of a program is correct or not. For example, a PDA can be constructed to check the syntax of all C programs.

Also, parsing is the process of checking whether a sentence written in a natural language is correct or not.

Parsing is of two types,

Top down parsing, and

Bottom up parsing.

Top Down Parsing

An example for top down parsing is given below:

Example 1:

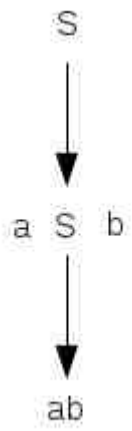
Consider the following CFG,

$S \longrightarrow aSb \,|\, ab$

where S is the start symbol.

Check whether the syntax of string $aabb$ is according to the above CFG.

Top down parsing is shown below:



Thus top down parsing is done beginning from the start symbol, by following a set of productions, reach the given string. If the string cannot be reached, its syntax is not correct.

Above diagram is called a derivation tree (parse tree).

Here, by performing top down parsing , we got a^2b^2. So a^2b^2 belongs to the above CFL.

PDA for Top down Parsing

A top down parser can be implemented by using a Pushdown automation. Using the pushdown automation, we can parse a given string.

Example 1:

For example, the PDA corresponding to the CFG, $S \longrightarrow aSb \,|\, ab$

is,

$$P = (Q, \sum, \Gamma, \delta, q_0, F)$$

where

$Q = \{p, q\}$

$\sum = \{a, b\}$

$\Gamma = \{S, a, b\}$

$q_0 = p$

$F = q$

δ is given as follows:

$$\delta(p, \varepsilon, \varepsilon) \longrightarrow (q, S)$$

$$\delta(q, \varepsilon, S) \longrightarrow (q, aSb)$$

$$\delta(q, \varepsilon, S) \longrightarrow (q, ab)$$

$$\delta(q, a, a) \longrightarrow (q, \varepsilon)$$

$$\delta(q, b, b) \longrightarrow (q, \varepsilon)$$

This PDA acts as a top down parser for the CFL corresponding to above CFG.

Bottom Up Parsing

This approach can also be used to check whether a string belongs to a context free language. Here, the string w is taken and an inverted tree is made. This is done by performing a number of reductions starting from the given string using the productions of the grammar.

Example 1:

Consider the grammar,

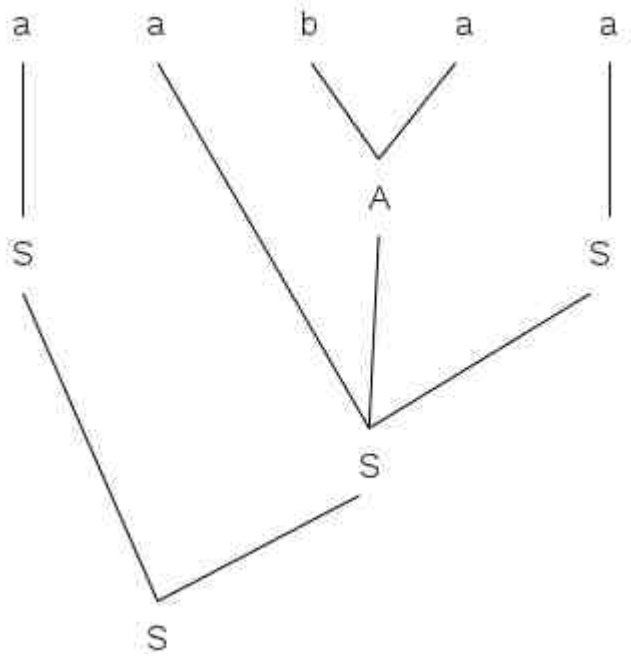
$S \longrightarrow aAS|a|SS$

$A \longrightarrow SbA|ba$

where S is the start symbol.

Check whether the string aabaa belongs to the CFL associated with the above grammar.

Bottom up parsing is shown below:



Above is a bottom up parse tree.

Thus by performing a number of reductions, we finally reached the start symbol. So the string aabaa belongs to the above CFL.

This parsing process can be done using a PDA.

PDA for Bottom down Parsing

A top down parser can be implemented by using a Pushdown automation. Using this pushdown automation, we can parse a given string.

Example 1:

Consider the grammar,

$S \longrightarrow aAS|a|SS$

$A \longrightarrow SbA|ba$

where S is the start symbol.

A pushdown automation for performing bottom up parsing is shown below:

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{p, q\}$

$\sum = \{a, b\}$

$\Gamma = \{S, A, a, b\}$

$q_0 = p$

$F = q$

δ is given as follows:

$$\delta(q, \varepsilon, S) \longrightarrow (p, \varepsilon)$$

$$\delta(q, \varepsilon, SAa) \longrightarrow (q, S)$$

$$\delta(q, \varepsilon, a) \longrightarrow (q, S)$$

$$\delta(q, \varepsilon, SS) \longrightarrow (q, S)$$

$$\delta(q, \varepsilon, AbS) \longrightarrow (q, A)$$

$$\delta(q, \varepsilon, ab) \longrightarrow (q, A)$$

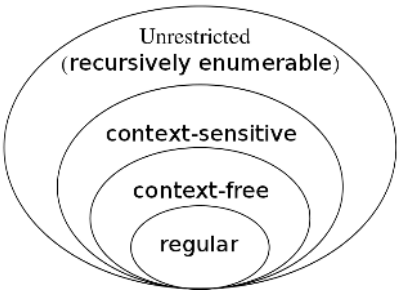
$$\delta(q, a, \varepsilon) \longrightarrow (q, a)$$

$$\delta(q, b, \varepsilon) \longrightarrow (q, b)$$

This PDA acts as a bottom up parser for the CFL corresponding to above CFG.

Part V. Pumping Lemma for Context Free Languages (CFLs)

Consider the following figure:



From the diagram, we can say that not all languages are context free languages. All context free languages are context sensitive and unrestricted. But all context sensitive and unrestricted languages are not context free from the above diagram.

We have a mechanism to show that some languages are not context free. The pumping lemma for CFLs allow us to show that some languages are not context free.

12 Pumping lemma for CFLs

Let G be a CFG. Then there exists a constant, n such that if W is in $L(G)$ and $|W| \geq n$, then we may write $W = uvxyz$ such that,

1. $|vy| \geq 1$ that is either v or y is non-empty,
2. $|vxy| \leq n$ then for all $i \geq 0$,

   uv^ixy^iz is in $L(G)$.

This is known as pumping lemma for context free languages.

Pumping lemma for CFLs can be used to prove that a language, L is not context free.

We assume L is context free. Then we apply pumping lemma to get a contradiction.

Following are the steps:

Step 1:

Assume L is context free. Let n be the natural number obtained by using the pumping lemma.

Step 2:

Choose $W \in L$ so that $|W| \geq n$. Write $W = uvxyz$ using the pumping lemma.

Step 3:

Find a suitable k so that $uv^kxy^kz \notin L$. This is a contradiction, and so L is not context free.

Examples:

Example 1:

Show that $L = \{a^nb^nc^n | n \geq 1\}$ is not context free.

Step 1:

Assume L is context free. Let n be the natural number obtained by using the pumping lemma.

Step 2:

Choose $W \in L$ so that $|W| \geq n$. Write $W = uvxyz$ using the pumping lemma.

Let $W = a^n b^n c^n$. Then $|W| = 3n > n$.

Write $W = uvxyz$, where $|vy| \geq 1$, that is, at least one of v or x is not ε.

Step 3:

Find a suitable k so that $uv^k xy^k z \notin L$. This is a contradiction, and so L is not context free.

$uvxyz = a^n b^n c^n$. As $1 \leq |vy| \leq n$, v or x cannot contain all the three symbols a, b, c.

i) So v or y is of the form $a^i b^j$ (or $b^j c^j$) for some i,j such that $i + j \leq n$. or

ii) v or y is a string formed by the repetition of only one symbol among a, b, c.

When v or y is of the form $a^i b^j$, $v^2 = a^i b^j a^i b^j$ (or $y^2 = a^i b^j a^i b^j$). As v^2 is a substring of the form uv^2xy^2z, we cannot have uv^2xy^2z of the form $a^m b^m c^m$. So $uv^2xy^2z \notin L$.

When both v and y are formed by the repetition of a single symbol (example: $u = a^i$ and $v = b^j$ for some i and j, $i \leq n$, $j \leq n$), the string uxz will contain the remaining symbol, say a_1. Also a_1^n will be a substring of uxz as a_1 does not occur in v or y. The number of occurrences of one of the other two symbols in uxz is less than n (recall $uvxyz = a^n b^n c^n$), and n is the number of occurrences of a_1. So $uv^0xy^0z = uxz \notin L$.

Thus for any choice of v or y , we get a contradiction. Therefore, L is not context free.

Example 2:

Show that $L = \{a^p | p$ is a prime number$\}$ is not a context free language.

This means, if $w \in L$, number of characters in w is a prime number.

Step 1:

Assume L is context free. Let n be the natural number obtained by using the pumping lemma.

Step 2:

Choose $W \in L$ so that $|W| \geq n$. Write $W = uvxyz$ using the pumping lemma.

Let p be a prime number greater than n. Then $w = a^p \in L$.

We write $W = uvxyz$

Step 3:

Find a suitable k so that $uv^k xy^k z \notin L$. This is a contradiction, and so L is not context free.

By pumping lemma, $uv^0xy^0z = uxz \in L$.

So $|uxz|$ is a prime number, say q.

Let $|vy| = r$.

Then $|uv^q xy^q z| = q + qr$.

Since, $q + qr$ is not prime, $uv^q xy^q z \notin L$. This is a contradiction. Therefore, L is not context free.

Example 3:

Show that the language $L = \{a^n b^n c^n | n \geq 0\}$ is not context free.

Step 1:

Assume L is context free. Let n be the natural number obtained by using the pumping lemma.

Step 2:

Choose $W \in L$ so that $|W| \geq n$. Write $W = uvxyz$ using the pumping lemma.

Let $W = a^n b^n c^n$.

We write $W = uvxyz$ where $|vy| \geq 1$ and $|vxy| \leq n$, then

pumping lemma says that $uv^i xy^i z \in L$ for all $i \geq 0$.

Observations:

a. But this is not possible; because if v or y contains two symbols from {a,b,c} then uv^2xy^2z contains a 'b' before an 'a' or a 'c' before 'b', then uv^2xy^2z will not be in L.

b. In other case, if v and y each contains only a's, b's or only c's, then uv^ixy^iz cannot contain equal number of a's, b's and c's. So uv^ixy^iz will not be in L.

c. If both v and y contains all three symbols a, b and c, then uv^2xy^2z will not be in L by the same logic as in observation (a).

So it is a contradiction to our statement. So L is not context free.

Step 3:

Find a suitable k so that $uv^kxy^kz \notin L$. This is a contradiction, and so L is not context free.

By pumping lemma, $uv^0xy^0z = uxz \in L$.

So $|uxz|$ is a prime number, say q.

Let $|vy| = r$.

Then $|uv^qxy^qz| = q + qr$.

Since, $q + qr$ is not prime, $uv^qxy^qz \notin L$. This is a contradiction. Therefore, L is not context free.

Questions (from Old syllabus of S7CS TOC)

MGU/Nov2011

1. Define a pushdown automata (4marks).

2. Design a context free grammar that accepts the language $L = \{0^n 1^n / n \geq 1\}$ (4marks).

3a. Design a pushdown automata that accepts the language, $L = \{w/w$ contains equal number of 0's and 1's } from the input alphabet {0,1}.

OR

b. Differentiate between deterministic and non-deterministic PDA with examples (12marks).

MGU/April2011

1. Define a context free language (4marks).

2a. Construct a context free grammar to generate $\{W \subset W^R / W \in \{a, b\}^*\}$.

OR

b. If L is given by $L = \{a^n b^n c^n\}/n \geq 1$, check whether L is CFL or not? Prove (12marks).

MGU/Nov2010

1. Define PDA and what are the languages accepted by PDA (4marks).

2a. ii. Show that $\{0^x/x$ is prime} is not context free (12marks).

3a. Construct a PDA to accept $L = \{w \in \{a, b\}^* / w = wR\}$.

OR

b. Prove that context free language is closed under union (12marks).

MGU/May2010

1. Define context free grammar. Construct context free grammar that generate the language $\{wcw^r ? w \in \{a, b\}^*\}$(4marks).

2. Differentiate deterministic PDA from non-deterministic PDA (4marks).

3a. Construct context free grammar that generate the language $\{wcw^R / w \in \{a, b\}^*\}$.

OR

b. Construct a context free grammar for the given language $L = \{anbn1/n >= 1\} \cup \{amb\, 2m/m > -1\}$ and hence a PDA accepting L by empty stack (12marks).

MGU/Nov2009

1. Give pumping lemma to prove that given language L is not context free (4marks).

2. Give an example of a language accepted by a PDA but not by DPDA (4marks).

3a. Prove that $\{op/p$ is prime} is not context free.

OR

b. Convert the grammar $S \longrightarrow ABb/a,\ A \longrightarrow aaA/B\ B \longrightarrow bAb$ into Greibach normal form (12marks).

MGU/Nov2008

1. Give the formal definition of context free grammar (4marks).

2a i. Construct automation that accepts the language $S \longrightarrow aA,\ A \longrightarrow abB/b,\ B \longrightarrow aA/a$ (6marks).

OR

b. Prove that if L is $L(M_2)$ for some PDA M_2, then L is $N(M_1)$, for some PDA M_1 (12marks).

MGU/May2008

1. State pumping lemma for context free languages (4marks).

2. What is acceptance concept of push down automata (4marks).

3. Explain two normal forms of context free grammar (4marks).

4a. ii. Show that $(0^{n/n}$ is prime} is not context free (6marks).

5a. i. Construct PDA for the language $L = \{x \in (a,b)^*/n_a(x) > n_b(n)\}$.

ii. Explain applications of PDA.

OR

b. Construct PDA equivalent to the grammar $S \longrightarrow aAA,\ A \longrightarrow aS/bS/a$ (12marks).

MGU/Dec2007

1. Define context free language (4marks).

2a ii. Convert CFG to NDFA for $S \longrightarrow ABaC,\ A \longrightarrow BC,\ B \longrightarrow b/E,\ C \longrightarrow D,\ D \longrightarrow d$ (6marks).

3a. Construct PDA to accept the language given by $\{w \in (a,b)^*/w$ has the same number of a's as that of b's }.

OR

b. Is $\alpha = \{0^n/n$ is prime } is context free or not. Prove the answer (12marks).

MGU/July2007

1. When do you call a context free grammar ambigous? Give an example (4marks).

2. Give a CFG that generates all palindromes over {a,b} and show that derivation of ababa (4marks).

3a. i. Construct a PDA which accepts the language $\{wcw^R/w \in \{a,b\}^*\}$ and show the processing of abcba.

ii. Briefly explain parsing using PDA with an example.

OR

b. i. Construct a PDA which accepts the language $\{ww^R/w \in \{a,b\}^*\}$ and show the processing of abba.

ii. Design a CFG for the language $L = \{a^n b^m/n$ not equal to m} (12marks).

MGU/Jan2007

1. Find a CFG for the language $L = \{a^n b^{2n} c^m/n, m \geq 0\}$ (4marks).

2. Find a CFG for the language of all strings over [a,b] with exactly one a or exactly one b (4marks).

3a. i. Remove all unit productions, useless productions and ε productions from the grammar-

$S \longrightarrow aA|aBB$

$A \longrightarrow aaA|\varepsilon$

$B \longrightarrow bB|bbC$

$C \longrightarrow B$

What language does the grammar generate?

ii. Convert the following grammar to Chomsky normal form:

$S \longrightarrow ABa$

$A \longrightarrow aaB$

$B \longrightarrow Ac$

OR

b. Prove that for any context free language L there exists a non-deterministic PDA, M such that L=L(M). (12marks)

MGU/July2006

1. What is a context free language? Explain with an example (4marks).

2. Define a PDA (4marks).

3a. Construct a PDA that accepts the language $L = \{a^n b a^m / n,\ m \geq 1\}$ by empty stack.

OR

b. i. Given the CFG $(\{S\}, \{a, b\}, \{S \longrightarrow SaS,\ S \longrightarrow b\},\ S)$, draw the derivation tree for the string bababab.

ii. Give a CFG which generates the language $L = \{a^n b^n c^n / n \geq 1\}$ (12marks).

MGU/Nov2005

1. Give the formal definition of a grammar (4marks).

2. Define a push down automata (4marks).

3. What is meant by an inherently ambigous CFL (4marks)?

4a. Prove that if L is $L(M_2)$ fro some PDA M_2 then L is $N(M_1)$ for some PDA M_1.

OR

b. i. Give a context free grammar which generates the language $L = \{w/w$ contains twice as many 0s and 1s}.

ii. Consider the grammar: $S \longrightarrow SS + /SS * /a$. Explain how the string $aa + a*$ can be generated by the grammar (12marks).

References

.

Pandey, A, K (2006). An Introduction to automata Theory and Formal Languages. Kataria & Sons.

Mishra, K, L, P; Chandrasekaran, N (2009). Theory of Computer Science. PHI.

Nagpal, C, K (2011). Formal Languages and Automata Theory. Oxford University Press.

Murthy, B, N, S (2006). Formal Languages and Automata Theory. Sanguine.

Kavitha,S; Balasubramanian,V (2004). Theory of Computation. Charulatha Pub.

Linz, P (2006). An Introduction to Formal Languages and Automata. Narosa.

Hopcroft, J, E; Motwani, J; Ullman, J, D (2002). Introduction to Automata Theory, Languages and Computation. Pearson Education.

website: http://sites.google.com/site/sjcetcssz

St. Joseph's College of Engineering & Technology Palai

Department of Computer Science & Engineering

S4 CS

CS010 406 Theory of Computation

Module 4

Theory of Computation - Module 4

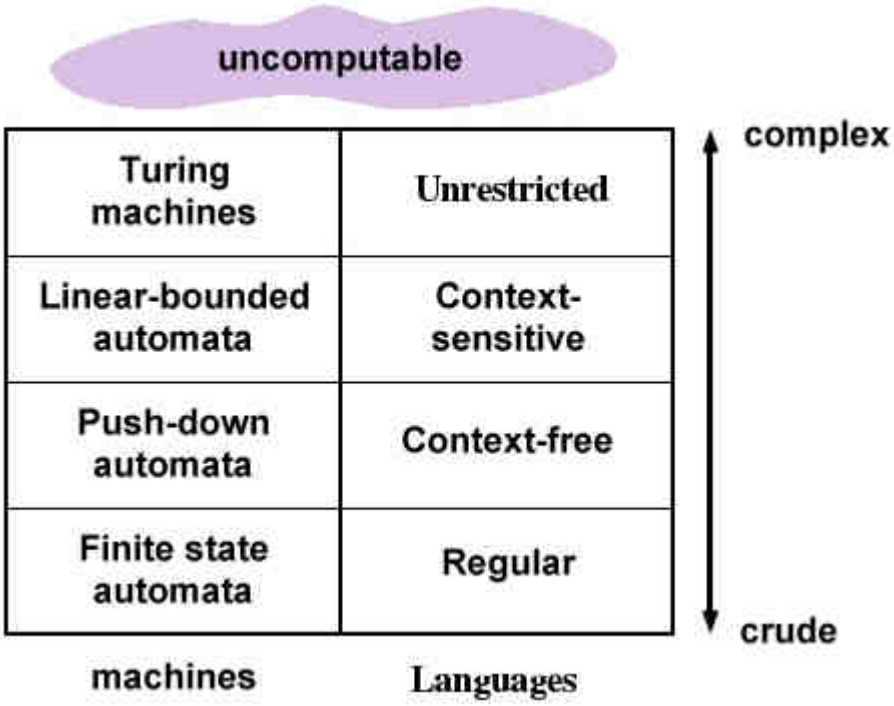
Syllabus

Turing Machines – Formal definition – Language acceptability by TM –TM as acceptors, Transducers - designing of TM- Two way infinite TM- Multi tape TM - Universal Turing Machines- Church's Thesis-Godelization.- - Time complexity of TM - Halting Problem - Rice theorem - Post correspondence problem- Linear Bounded Automata.

Contents

Turing Machines

Consider the following figure:



From the diagram, unrestricted grammars produce unrestricted languages (Type-0 languages). These Type-0 languages are recognised using Turing machines.

Following is an example for unrestricted grammar:

S $\longrightarrow$ ACaB

CaBc $\longrightarrow$ aaC

CB $\longrightarrow$ DB

aD $\longrightarrow$ Da

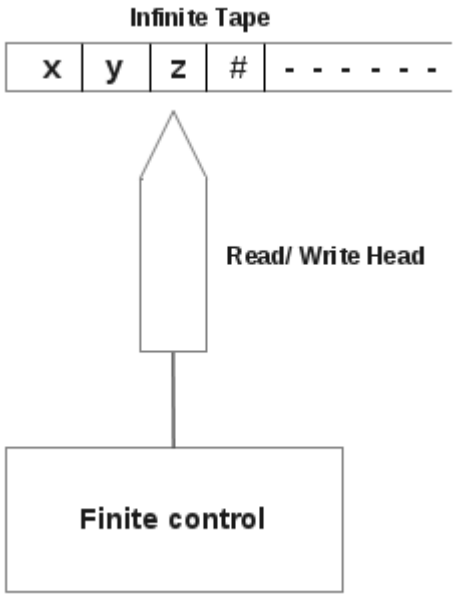
aEC $\longrightarrow$ Ea

AE $\longrightarrow$ ε

Here LHS and RHS can contain any set of terminals and non terminals.

Part I. Turing Machine (TM)

A Turing machine is a computing device that can be represented as,



From the above diagram, a TM consists of a,

1. Tape

   It is divided into a number of cells. Tape is infinite. A cell can hold a tape symbol or a blank symbol (#).

2. Finite control

   At a particular instant, finite control is in a state. States are divided into,

   Start state (q_0),

   Halt state (h),

   Intermediate states ($q_1, q_2, ...$).

3. Tape head

   Tape head points to one of the tape cells. It communicates between finite control and tape. It can move left , right or remain stationary.

1 Definition of a TM

A Turing machine is,

$$TM = (Q, \sum, \Gamma, \delta, q_0, \#, h)$$

where

Q is a set of states,

$\sum$ is a set of input symbols,

Γ is a set of tape symbols, including #,

δ is the next move function from

$$(Q \times \Gamma) \longrightarrow (Q \times \Gamma \times \{L, R, N\}),$$

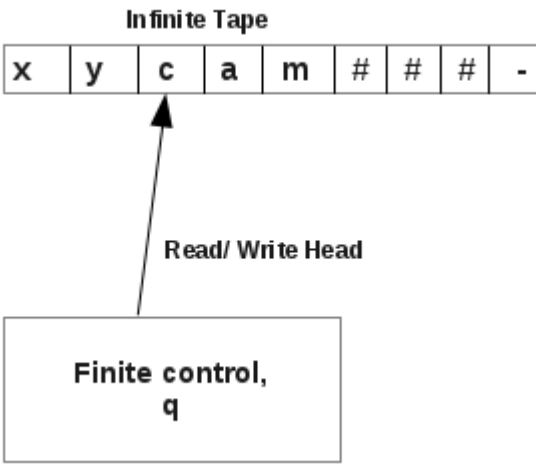
q_0 is the start state,

$\#$ is the blank symbol,

h is the halt state.

Example:

Consider the following TM,

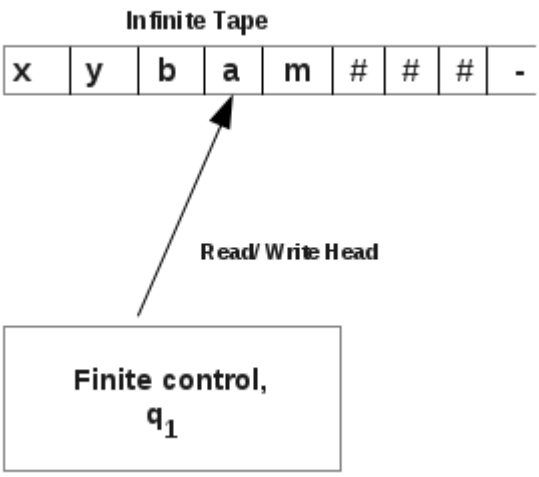


Here TM is in state, q and head points to tape symbol, c.

Let a move is defined as,

$$\delta(q, c) = (q_1, b, R)$$

This means, if TM is in state q and points to input symbol, c, then it enters state, q_1, replaces symbol, c with b and moves one position towards right (R). Thus the TM now is,

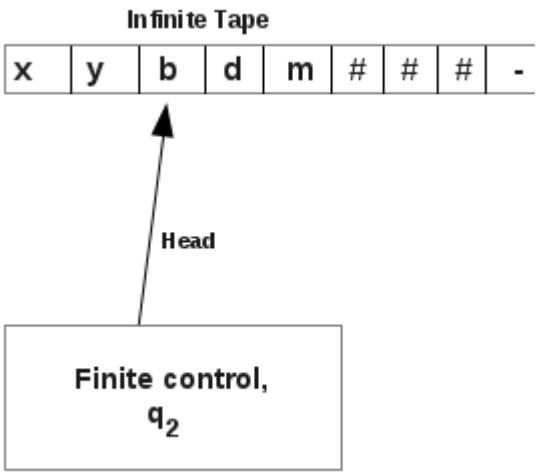


Now TM is in state q_1,and head points to the tape symbol, a.

Let a move of the TM is defined as,

$$\delta(q_1, a) = (q_2, d, L)$$

This means, if TM is in state q_1 and points to input symbol, a, then it enters state, q_2, replaces symbol, a with d and moves one position towards left (L). Thus the TM now is,

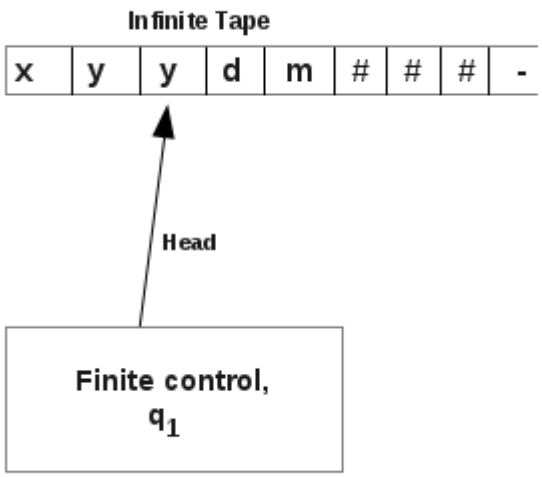


Now TM is in state q_2,and head points to the tape symbol, b.

Let a move of the TM is defined as,

$$\delta(q_2, b) = (q_1, y, N)$$

This means, if TM is in state q_2 and points to input symbol, b, then it enters state, q_1, replaces symbol, b with y and head remains in the same position (N). Thus the TM now is,



Now TM is in state q_1,and head points to the tape symbol, y.

Example:

Consider a Turing machine,

$$TM = (Q, \sum, \Gamma, \delta, q_0, \#, h)$$

where

$$Q = \{q_0, h\}$$

$$\sum = \{a, b\}$$

$$\Gamma = \{a, b, \#\}$$

$$q_0 = q_0$$

$\# = \#$

$h = h$

δ is defined as,

$\delta(q_0, a) = (q_0, \#, L)$

$\delta(q_0, b) = (q_0, \#, L)$

$\delta(q_0, \#) = (h, \#, N)$

$\delta(h, \#) = ACCEPT$

Language Acceptability by TM

A string, w is said to be accepted by a Turing machine, if TM halts in an accepting state. That is,

$q_0 w \overset{*}{\vdash} \alpha_1 h \alpha_2$

A string, w is not accepted by a TM, if TM halts in a non-accepting state or does not halt.

Example 1:

Consider a Turing machine,

$TM = (Q, \sum, \Gamma, \delta, q_0, \#, h)$

where

$Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$

$\sum = \{0, 1, x, y\}$

$\Gamma = \{0, 1, x, y, \#\}$

$q_0 = q_1$

$\# = \#$

$h = q_6$

δ is defined as,

$\delta(q_1, 0) = (q_2, x, R)$ $\quad$ $\delta(q_1, \#) = (q_5, \#, R)$

$\delta(q_2, 0) = (q_2, 0, R)$ $\quad$ $\delta(q_2, 1) = (q_3, y, L)$

$\delta(q_2, y) = (q_2, y, R)$ $\quad$ $\delta(q_3, 0) = (q_4, 0, L)$

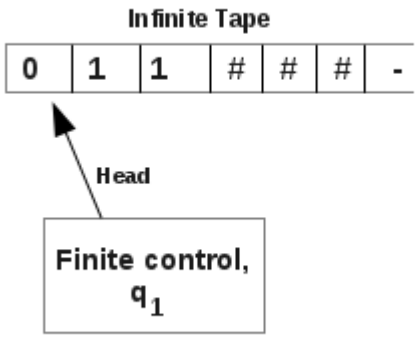
$\delta(q_3, x) = (q_5, x, R)$ $\quad$ $\delta(q_3, y) = (q_3, y, L)$

$\delta(q_4, 0) = (q_4, 0, L)$ $\quad$ $\delta(q_4, x) = (q_1, x, R)$

$\delta(q_5, y) = (q_5, x, R)$ $\quad$ $\delta(q_5, \#) = (q_6, \#, R)$

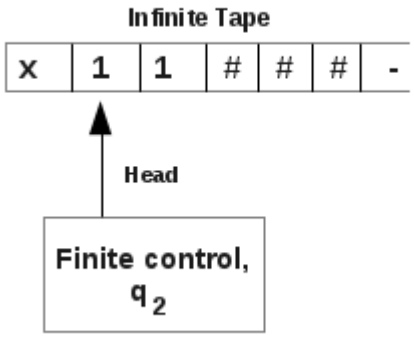
Check whether the string 011 is accepted by the above TM?

Initially, TM is in start state, q_1 and the tape contains the given string 011. Initially, head points to symbol 0 in the input string.



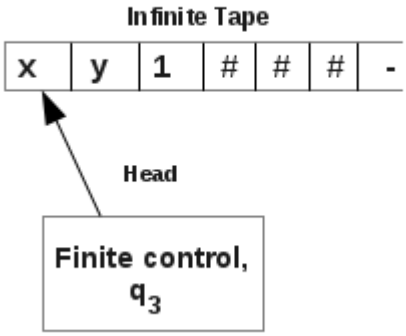
A given transition is, $\delta(q_1, 0) = (q_2, x, R)$

Then, TM becomes,



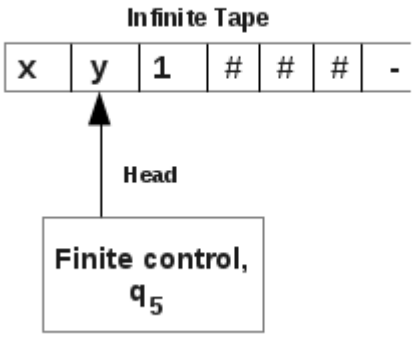
A given transition is, $\delta(q_2, 1) = (q_3, y, L)$

Then, TM becomes,



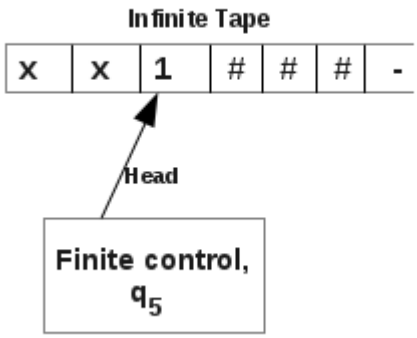
A given transition is, $\delta(q_3, x) = (q_5, x, R)$

Then, TM becomes,



A given transition is, $\delta(q_5, y) = (q_5, x, R)$

Then, TM becomes,



$\delta(q_5, 1)$ is not defined for the TM. So the string 011 is not accepted by the above TM.

Example 2:

Consider a Turing machine,

$$TM = (Q, \sum, \Gamma, \delta, q_0, \#, h)$$

where

$$Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$$

$$\sum = \{0, 1, x, y\}$$

$$\Gamma = \{0, 1, x, y, \#\}$$

$$q_0 = q_1$$

$$\# = \#$$

$$h = q_6$$

δ is defined as,

$$\delta(q_1, 0) = (q_2, x, R) \qquad \delta(q_1, \#) = (q_5, \#, R)$$

$$\delta(q_2, 0) = (q_2, 0, R) \qquad \delta(q_2, 1) = (q_3, y, L)$$

$$\delta(q_2, y) = (q_2, y, R) \qquad \delta(q_3, 0) = (q_4, 0, L)$$

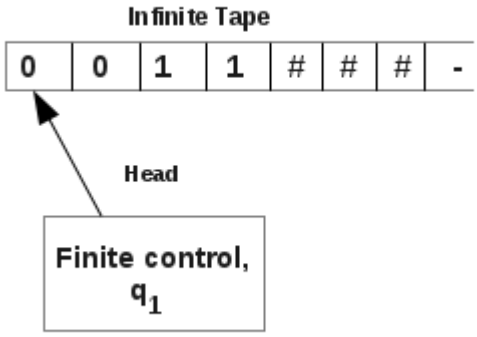
$$\delta(q_3, x) = (q_5, x, R) \qquad \delta(q_3, y) = (q_3, y, L)$$

$$\delta(q_4, 0) = (q_4, 0, L) \qquad \delta(q_4, x) = (q_1, x, R)$$

$$\delta(q_5, y) = (q_5, x, R) \qquad \delta(q_5, \#) = (q_6, \#, R)$$

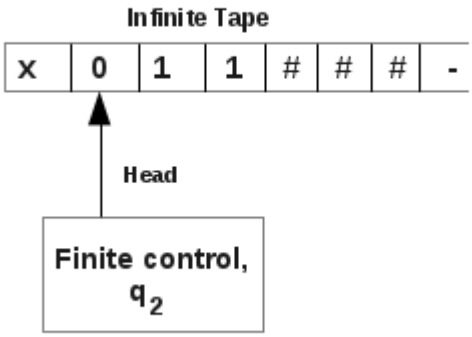
Check whether the string 0011 is accepted by the above TM?

Initially, TM is in start state, q_1 and the tape contains the given string 0011. Initially, head points to symbol 0 in the input string.



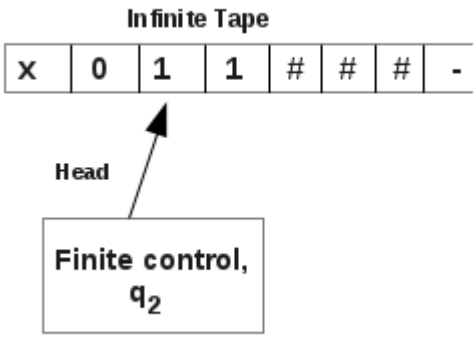
A given transition is, $\delta(q_1, 0) = (q_2, x, R)$

Then, TM becomes,



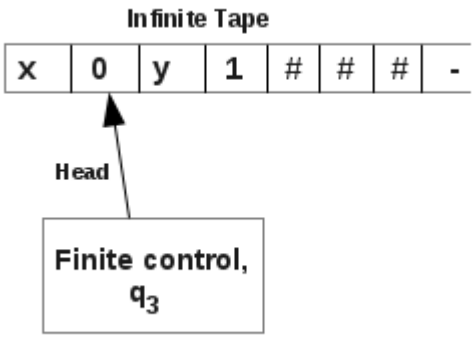
A given transition is, $\delta(q_2, 0) = (q_2, 0, R)$

Then, TM becomes,



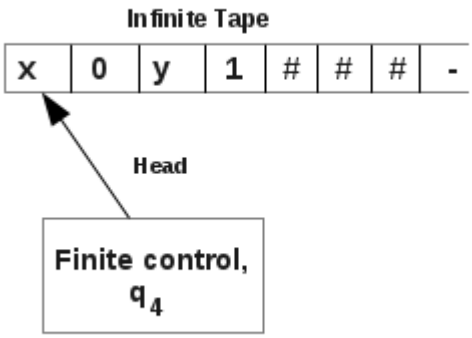
A given transition is, $\delta(q_2, 1) = (q_3, y, L)$

Then, TM becomes,



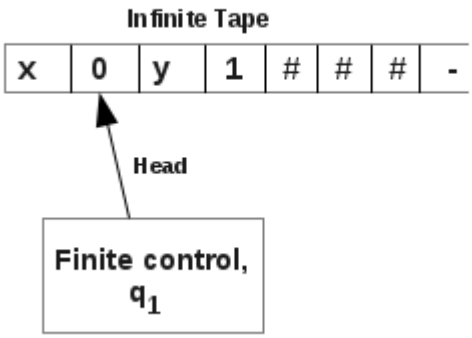
A given transition is, $\delta(q_3, 0) = (q_4, 0, L)$

Then, TM becomes,



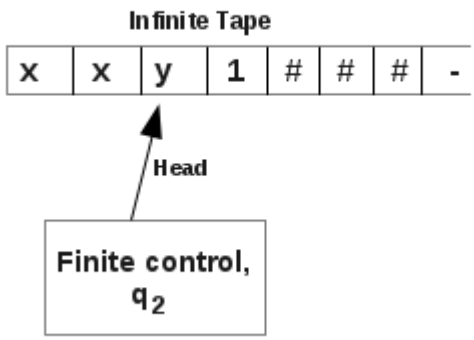
A given transition is, $\delta(q_4, x) = (q_1, x, R)$

Then, TM becomes,



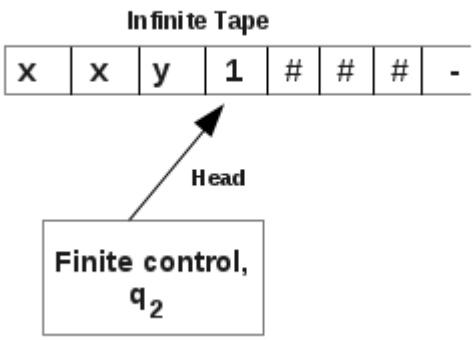
A given transition is, $\delta(q_1, 0) = (q_2, x, R)$

Then, TM becomes,



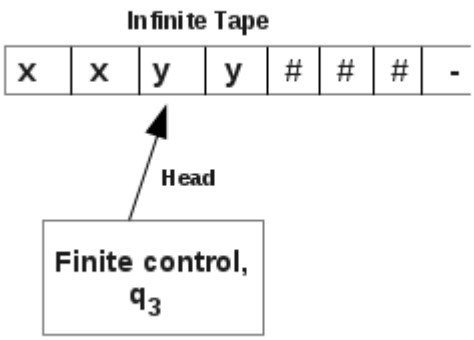
A given transition is, $\delta(q_2, y) = (q_2, y, R)$

Then, TM becomes,



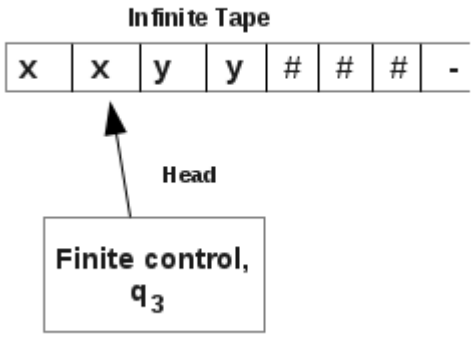
A given transition is, $\delta(q_2, 1) = (q_3, y, L)$

Then, TM becomes,



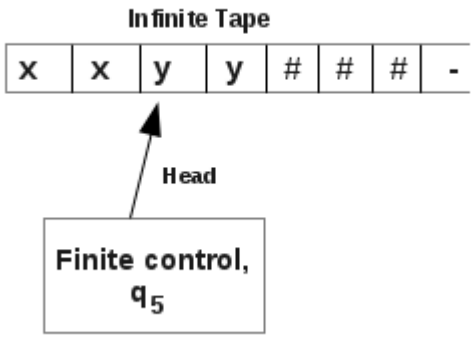
A given transition is, $\delta(q_3, y) = (q_3, y, L)$

Then, TM becomes,



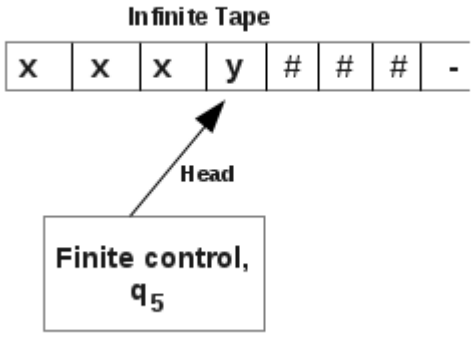
A given transition is, $\delta(q_3, x) = (q_5, x, R)$

Then, TM becomes,



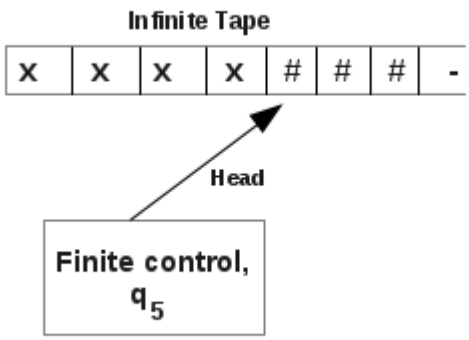
A given transition is, $\delta(q_5, y) = (q_5, x, R)$

Then, TM becomes,



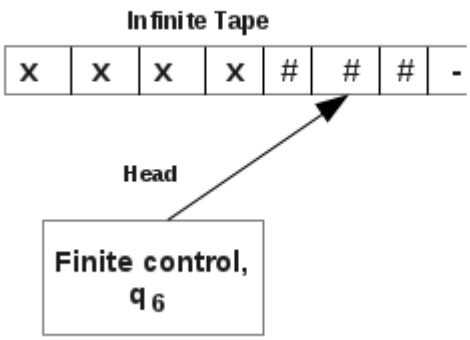
A given transition is, $\delta(q_5, y) = (q_5, x, R)$

Then, TM becomes,



A given transition is, $\delta(q_5, \#) = (q_6, \#, R)$

Then, TM becomes,



q_6 is the halt state or accepting state of the TM. So the string 0011 is accepted by the above TM.

Example 3:

Consider a Turing machine,

$$TM = (Q, \sum, \Gamma, \delta, q_0, \#, h)$$

where

$$Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$$

$$\sum = \{0, 1, x, y\}$$

$$\Gamma = \{0, 1, x, y, \#\}$$

$$q_0 = q_1$$

$$\# = \#$$

$$h = q_6$$

δ is defined as,

$$\delta(q_1, 0) = (q_2, x, R) \qquad \delta(q_1, \#) = (q_5, \#, R)$$

$$\delta(q_2, 0) = (q_2, 0, R) \qquad \delta(q_2, 1) = (q_3, y, L)$$

$$\delta(q_2, y) = (q_2, y, R) \qquad \delta(q_3, 0) = (q_4, 0, L)$$

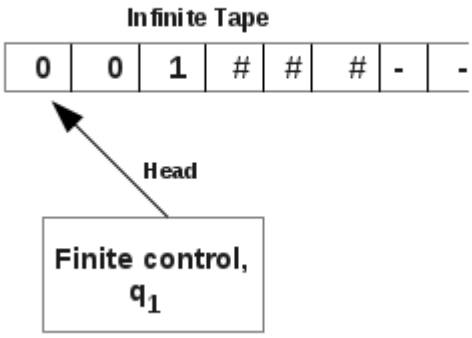
$$\delta(q_3, x) = (q_5, x, R) \qquad \delta(q_3, y) = (q_3, y, L)$$

$$\delta(q_4, 0) = (q_4, 0, L) \qquad \delta(q_4, x) = (q_1, x, R)$$

$\delta(q_5, y) = (q_5, x, R)$ $\delta(q_5, \#) = (q_6, \#, R)$

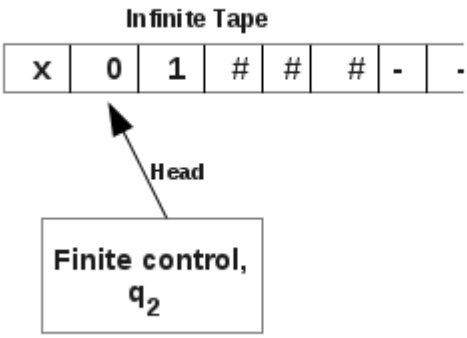
Check whether the string 001 is accepted by the above TM?

Initially, TM is in start state, q_1 and the tape contains the given string 001. Initially, head points to symbol 0 in the input string.



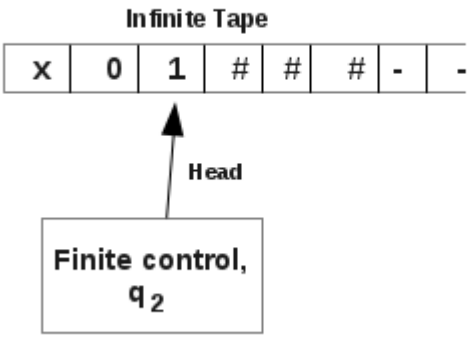
A given transition is, $\delta(q_1, 0) = (q_2, x, R)$

Then, TM becomes,



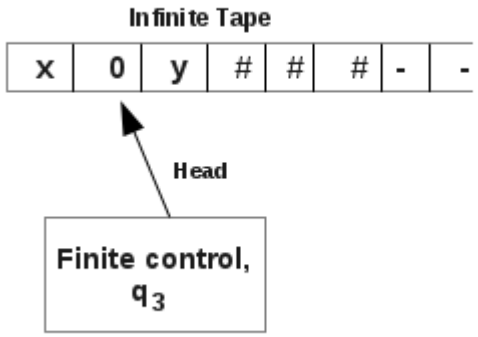
A given transition is, $\delta(q_2, 0) = (q_2, 0, R)$

Then, TM becomes,



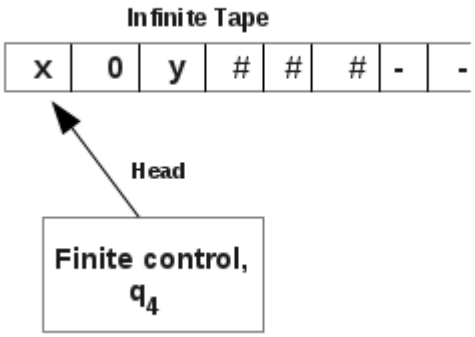
A given transition is, $\delta(q_2, 1) = (q_3, y, L)$

Then, TM becomes,



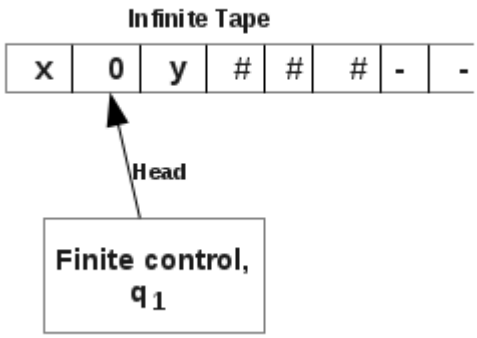
A given transition is, $\delta(q_3, 0) = (q_4, 0, L)$

Then, TM becomes,



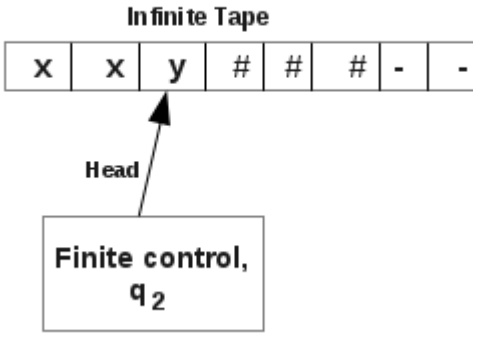
A given transition is, $\delta(q_4, x) = (q_1, x, R)$

Then, TM becomes,



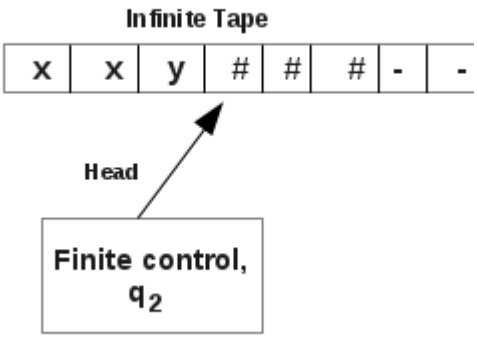
A given transition is, $\delta(q_1, 0) = (q_2, x, R)$

Then, TM becomes,



A given transition is, $\delta(q_2, y) = (q_2, y, R)$

Then, TM becomes,



Now TM halts because no δ is defined for q_2 and #. But q_2 is not the halt state of TM. So the string 001 is not accepted by the above TM.

2 Representation of Turing Machines

We can represent a TM in different ways. They are,

Instantaneous descriptions,

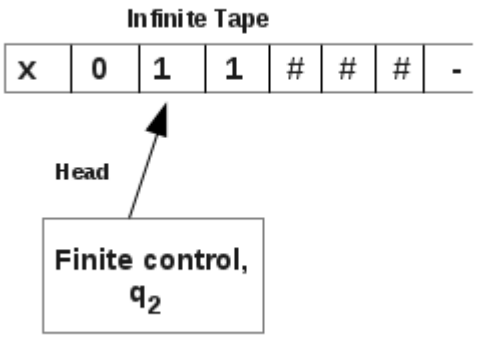
Transition tables,

Transition diagrams.

Representation of TM by Instantaneous Descriptions

In all the above examples, Turing machines shown in the pictures were instaneous descriptions. This means an instant of a TM is shown.

Following shows an instantaneous description of a TM,



Currently, symbol under the head is 1.

Current state is q_2.

Symbols on the left of 1 are x and 0.

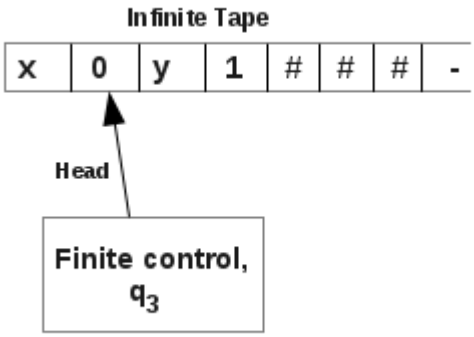
Symbnols on the right of 1 are 1, #, #, ...

This instant of TM can be written as,

$$x\,0\,q_2\,1\,1\,\#\,\#\,\#.$$

Let a transition is defines as, $\delta(q_2, 1) = (q_3, y, L)$

Then TM moves to,



That is,

$$x\,q_2\,0\,y\,1\,\#\,\#\,\#.$$

This transition from one state to other can be written as,

$$x\,0\,q_2\,1\,1\,\#\,\#\,\#\ \vdash\ x\,q_2\,0\,y\,1\,\#\,\#\,\#$$

Representation of TM by Transition Table

Let a Turing machine is defined as,

$$TM = (Q, \sum, \Gamma, \delta, q_0, \#, h)$$

where

$$Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$$

$$\sum = \{0, 1, x, y\}$$

$$\Gamma = \{0, 1, x, y, \#\}$$

$q_0 = q_1$

$\# = \#$

$h = q_6$

δ is defined as,

$\delta(q_1, 0) = (q_2, x, R)$ $\quad$ $\delta(q_1, \#) = (q_5, \#, R)$

$\delta(q_2, 0) = (q_2, 0, R)$ $\quad$ $\delta(q_2, 1) = (q_3, y, L)$

$\delta(q_2, y) = (q_2, y, R)$ $\quad$ $\delta(q_3, 0) = (q_4, 0, L)$

$\delta(q_3, x) = (q_5, x, R)$ $\quad$ $\delta(q_3, y) = (q_3, y, L)$

$\delta(q_4, 0) = (q_4, 0, L)$ $\quad$ $\delta(q_4, x) = (q_1, x, R)$

$\delta(q_5, y) = (q_5, x, R)$ $\quad$ $\delta(q_5, \#) = (q_6, \#, R)$

This TM can be represented as a Transition table.

Transition table corresponding to the above TM is,

	Tape Symbol				
Current State	#	x	y	0	1
$\longrightarrow q_1$	$(q_5, \#, R)$			(q_2, x, R)	
q_2			(q_2, y, R)	$(q_2, 0, R)$	(q_3, y, L)
q_3		(q_5, x, R)	(q_3, y, L)	$(q_4, 0, L)$	
q_4		(q_1, x, R)		$(q_4, 0, L)$	
$*q_5$	$(q_6, \#, R)$		(q_5, x, R)		

Above is the transition table representation of a TM.

In the above transition table, the entry, (q_2, y, R) corresponding to row q_2 and column y denotes the transition,

$\delta(q_2, y) = (q_2, y, R)$

Representation of TM by Transition Diagram

Let a Turing machine is defined as,

$TM = (Q, \sum, \Gamma, \delta, q_0, \#, h)$

where

$Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$

$\sum = \{0, 1, x, y\}$

$\Gamma = \{0, 1, x, y, \#\}$

$q_0 = q_1$

$\# = \#$

$h = q_6$

δ is defined as,

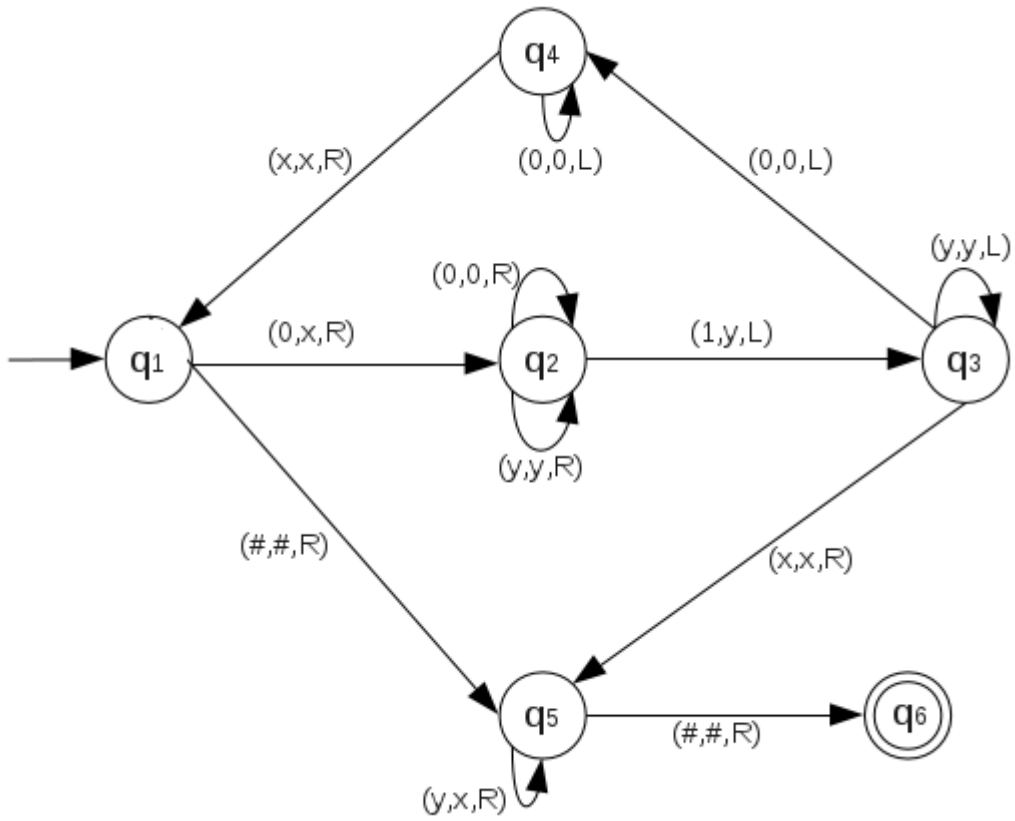
$\delta(q_1, 0) = (q_2, x, R)$ $\quad$ $\delta(q_1, \#) = (q_5, \#, R)$

$\delta(q_2, 0) = (q_2, 0, R)$ $\quad$ $\delta(q_2, 1) = (q_3, y, L)$

$$\delta(q_2, y) = (q_2, y, R) \qquad \delta(q_3, 0) = (q_4, 0, L)$$
$$\delta(q_3, x) = (q_5, x, R) \qquad \delta(q_3, y) = (q_3, y, L)$$
$$\delta(q_4, 0) = (q_4, 0, L) \qquad \delta(q_4, x) = (q_1, x, R)$$
$$\delta(q_5, y) = (q_5, x, R) \qquad \delta(q_5, \#) = (q_6, \#, R)$$

This TM can be represented as a Transition diagram.

Transition diagram corresponding to the above TM is,

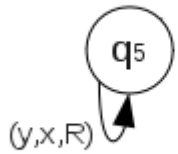


In the above transition diagram, the part of th picture,

denotes the transition,

$$\delta(q_1, 0) = (q_2, x, R)$$

Also the part of the picture,



denotes the transition,

$$\delta(q_5, y) = (q_5, x, R)$$

Exercises:

1. Consider the TM given in the following transition table:

	Tape symbol		
Current State	#	0	1
$\longrightarrow q_1$	$(q_2, 1, L)$	$(q_1, 0, R)$	
q_2	$(q_3, \#, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
q_3		$(q_4, \#, R)$	$(q_5, \#, R)$
q_4	$(q_5, 0, R)$	$(q_4, 0, R)$	$(q_4, 1, R)$
$*q_5$	$(q_2, 0, L)$		

Check whether the string 00 is accepted by the above TM.

2. Consider the TM given in the following transition table:

	Tape symbol				
Current State	#	0	1	x	y
$\longrightarrow q_1$	$(q_6, \#, R)$	$(q_2, 0, R)$			
q_2		$(q_2, 0, R)$	(q_2, y, L)		(q_2, y, R)
q_3		$(q_4, 0, L)$		(q_5, x, R)	(q_3, y, L)
q_4		$(q_4, 0, L)$		(q_1, x, R)	
q_5	$(q_6, \#, R)$				(q_5, y, R)
$*q_6$					

Check whether the string 0011 is accepted by the above TM.

Part II. Designing of Turing Machines

3 Designing of TM

Following are the basic guidelines for designing a TM.

1. The fundamental objective of scanning a symbol in the tape is to know what to do in the future. TM must remember the past symbols scanned. TM can remember this by going to the next unique state.

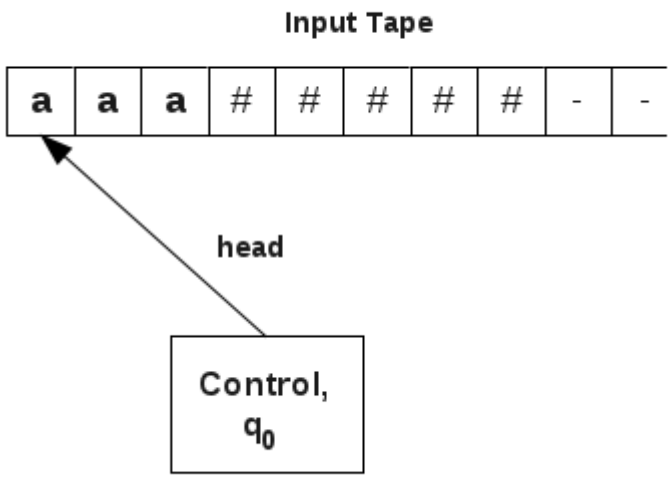
2. The number of states must be minimised. This is achieved by changing the states only when there is a change in the written symbol or when there is a change in the movement of the head.

Following examples show designing of TMs.

Example 1:

Design a turing machine over $\{a\}$ to accept the language L=$\{a^n|n$ is odd$\}$.

The initial instant of the TM is shown below. The input tape contains aaa followed by blanks. The TM is initially in state q_0 and head points to the first a of the input string w.



The working of the TM is as follows:

In state q_0, head reads symbol a from the current cell and

1. keeps the contents of the current cell as a,
2. changes state of the TM to q_1, and
3. moves to the next cell on the right side.

The corresponding transition function is,

$$\delta(q_0, a) = (q_1, a, R)$$

In state q_1, head reads symbol a from the current cell and

1. keeps the contents of the current cell as a,
2. changes the state of the TM to q_0, and
3. moves to the next cell on the right side.

The corresponding transition function is,

$$\delta(q_1, a) = (q_0, a, R)$$

Thsi process continues till a blank symbol, # is reached.

If the blank symbol is reached in state q_1, it indicates that input string contains an odd number of a's.

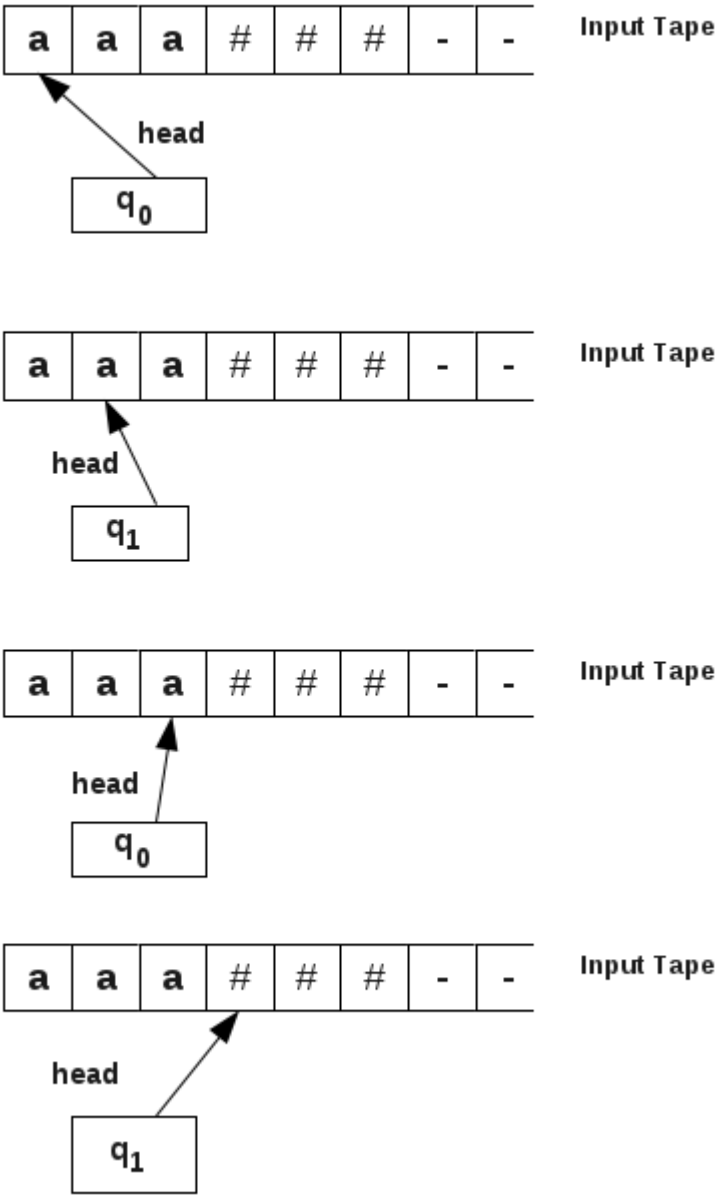
If the blank symbol is reached in state q_0, it indicates that input string contains an even number of a's.

Hence, $q_1$is the final state.

Transition table for the TM is,

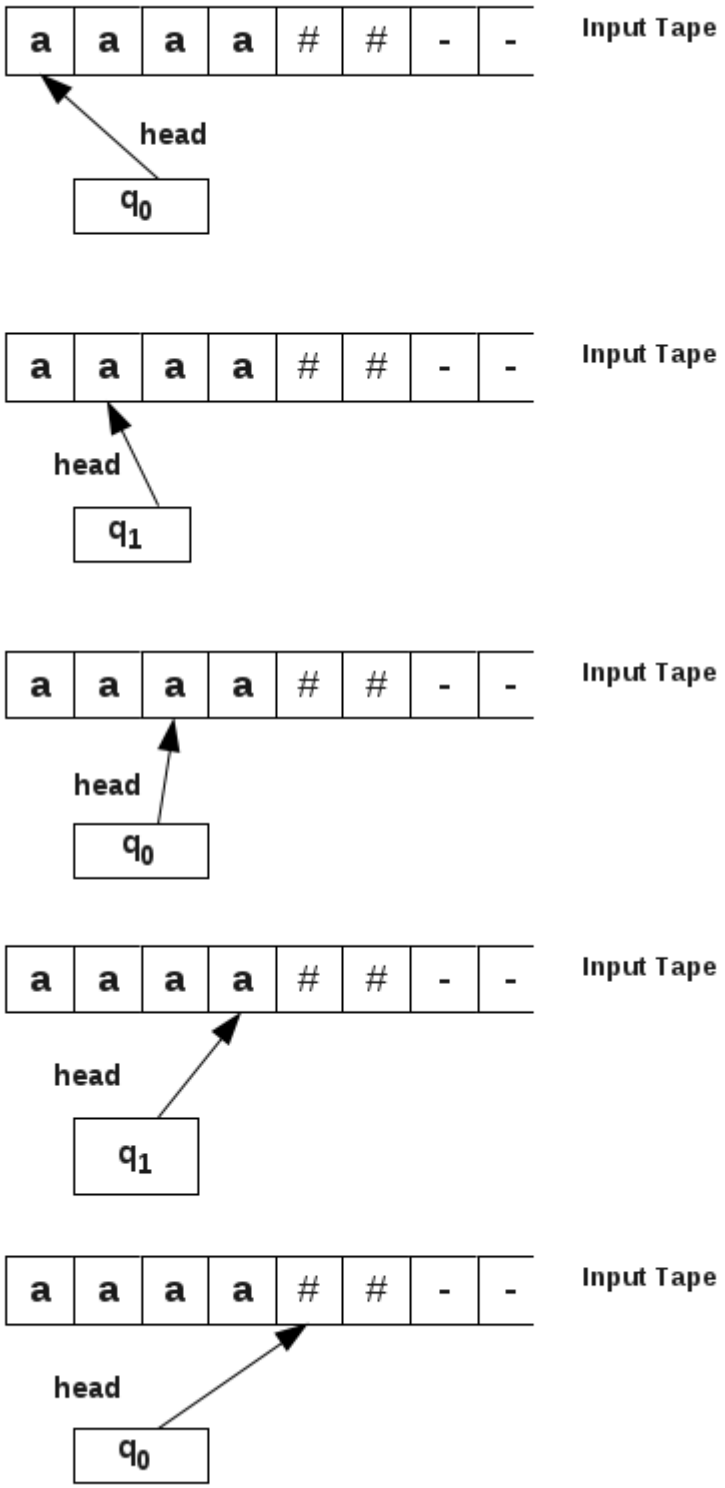
	Input	Symbol
Current State	a	#
$\longrightarrow q_0$	(q_1, a, R)	-
$*q_1$	(q_0, a, R)	-

Consider the processing of string, aaa using the above TM,



The TM halts at state q_1. $q_1$is a final state. So the string aaa is accepted.

Consider the processing of string, aaaa using the above TM,

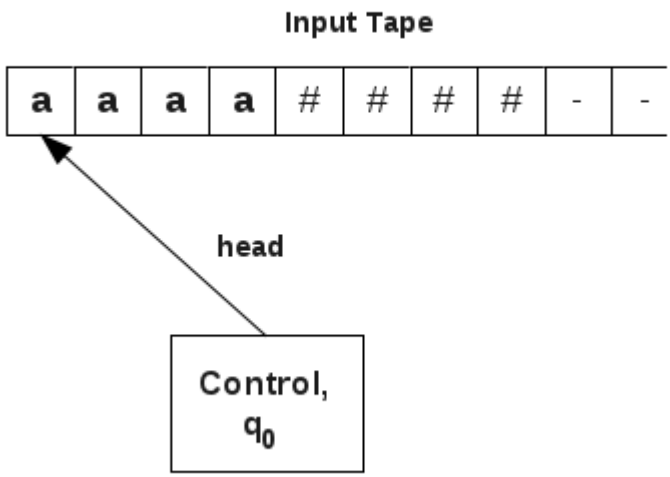


The TM halts at state q_0. $q_0$is not a final state. So the string aaaa is not accepted.

Example 2:

Design a turing machine over $\{a\}$ to accept the language L={$a^n|n$ is even}.

The initial instant of the TM is shown below. The input tape contains $aaaa$ followed by blanks. The TM is initially in state q_0 and head points to the first a of the input string w.



The working of the TM is as follows:

In state q_0, head reads symbol a from the current cell and

1. keeps the contents of the current cell as a,
2. changes state of the TM to q_1, and
3. moves to the next cell on the right side.

The corresponding transition function is,

$$\delta(q_0, a) = (q_1, a, R)$$

In state q_1, head reads symbol a from the current cell and

1. keeps the contents of the current cell as a,
2. changes the state of the TM to q_0, and
3. moves to the next cell on the right side.

The corresponding transition function is,

$$\delta(q_1, a) = (q_0, a, R)$$

Thsi process continues till a blank symbol, # is reached.

If the blank symbol is reached in state q_0, it indicates that input string contains an even number of a's.

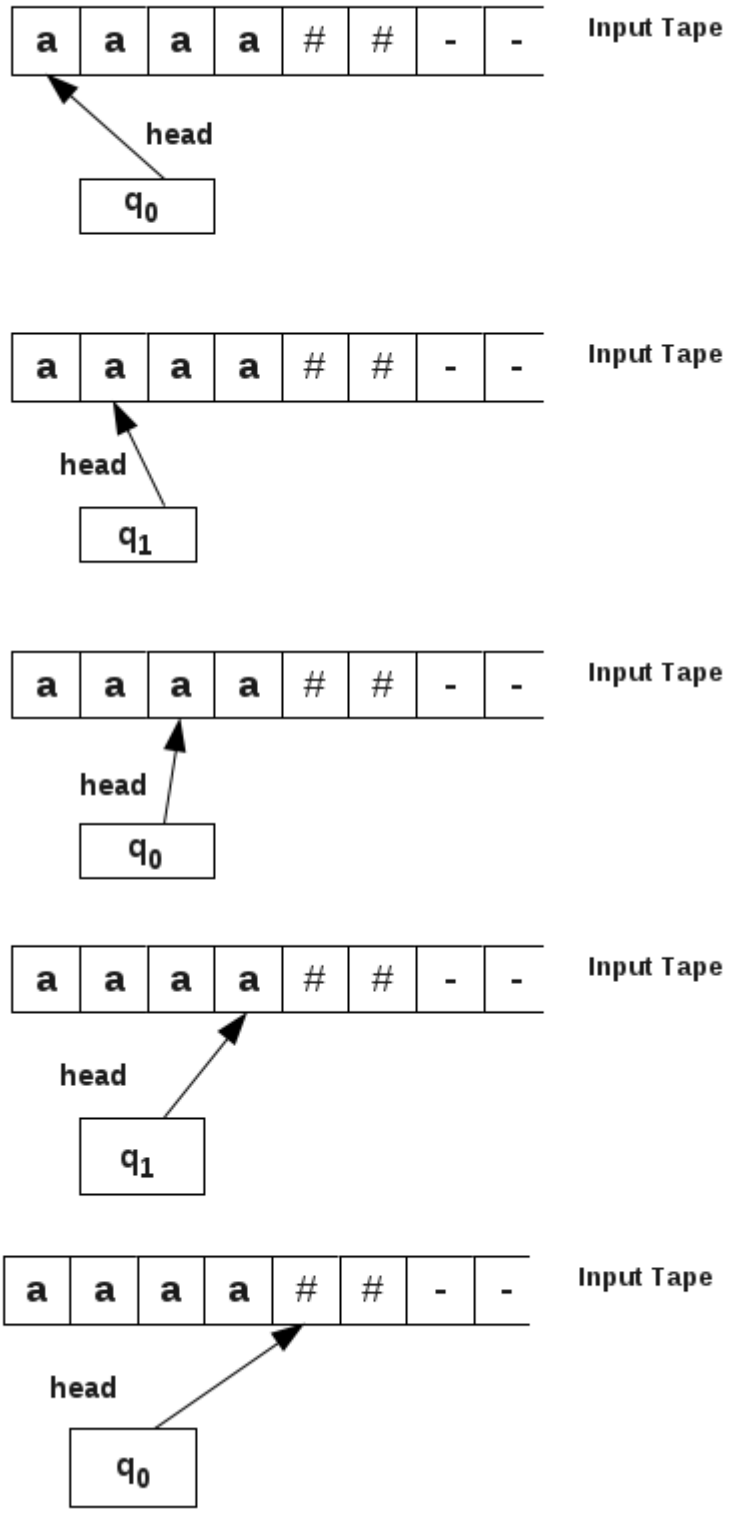
If the blank symbol is reached in state q_1, it indicates that input string contains an odd number of a's.

Hence, $q_1$is the final state.

Transition table for the TM is,

	Input	Symbol
Current State	a	#
$\longrightarrow *q_0$	(q_1, a, R)	-
q_1	(q_0, a, R)	-

Consider the processing of string, aaaa using the above TM,



The TM halts at state q_0. $q_0$is a final state. So the string aaaa is accepted.

Example 3:

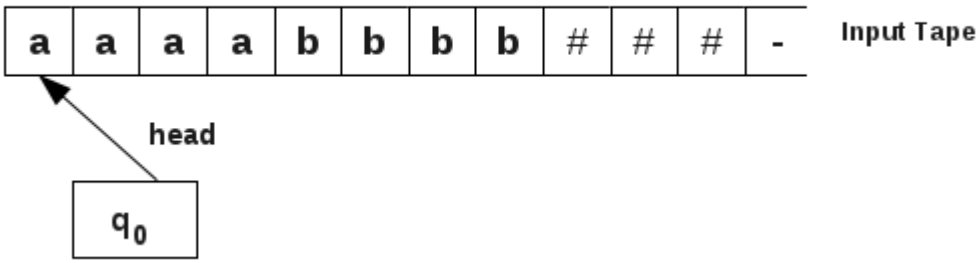
Design a Turing machine over $\{a, b\}$ to accept the language L={$a^n b^n | n \geq 1$ }.

Example strings in this language are ab, aabb, aaabbb, aaaabbbb,.....

Let the string be aaaabbbb.

String recognition process of the Turing machine is as follows:

Initially, TM is in state q_0 and head points to first a in the string.



1. Read the first symbol a of the string and convert this a to blank (#).

2. The head keeps moving towards right till the input string is crossed and the first blank symbol after the string is reached.

3. From this #, turn one cell left and reach the last symbol of the input string. If it is 1, convert it to #.

After performing the first three steps, input tape with aaaabbbb###, will contain #aaabbb####. That is, leftmost a is cancelled with rightmost b.

4. Now after converting the last b to #, the head keeps moving towards left till a # is reached, turn towards right and converts the first available a to #, and keeps moving towards right till a # is reached. It then turns left and converts the first available b to # and keeps moving towards left.

After step 4, input tape will contain ##aabb#####. This marks the cancellation of the next leftmost a by the next rightmost b.

5. The head keeps repeating step 4 till all the a's are cancelled by teh corresponding b's.

6. If the cancellation process is successfully completed, then the string is accepted.

7. If there are remaining uncancelled a's or b's, then the input string is not accepted.

Transition table corresponding to the TM is given below:

	Input Symbol		
Current State	a	b	#
$\longrightarrow q_0$	$(q_1, \#, R)$	—	—
q_1	(q_0, a, R)	(q_2, b, R)	—
q_2	—	(q_2, b, R)	$(q_3, \#, L)$
q_3	—	$(q_4, \#, L)$	—
q_4	(q_5, a, L)	(q_4, b, L)	$(q_6, \#, R)$
q_5	(q_5, a, L)	—	$(q_0, \#, R)$
q_6	—	—	$(q_f, \#, R)$
$*q_f$	—	—	—

The working of the above TM is explained below:

1. The head reads the first symbol of the input string in state q_0 and does the following:

a. If the symbol is # or b, then TM halts. If the first symbol is #, then it is a null string and does not belong to the language. If the symbol is 'b', then it is also an invalid string. Thus, q_0 is a non-final state.

b. If the first symbol is 'a', then head converts this input symbol to #, changes state to q_1, and moves towards right. $\delta(q_0, a) = (q_1, \#, R)$

2. In state q_1, head does the following:

a. If the head is pointing to input symbol a, then symbol 'a' remains unchanged, state q_1 remains unchanged, and head moves to next cell on the right side. $\delta(q_1, a) = (q_1, a, R)$

b. If the head is pointing to input symbol b, then symbol 'b' remains unchanged, state q_1 changes to state q_2, and head moves to next cell on the right side. $\delta(q_1, b) = (q_2, b, R)$

In state q_1, head simply passes over all the 0s of the input string from left to right and changes to state q_2 as soon as symbol b is received.

3. In state q_2, head does the following:

a. If the head is pointing to input symbol b, then symbol 'b' remains unchanged, state q_2 remains unchanged, and head moves to next cell on the right side. $\delta(q_2, b) = (q_2, b, R)$

b. If the head is pointing to symbol #, then symbol '#' remains unchanged, state q_2 changes to state q_3, and head moves to cell on the left side. $\delta(q_2, \#) = (q_3, \#, L)$

In state q_2, head simply passes over all the b's of the input string from left to right and changes to state q_3 as soon as symbol # is received and turns towards left.

4. In state q_3, head does the following:

a. If the head is pointing to input symbol b, then symbol 'b' is replaced with #, state q_3 changes to q_4, and head moves to next cell on the left side. $\delta(q_3, b) = (q_4, \#, L)$

In state q_3, symbol 'b' is replaced with # in response to the 'a' symbol replaced with # in state q_0.

5. In state q_4, head does the following:

a. If the head is pointing to input symbol b, then symbol 'b' remains unchanged, state q_4 remains unchanged, and head moves to next cell on the left side. $\delta(q_4, b) = (q_4, b, L)$

b. If the head is pointing to symbol a, then symbol 'a' remains unchanged, state q_4 changes to state q_5, and head moves to cell on the left side. $\delta(q_4, a) = (q_5, a, L)$

c. If the head is pointing to symbol #, then symbol '#' remains unchanged, state q_4 changes to state q_6, and head moves to cell on the right side. $\delta(q_4, \#) = (q_6, \#, R)$. This indicates that all a's have been cancelled.

In state q_4, head simply passes over all the b's of the input string from right to left and changes to state q_5 as soon as symbol a is received.

6. In state q_5, head does the following:

a. If the head is pointing to input symbol a, then symbol 'a' remains unchanged, state q_5 remains unchanged, and head moves to next cell on the left side. $\delta(q_5, a) = (q_5, a, L)$

b. If the head is pointing to symbol #, then symbol '#' remains unchanged, state q_5 changes to state q_0, and head moves to cell on the right side. $\delta(q_5, \#) = (q_0, \#, R)$

In state q_5, head simply passes over all the a's of the input string from right to left and changes to state q_0 as soon as symbol # is received.

7. In steps 1-6, leftmost a and rightmost b cancel out each other. This is done by converting them to blanks. This is repeated till all the a's are cancelled by the corresponding b's.

When all a's are cancelled by the respective b's, head will point ot a # symbol in state q_6 and moves to final state where it halts and it indicates that the string is accepted.

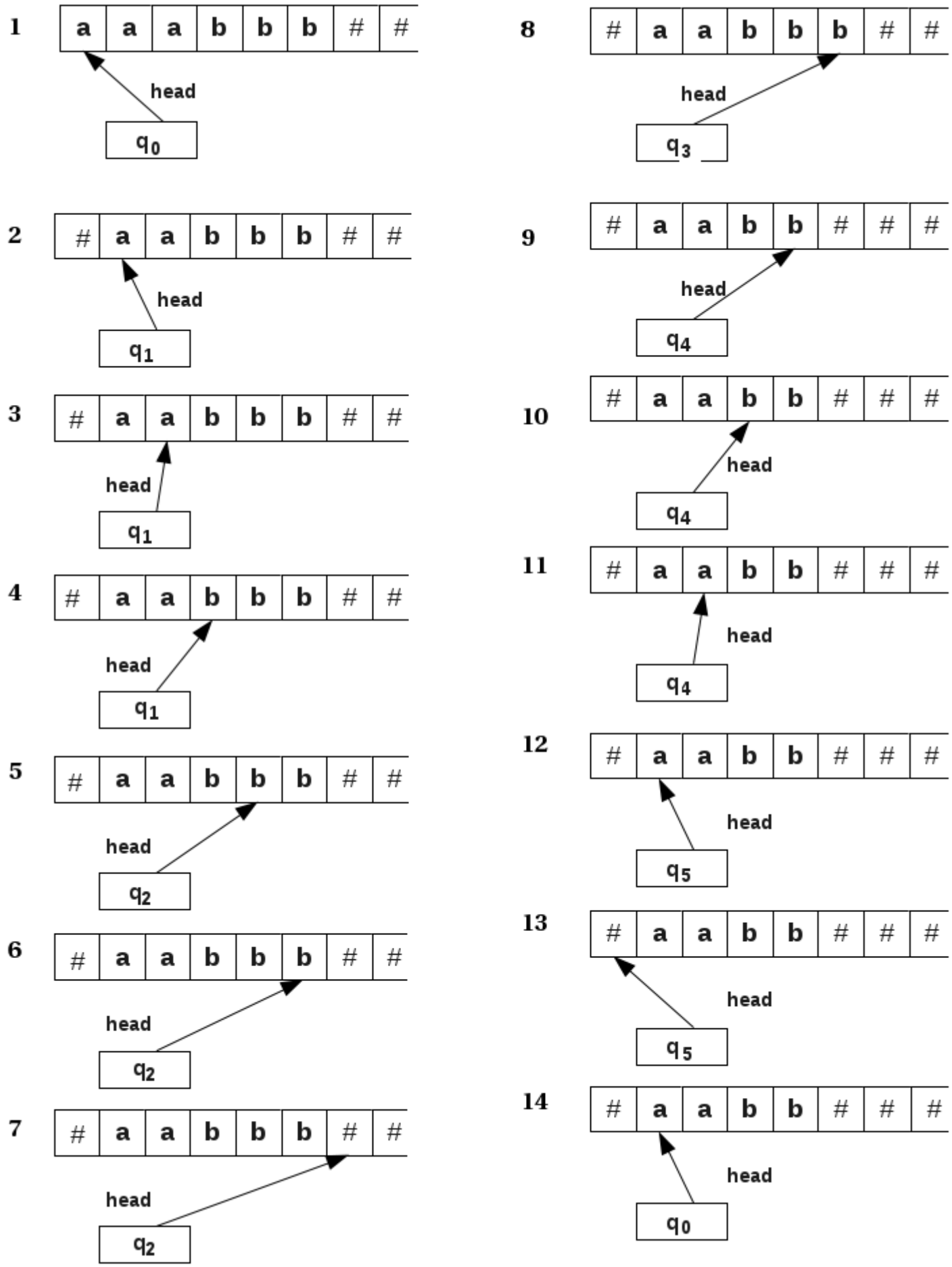
8. In state q_6, head does the following:

a. If the head is pointing to input symbol #, then it indicates that all the a's have been successfully matched with the corresponding b's. Hence the TM changes state to q_f and halts. It indicates taht the string is accepted. $\delta(q_6, \#) = (q_f, \#, R)$

b. If the head is pointing to symbol b, this shows that the number of b's is more than the number of a's and hence the TM halts in state q_6 to indicate the non-acceptance of the string.

Let the input string is aaabbb.

TM processes the string as follows:



After the 14th step leftmost a is cancelled with the rightmost b. This process is repeated and finally we get all #s and TM will reach state q_f. Then the string aaabbb is accepted by the above TM.

Example 4:

Design a Turing machine over $\{a, b\}$ to accept the language L=$\{ww^R | w \in (a, b)^+\}$.

Example strings in this language are aa, abba, bb, baab, ababbbaba, ...

String recognition process of the Turing machine is as follows:

1. Initially, TM is in state q_0 and head points to first symbol in the string.

2. The symbol may be a or b. Aftyer reading this symbol, this symbol is replaced with a # and keeps moving towards right till the whole of the string is crossed and the first # after the string is reached.

3. From this blank, the head turns one cell left and reaches the last symbol of the input string. If this symbol matches with the first symbol of the input string, it converts it into a #.

4. Steps 1-3 match the first and last symbols of the input string. If this matching is successful, the head traverses back and crosses the string to reach the first #. From this #, it turns towards right and matches the leftmost and rightmost symbols of the remaining string. If the matching is successful, then the symbols are converted to #s.

5. The process repeats till the symbols are matched.

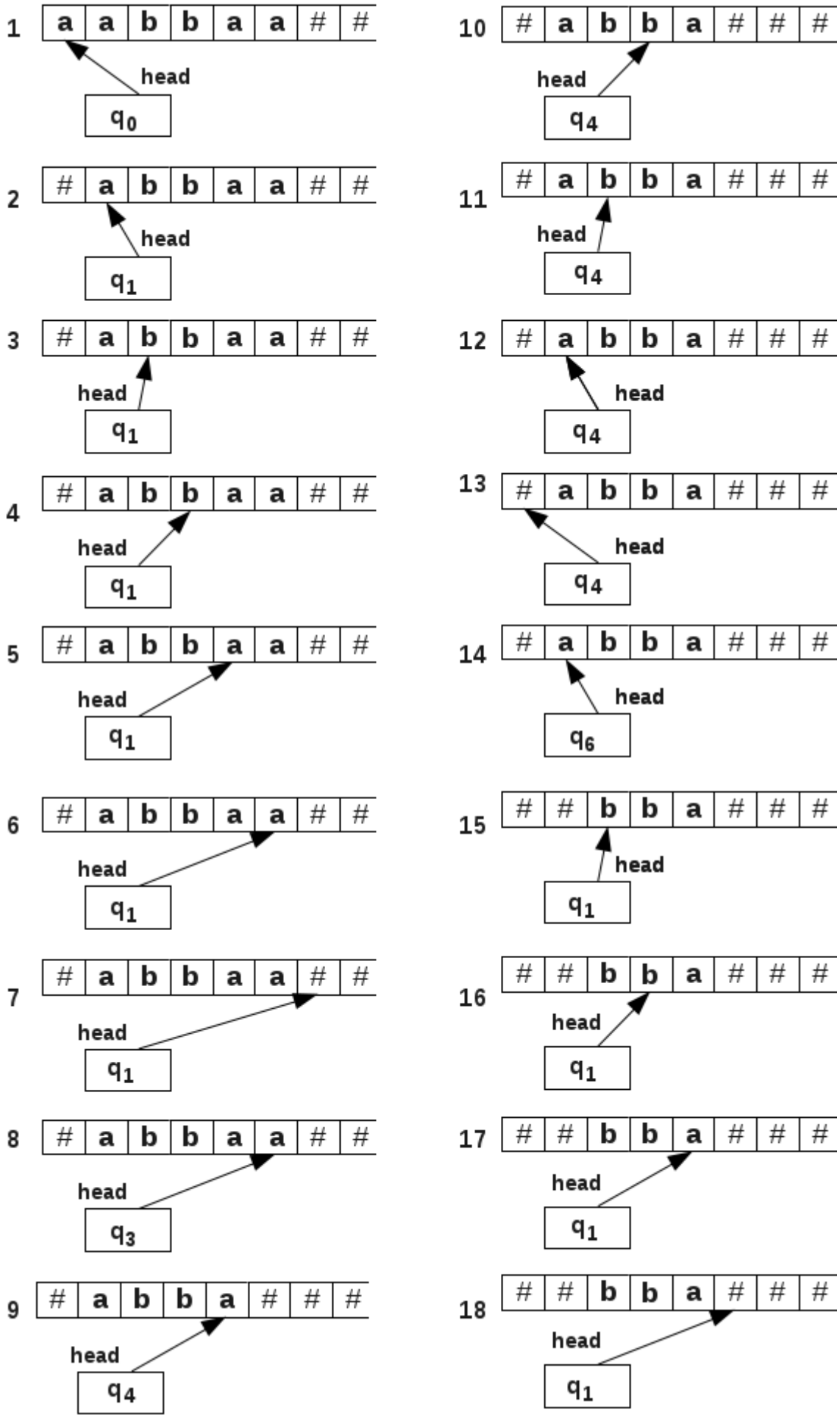
6. To keep track of the symbol to be matched, if symbol 'a' is received in state q_0, then head enters state q_1, and if symbol 'b' is received, then head enters state q_2.

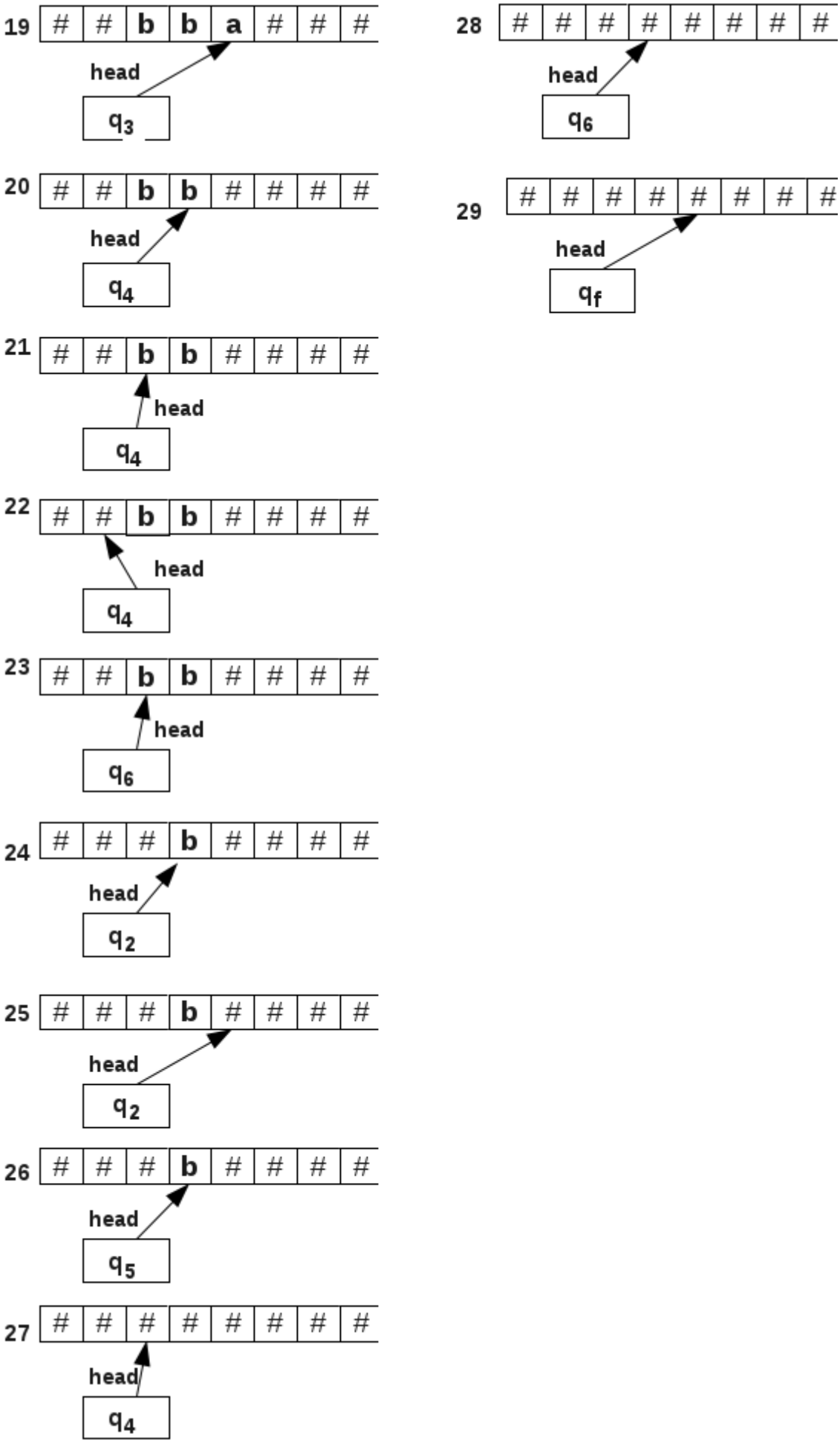
Transition table corresponding to the Turing machien is given below:

	Input Symbol		
Current State	a	b	#
$\longrightarrow q_0$	$(q_1, \#, R)$	$(q_2, \#, R)$	—
q_1	(q_1, a, R)	(q_1, b, R)	$(q_3, \#, L)$
q_2	(q_2, a, R)	(q_2, b, R)	$(q_5, \#, L)$
q_3	$(q_4, \#, L)$	—	—
q_4	(q_4, a, L)	(q_4, b, L)	$(q_6, \#, R)$
q_5	—	$(q_4, \#, L)$	—
q_6	$(q_1, \#, R)$	$(q_2, \#, R)$	$(q_f, \#, R)$
$*q_f$	—	—	—

Let the input string is abaaba.

TM processes the string as follows:


1 a a b b a a # #
head
q0
2 # a b b a a # #
head
q1
3 # a b b a a # #
head
q1
4 # a b b a a # #
head
q1
5 # a b b a a # #
head
q1
6 # a b b a a # #
head
q1
7 # a b b a a # #
head
q1
8 # a b b a a # #
head
q3
9 # a b b a # # #
head
q4
10 # a b b a # # #
head
q4
11 # a b b a # # #
head
q4
12 # a b b a # # #
head
q4
13 # a b b a # # #
head
q4
14 # a b b a # # #
head
q6
15 # # b b a # # #
head
q1
16 # # b b a # # #
head
q1
17 # # b b a # # #
head
q1
18 # # b b a # # #
head
q1


19
b b a # #
head
q3
20
b b # # #
head
q4
21
b b # # #
head
q4
22
b b # # #
head
q4
23
b b # # #
head
q6
24
b # # #
head
q2
25
b # # #
head
q2
26
b # # #
head
q5
27
# # # # # # # #
head
q4
28
# # # # # # # #
head
q6
29
# # # # # # # #
head
qf

Example 5:

Design a Turing machine over $\{a, b, c\}$ to accept the language L=$\{a^n b^n c^n | n \geq 1\}$.

Example strings in this language are abc, aabbcc, aaabbbccc, aaaabbbbcccc, ...

String recognition process of the Turing machine is as follows:

1. Initially, TM is in state q_0 and head points to first symbol, a in the string. a is replaced with a #, changes state to q_1, and moves towards right.

2. In state q_1, the head continues to move towards right till all the a's are passed over and the first 'b' is obtained. Now 'b' is replaced with 'x', changes state to q_2, and again moves towards right.

3. In state q_2, the head continues to move towards right till all the b's are passed over and the first 'c' is obtained. Now 'c' is replaced with 'y', changes state to q_3, and turns back towards left.

4. In steps 1-3, each symbol 'a' is matched with the corresponding symbols 'b' and 'c'. In state q_3, head continues to move towards left and passes over the whole string to reach the blank symbol, #. At #, it turns right and changes state to q_4.

5. In state q_4, head reads symbol 'a', replaces it with a #, changes to state $q_1$and moves towards right.

6. Now steps 1-3 are repeated to match the symbols a, b and c. this process continues till the symbol 'x' is obtained in state q_4. This shows that all a's were replaced with #'s. Now if the string is valid, no 'b' should be available in state q_4.

7. In state q_4,head continues to move towards right till all the x's are passed over and no symbol, 'b' is obtained. If 'b' is found, it indicates a mismatch in the number of 'a' and 'b' symbols. TM halts in state q_4 to indicate the rejection of the string.

If no 'b' is encountered, and if symbol 'y' is obtained, then it changes state to q_5, and moves towards right.

8. In state q_5,head continues to move towards right till all the y's are passed over and no symbol, 'c' is obtained. If 'c' is found, it indicates a mismatch in the number of 'a' and 'c' symbols. TM halts in state q_5 to indicate the rejection of the string.

If no 'c' is encountered, and if symbol '#' is obtained, then it changes state to q_f, and moves towards left.

9. In state q_f, TM halts to indicate the acceptance of the string.

Transition table for the TM is shown below:

	Input Symbol					
Current State	a	b	c	x	y	#
$\longrightarrow q_0$	$(q_1, \#, R)$	—	—	—	—	—
q_1	(q_1, a, R)	(q_2, x, R)	—	(q_1, x, R)	—	—
q_2	—	(q_2, b, R)	(q_3, y, L)	—	(q_2, y, R)	—
q_3	(q_3, a, L)	(q_3, b, L)	—	(q_3, x, L)	(q_3, y, L)	$(q_4, \#, R)$
q_4	$(q_1, \#, R)$	—	—	(q_4, x, R)	(q_5, y, R)	—
q_5	—	—	—	—	(q_5, y, R)	$(q_f, \#, L)$
$*q_f$	—	—	—		—	—

Exercises:

1. Design a TM that accepts the language $L = \{0^n | n$ is a multiple of 3$\}$.
2. Design a TM that accepts the language $L = \{a^n b^{2n} | n > 0\}$.
3. Design a TM over {a,b,c} to accept the language $L = \{wcw | w \in (a|b)^*\}$.
4. Design a TM over {a,b} to accept the language $L = \{ww | w \in (a|b)^*\}$.

Part III. TM as Transducers

4 TM as Transducers

In all the above TMs, TM either accepts or rejects a string. That is, TM acts as a language acceptor.

A TM can function as a transducer. That is, a string is given as input to TM, it produces some output.

We can view a Turing machine as a transducer.

Input to the computation is a set of symbols on the tape.

At the end of computation, whatever remains on the tape is the output.

A TM can be viewed as a transducer for the implementation of function f defined as,

$$\widehat{w} = f(w).$$

A function, f is said to be computable or Turing computable, if there exists a TM $M = (Q, \sum, \Gamma, \delta, q_0, \#, q_f)$ such that

$$q_0 w \overset{*}{\vdash} q_f\, f(w)$$

where w is in the domain of f.

We can see that all common mathematical functions are turing computable.

This means, basic operations such as addition, subtraction, multiplication, division can be performed on it.

This means that a TM is an abstract model of our modern computer system.

Example 1:

Design a Turing machine that computes the following function,

f(m, n) = m + n.

A positive integer on a Turing tape can be represented by an equal number of 0s.

For example,

integer 5 is represented as 00000, and

integer 8 is represented as 00000000.

In the tape two integers are separated using the symbol, $.

Let us assume that m=3 and n=5.

Then the input tape of the Turing machine will be,

0	0	0	$	0	0	0	0	0	#	#	#

After addition, tape will contain the results of addition as shown below:

0	0	0	0	0	0	0	0	#	#	#	#

TM for addition works as follows:

Head reads the 0s of the first number, m and reaches the separator symbol, $. Symbol, $ is replaced with 0 and head moves towards right.

It continues to move towards right till the second number, n is passed over and # is reached.

At #, it turns left and replaces rightmost 0 with #.

Now the tape contains the sum of m and n, and TM halts.

Conversion of the separator symbol, $ to 0, helps TM to make the single number on the tape. The conversion of symbol, 0 to # removes the additional 0 that was generated from the conversion of $ to 0.

Transition table for the TM is shown below:

	Input	Symbol	
Current State	0	$	#
$\longrightarrow q_0$	$(q_0, 0, R)$	$(q_1, 0, R)$	—
q_1	$(q_1, 0, R)$	—	$(q_2, \#, L)$
q_2	$(q_f, \#, L)$	—	—
$*q_f$	—	—	—

Above table is describes as follows:

1. Initially, TM is in state q_0. Head reads the first symbol, 0 and moves towards right without changing the state.

2. Head continues to move towards right till the separator symbol, $ is reached. It replaces $ with 0, changes state to q_1, and continues to move towards right.

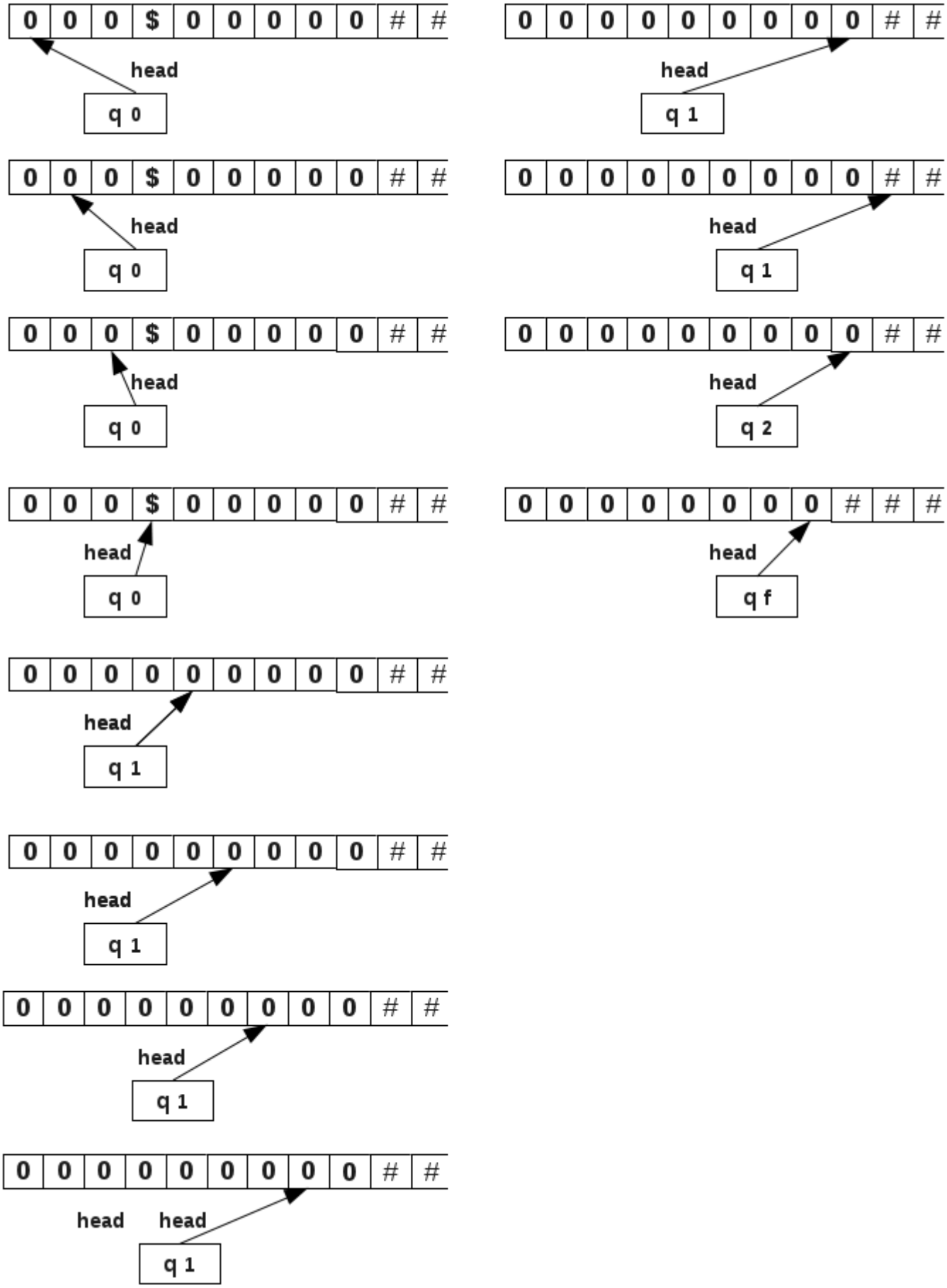
3. Head continues to move towards right and passes over the second number to reach #.

4. On reaching the #, head changes state to q_2 and turns left.

5. In state q_2, it replaces symbol 0 with #, and changes state to q_f.

6. In the final state, TM halts to indicate the completion of the process.

Addition of 3 and 5 is shown below:


0 0 0 $ 0 0 0 0 0 # #
head
q 0
0 0 0 $ 0 0 0 0 0 # #
head
q 0
0 0 0 $ 0 0 0 0 0 # #
head
q 0
0 0 0 $ 0 0 0 0 0 # #
head
q 0
0 0 0 0 0 0 0 0 0 # #
head
q 1
0 0 0 0 0 0 0 0 0 # #
head
q 1
0 0 0 0 0 0 0 0 0 # #
head
q 1
0 0 0 0 0 0 0 0 0 # #
head head
q 1
0 0 0 0 0 0 0 0 0 # #
head
q 1
0 0 0 0 0 0 0 0 0 # #
head
q 1
0 0 0 0 0 0 0 0 0 # #
head
q 2
0 0 0 0 0 0 0 0 # # #
head
q f

Example 2:

Design a Turing machine that can multiply two numbers.

f(m, n) = m x n.

Let m=2, n=4

Tape contains two integers, both containing a termination symbol $, at the end as shown below:

0	0	$	0	0	0	0	$	#	#	#	#

After the multiplication operation, tape contains the result as shown below:

0	0	$	0	0	0	0	$	0	0	0	0	0	0	0	0	#	#

TM for multiplication works as follows:

Leftmost 0 is replaced with symbol x and head moves towards right.

Head moves towards right till the remaining 0s are passed over and the separator symbol $ is reached. It crosses $ and reaches the leftmost 0 of the number n. It replaces this 0 with y and moves towards right.

Head moves towards right and reaches and crosses the second separator symbol, $.

After crossing $, it replaces # with 0.

Now head turns towards left and replaces second leftmost 0 of n to y.

Head then turns right and keeps moving towards right till a # is reached. It replaces # with 0 and turns towards left.

This process is repeated till all the 0's in the number n are converted to y and their copy is made after the second $.

This completes one cycle of copying.

After one cycle of copying, head turns left and converts all y's to 0. It moves towards left and converts the second leftmost 0 of m to x and makes a copy of n after the second $.

The cycle of copying n after the second $ is repeated m times and the number of such cycles is tracked with the help of the number of x symbols in the number m.

With m cycles of copying n after the second $, tape contains mxn numbers of 0s, which is the product.

Transition table for the TM is shown below:

	Symbol				
Current State	0	$	x	y	#
$\longrightarrow q_0$	(q_1, x, R)	$(q_9, \$, L)$	—	—	—
q_1	$(q_1, 0, R)$	$(q_2, \$, R)$		—	—
q_2	(q_3, y, R)	$(q_7, \$, L)$	—	—	—
q_3	$(q_3, 0, R)$	$(q_4, \$, R)$	—	—	—
q_4	$(q_4, 0, R)$	—	—	—	$(q_5, 0, L)$
q_5	$(q_5, 0, L)$	$(q_6, \$, L)$	—	—	—
q_6	$(q_6, 0, L)$	—	—	(q_2, y, R)	—
q_7	—	$(q_8, \$, L)$	—	$(q_7, 0, L)$	—
q_8	$(q_8, 0, L)$	—	(q_0, x, R)	—	—
q_9	—	—	$(q_9, 0, L)$	—	$(q_f, \#, R)$
$*q_f$	—	—		—	—

Example 3:

Design a TM that copies strings of 1's.

For this problem, let the given string is 11 as shown below:

#	1	1	#	#	-

Input Tape

Note that we store a # before the string in the tape.

The output from TM should be as follows:

#	1	1	1	1	#	#	-

Input Tape

Following is the transition table for this TM:

	Input Symbol		
Current State	1	#	x
$\longrightarrow q_0$	(q_0, x, R)	$(q_1, \#, L)$	—
q_1	$(q_1, 1, L)$	$(q_3, \#, R)$	$(q_2, 1, R)$
q_2	$(q_2, 1, R)$	$(q_1, 1, L)$	—
$*q_3$	—	—	—

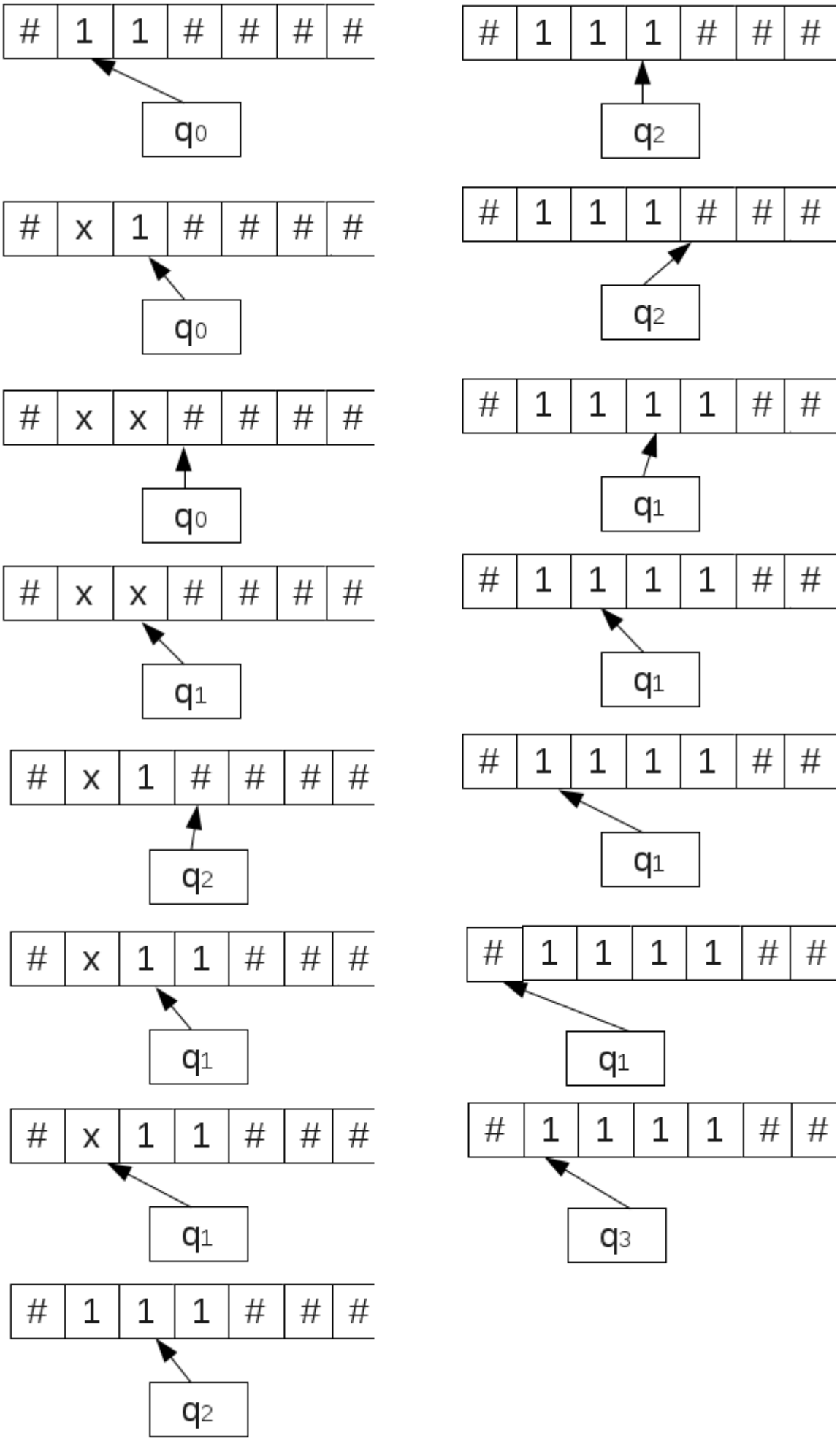
Here the TM replaces every 1 by symbol x. Then TM replaces rightmost x by 1. It goes to the right end of the string and writes a 1 there. Thus TM has added a 1 for the rightmost 1 in the input string. This process is repeated.

TM reaches q_1 after replacing all 1's by x's and reading the # symbol at the end of the input string. After replacing x by 1, TM reaches q_2. TM reaches q_3 at the end of the process and halts.

If the string is 11, we get 1111 at the end of computation.

If the string is 111, we get 111111 at the end of computation.

Consider the processing of the string 111:


1 1 # # #
q0
x 1 # # #
q0
x x # # #
q0
x x # # #
q1
x 1 # # #
q2
x 1 1 # #
q1
x 1 1 # #
q1
1 1 1 # #
q2
1 1 1 # #
q2
1 1 1 # #
q2
1 1 1 1 #
q1
1 1 1 1 #
q1
1 1 1 1 #
q1
1 1 1 1 #
q1
1 1 1 1 #
q3

TM as a Computer

Thus a TM can perform the basic operations such as addition, multiplication, subtraction and division. That means it can act as a computer. TM is a simple mathematical model of a computer. TM can do everything a computer can do. If TM cannot solve certain problems, then these problems are beyond the theoretical limits of computation.

Exercises:

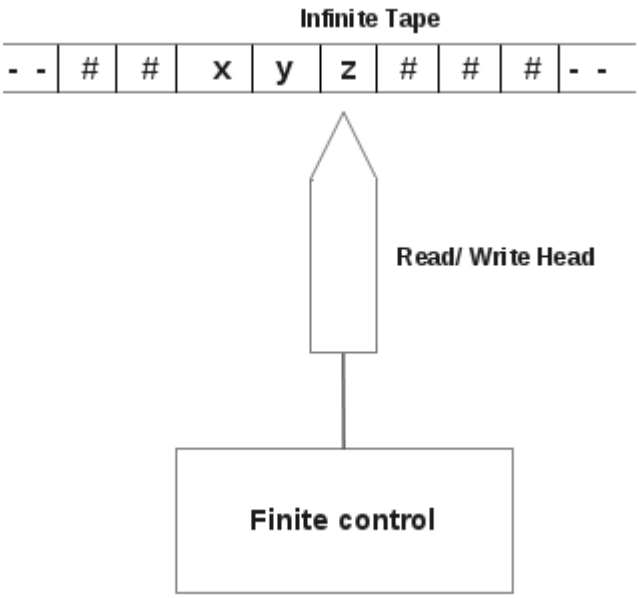
1. Design a Turing machine that computes m - n where m and n are positive integers and m>n.
2. Design a TM to divide m by 3 and to compute the quotient and the remainder.

Part IV. Types of TM

5 Two Way Infinite Turing Machines

The TM we discussed so far had a tape that was finite on the left end and infinite on the right end.

Two way infinite TM is an extension to this such that tape is infinite at both ends. This is shown below:



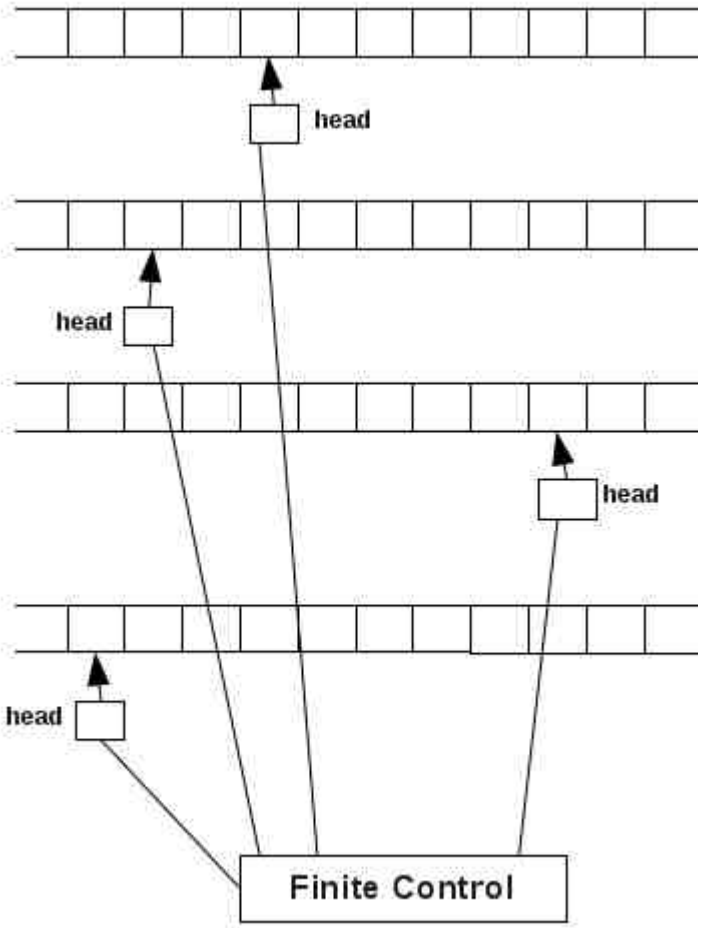
Thus on both sides of the tape, there is an infinite sequence of blank symbols (#).

This model does not provide any additional computational capability.

6 Multitape Turing Machines

A multitape TM consists of multiple tapes. Each tape has a separate head.

This is shown below:



There are n tapes, each divided into cells. The first tape holds the input string. Initially, all the other tapes hold the blank symbol, #.

A head has three possible options:

to move towards left (L),

to move towards right (R), or

to remain stationary (N).

All the heads are connected to a finite control. Finite control is in a state at an instant.

1. Initially, finite control is in state q_0.
2. Head in the lowermost tape points to the cell containing leftmost symbol of the input string.
3. All the cells in the upper tapes contain # symbol.

When a transition occurs,

1. Finite control may change its state.
2. Head reads the symbol from the current cell and writes a symbol on it.
3. Each head can move towards left (L), right (R) or stay stationary (N).

An example transition function for a 4 tape TM is given below:

$\delta(q_1, [a_1, a_2, a_3, a_4]) = (q_2, [b_1, b_2, b_3, b_4], [L, R, R, N])$

Here the finite control is in state q_1, It reads the symbol $a_1$from the uppermost tape, $a_2$from the next uppermost tape

and so on.

After reading, finite control changes its state to q_2, and replaces the symbol $a_1$by b_1, $a_2$by b_2, $a_3$by b_3, $a_4$by b_4.

After this head in the upper most tape turns left. Head in the 2nd tape turns right. Head in the 3rd tape turns right. Head in the 4th tape remains stationary.

The advantage of using multi tape TM can be seen from the following example.

Example:

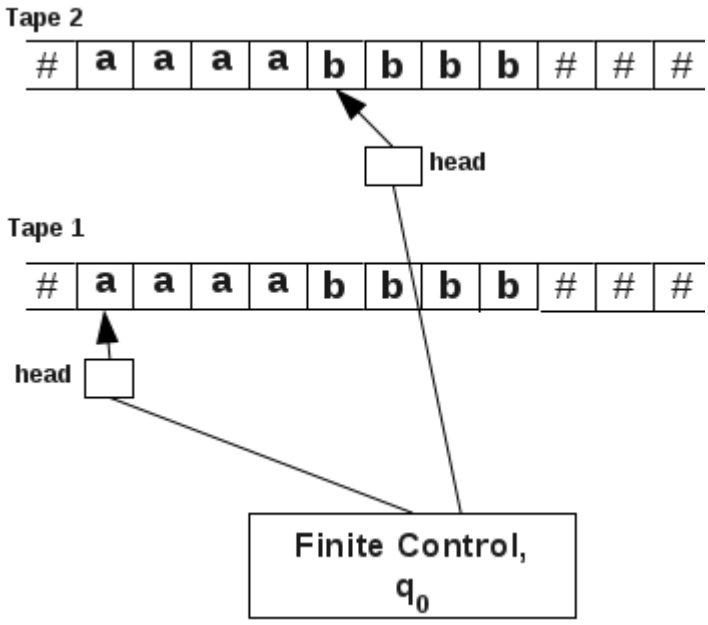
Consider the language $L = \{a^n b^n | n \geq 1\}$.

In a normal TM, head has to move back and forth to match each pair of symbols a and b. On a multitape TM, no such movements are needed.

This is done by making a copy of the input string in another tape.

Let the input string be aaaabbbb.

Let the input srtring be in tape 1. Make a copy of this string in tape 2.



The head of tape 1 is positioned on the first a of the input string. Head of tape 2 is positioned on the first b of the input string.

Now the heads advance on both tapes simultaneously towards right and the string is accepted if there are equal number of a's and b's in the string.

This will happen if the head on tape 1 encounters the first b and the head on tape 2 encounters the first # simultaneously.

7 Universal Turing Machines

In the previous sections, a separate TM was designed for each language.

For example, for the language, $L = \{a^n b^n | n \geq 1\}$, we designed a TM.

For the language, $L = \{a^n b^n c^n | n \geq 1\}$, we designed another TM and so on.

A Universal turing Machine (UTM) takes the code of a normal TM and a string w, and checks whether w is recognised by TM.

This is done as follows:

Consider a TM defined as,

$$TM = (Q, \sum, \Gamma, \delta, q_0, \#, q_f)$$

where

$$Q = \{q_0, q_1, q_2, q_f\}$$

$$\sum = \{a, b\}$$

$$\Gamma = \{a, b, x, y, \#\}$$

δ is defined as,

$$\delta(q_0, a) = (q_1, b, R)$$

$$\delta(q_0, x) = (q_1, y, L)$$

$$\delta(q_1, x) = (q_2, y, R)$$

$$\delta(q_2, y) = (q_f, \#, R)$$

$$\delta(q_f, x) = (q_0, y, L)$$

The code for this Tm is to be made.

First we encode all the components of this TM using binary coding. In binary coding only symbols 0 and 1 are available. Here 0 is used to code all the transition functions and 1 is used as a separator.

States of the TM are,

$$Q = \{q_0, q_1, q_2, q_f\}$$

The states can be coded as,

$$q_0 = 0$$

$$q_1 = 00$$

$$q_2 = 000$$

$$q_f = 0000$$

Tape symbols are,

$$\Gamma = \{a, b, x, y, \#\}$$

Tape symbols can be coded as,

$$a = 0$$

$$b = 00$$

$$x = 000$$

$$y = 0000$$

$$\# = 00000$$

Direction of motion is coded as follows:

$$L = 0$$

$$R = 00$$

Now the transition functions are coded as follows:

$$\delta(q_0, a) = (q_1, b, R) \qquad \Longrightarrow 010100100100$$

$$\delta(q_0, x) = (q_1, y, L) \quad \Longrightarrow 010001001000010$$

$$\delta(q_1, x) = (q_2, y, R) \quad \Longrightarrow 001000100010000100$$

$$\delta(q_2, y) = (q_f, \#, R) \quad \Longrightarrow 0001000010000100000100$$

$$\delta(q_f, x) = (q_0, y, L) \quad \Longrightarrow 00001000101000010$$

In the coding of TM, coding ends with 111. Transition functions are separated using 11.

Total coding of the TM involves coding of the transition functions.

Total coding of the TM is,

$$0101001001001101000100100001011001000100010000100110001000010000100000100110000100010100001011 1$$

Let w=abbb be the string to be checked on the Turing machine, M. The input to UTM will be,

$$0101001001001101000100100001011001000100010000100110001000010000100000100110000100010100001011 1abbb$$

UTM uses the binary code of the turing machine, M on string abbb and will check if abbb is recognised by M. If true UTM, will halt to say yes and if false UTM will stop to say no.

UTM and Modern Computer

Thus UTM is a generic machine that is capable of implementing the code of any arbitrary TM. This concept of UTM led to the modern day computer.

The concept of UTM is similar to the behaviour of modern day computer. Our modern computer system can choose a program for a problem and solve it on the required data.

Part V. Church's Thesis

8 Church's Thesis

It is also called Church- Turing thesis.

Church's thesis states that

A function is said to be computable if it can be computed by a Turing machine.

This thesis talks about mechanical computation devices and the kind of computations they can perform.

Some more definitions of this thesis are given below:

1. Any mechanical computation can be performed by a Turing machine.

2. For every computational problem, there is a corresponding Turing machine.

3. Turing machines can be used to model any mechanical computer.

4. The set of languages that can be decided by a TM is the same as that which can be decided by any mechanical computing machine.

5. If there is no TM that decides problem P, there is no algorithm that can solve problem P.

Part VI. Godelization

9 Godelization

Godelization is an encoding technique which encodes a string as a number. This is called as Godel numbering.

Godel numbering is based on the concept that every positive integer can be factored into a unique set of prime factors. For example,

6 = 2 x 3

8 = 2 x 2 x 2

9 = 3 x 3

10 = 2 x 5

20 = 2 x 2 x 5

30 = 2 x 3 x 5

50 = 2 x 5 x 5

100 = 2 x 2 x 5 x 5

5,71,725 = 3 x 3 x 3 x 5 x 5 x 7 x 11 x 11

Also, it is possible to assign a serial number to each prime number, as

1 to 2

2 to 3

3 to 5

4 to 7

5 to 11, and so on.

Now the number,

5,71,725 = 3 x 3 x 3 x 5 x 5 x 7 x 11 x 11 = $2^0 \times 3^3 \times 5^2 \times 7^1 \times 11^2$

can be represented in the form of a sequence (0, 3, 2, 1, 2).

In terms of Godel numbering, we say that the Godel number associated with the sequence (0, 3, 2, 1, 2) is 5,71,725.

Thus the formal definition of Godel numbering is,

For any finite sequence $(x_0, x_1, x_2, ...x_n)$ the associated Godel number, G_n is,

$$G_n(x_0, x_1, x_2, ...x_n) = 2^{x_0}3^{x_1}5^{x_2}7^{x_3}.....P_n^{x_n},$$

where P_n is the nth prime number.

The importance of Godel numbering lies in the fact that each Godel number encodes a unique sequence and a sequence encodes a unique Godel number.

This allows us to represent different configurations of a multi parameter dependent object in the form of a unique number.

A TM at any instant, can exist in different configurations, with each configuration being defined by the current state and position of the head. The position of the head can be defined by a cell number. It is possible to number each state and

each cell, and to encode every configuration of a TM in the form of a Godel number. Also, it is possible to define every transition function of a TM in the form of a Godel number.

Part VII. Time Complexity of Turing Machine

10 Time Complexity of Turing Machine

A Turing machine takes a string w, it gives the output in the form yes or no. Here it is possible to exactly measure the number of moves made by the head of the TM during the processing of the string.

Although the number of moves made by the TM to process the string depend on the specific string, it is possible to make an estimate of the maximum possible number of moves that TM will make to accept or reject a string of length of n.

Time complexity of a Turing machine, M is written as,

$\tau_M(n)$

where τ is the time complexity,

n is a natural number.

Time complexity of a TM is the maximum number of moves made by the head to accept or reject a string of length n in the worst case scenario.

Example 1:

Consider the TM to accept the language, $L = \{ww^R | w \in (a,b)^+\}$

Find the time complexity of the TM.

Let the string is of length n.

Here the TM works as follows:

It checks the first symbol of the input string, converts it to #, and travels the whole of the string and reaches the # just following the string. It turns back and matches the last symbol of the string with the first one. If the match occurs, this last symbol is replaced with a #. Next the head travels back to the second symbol of the input string.

Here the number of moves required is 2n + 1.

On reaching the second symbol, it finds its corresponding match.

Here the number of moves required is 2n - 3.

When al lthe pairs are successfully matched, TM just makes one move and halts.

The worst case is one in which the input string is valid and the TM has to match all the corresponding pairs. Also, with every match, number of moves required to match the next symbol decreases. the process stops when all the symbols are successfully matched and no move is required. Now the TM halts.

Total number of moves made by the TM is,

(2n+1) + (2n-3) + (2n-7) + ... + 1

= ((2n+1) + 1) /2 x (n/2 +1)

= (n+1) x (n/2 +1)

=$n^2/2 + 3n/2 + 1$

Thus the time complexity of TM is $\mathcal{O}(n^2)$

$\tau(n) = \mathcal{O}(n^2)$

Example 2:

Consider the TM to accept the language, $L = \{a^n b^n | n \geq 1\}$

Find the time complexity of the TM.

Here head replaces leftmost 0 by x and leftmost 1 by y. Then head reaches back to the second symbol. Here 2n moves are needed.

Thus at most 2n moves are needed to match a 0 with a 1.

For matching all 0s with 1s, this process is to be repeated n/2 times.

Then the total number of moves needed is around 2n x n/2. That is $O(n^2)$

Time complexity of this TM is,

$$\tau(n) = \mathcal{O}(n^2)$$

Part VIII. Halting Problem of TM

Here we use a process called reduction.

Let A is the problem of finding some root of $x^4 - 3x^2 + 2 = 0$, and

B is the problem of finding some root of $x^2 - 2 = 0$.

Here $x^2 - 2$ is a factor of $x^4 - 3x^2 + 2 = 0$.

Thus a root of $x^2 - 2 = 0$ will also be a root of $x^4 - 3x^2 + 2 = 0$.

Then we can say that A is reducible to B.

Thus a problem A is reducible to problem B if a solution to problem B can be used to solve problem A.

Then if A is reducible to B and B is decidable, then A is decidable.

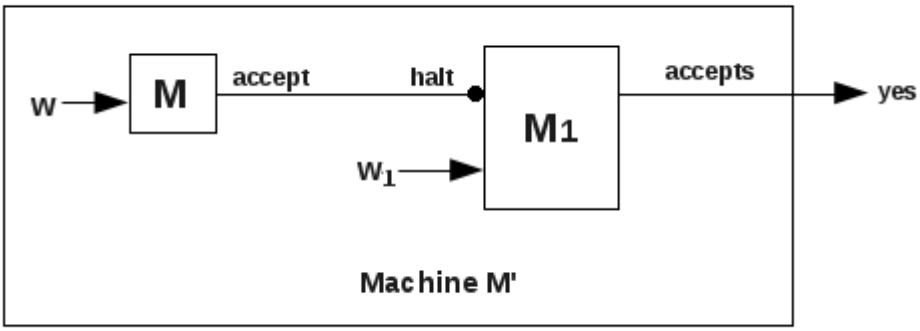
If A is reducible to B and A is undecidable, then B is undecidable.

11 Halting Problem of TM

This problem states that halting problem of a Turing machine is undecidable.

Proof

To show this, we reduce problem of halting to a problem of acceptance. Consider the following TM,



Here we take an instance (M, w) and construct another instance (M_1, w_1).

It is taken such that M_1 halts on input w_1, iff M accepts w.

The machine M' stops when M_1 halts.

Initially, the string w is fed to the TM, M and w_1 is fed to the TM, M_1.

If M accepts w, then it sends a halt signal to M_1. Then TM, M_1 halts on input w_1.

If M rejects w, then M_1 does not halt on w_1.

Thus halting of M_1 depends on the acceptance behaviour of M. Acceptance behaviour of a TM is undecidable, halting of M_1 or M' is undecidable.

Part IX. Rice Theorem

Recursive and Non-Recursive Languages

Recursive Languages

A language, L is recursive, if

it is possible to design a TM that halts in the final state to say yes if w∈L, and

halts in the non-final state to say no if w $\notin$ L.

Recursively Enumerable Languages

A language, L is recursively enumerable, if

it is possible to design a TM that halts in the final state to say yes if w∈L, and

halting cannot be guranteed if w $\notin$ L.

Languages that are neither Recursive nor Recursively Enumerable

A language, L is neither recursive nor recursively enumerable, if

the structure of the language is such that no TM which recognises w can be designed.

12 Rice Theorem

Rice's theorem states that

every non-trivial property of a recursively enumerable language is undecidable.

Proof

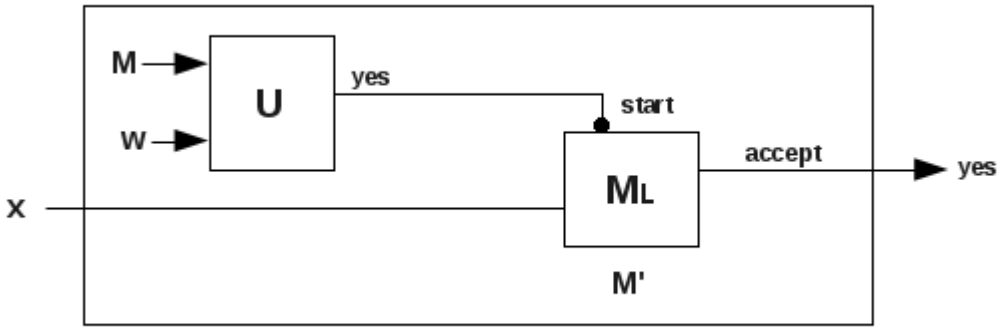
A non-trivial property is one that is possessed by some objects of a class, but not all.

For example, being a mathematician is a property that is possessed by some humans but not by all.

Some cats are black but not all. So black colour property cannot be trivially associated with cats.

Let χ be a non-trivial property that is not possessed by all recursively enumerable languages. This problem can be reduced to one consisting of a pair (M, w) such that L possesses χ iff $w \in L(M)$. We take a UTM U that takes a pair (M, w); its output is yes iff χ is possessed by L.

Since L is a recursively enumerable language, there must be a TM, M_L that accepts L. Let x be a string belonging to L. Now, we design a machine M' to decide χ as shown below:



Here U is a UTM.

UTM, U takes the pair (M, w) and checks if $w \in L(M)$. If the output is yes, then the machine $M_L$that accepts the string x starts and the output of the machine M' is yes. Thus the decidability of the problem of possessing the trivial property reduces to the problem of L_u. If the pair $(M, w) \in L_u$, then L possesses χ ; otherwise not. Since L_u is not recursive, possession of χ by L is also not decidable.

Part X. Post Correspondence Problem

13 Post Correspondence Problem

Post correspondence problem (PCP) is a problem formulated by E. Post in 1940.

Let us illustrate this problem by an example:

Example 1:

Let there be two series of strings, say x series and y series as shown below:

i	x_i	y_i
1	10	101
2	01	100
3	0	10
4	100	0
5	1	010

Both X series and Y series contains 5 strings.

Let us call the strings in X series as X substrings, and

the strings in Y series as Y substrings.

If we concatenate X substrings $x_1x_5x_2x_3x_4x_4x_3x_4$, we get 1010101001000100.

If we concatenate Y substrings $y_1y_5y_2y_3y_4y_4y_3y_4$, we get 1010101001000100.

Let us call these strings as x string and y string.

It can be seen that x string and y string are same.

Here we say that this instance of PCP has a solution in the form 15234434.

If we take 23,

x string is, x_2x_3 =010

y string is, y_2y_3 =10010.

Here x string and ys tring are different. Hence 23 is not a solution to this instance of PCP.

Example 2:

Find the solution to the instance of PCP given in the following table.

i	x_i	y_i
1	0	000
2	01000	01
3	01	1

Solution,

$x_2x_1x_1x_3$= 010000001

$y_2y_1y_1y_3$= 010000001

Hence, 2113 is a solution to this instance of PCP.

Example 3:

Find why the instance of PCP given below cannot have a solution.

i	x_i	y_i
1	0	000
2	010	0100
3	01	100
4	11	110

It can be seen that for every pair, $|x_i| > |y_i|$.

So, in whatever way we concatenate the string, the length of the x string will be longer than the corresponding y string. Thsu there is no solution for this instance of PCP.

Definition of PCP

Let there be two series, x series and y series of size n with same charactere set $\sum$ with their i^{th} element as x_i and y_i, respectively; does there exist a solution tha tforms the same x series and y series?

The generic solution of PCP can be written as,

$$x_{i_1}x_{i_2}x_{i_3}.....x_{i_k} = y_{i_1}y_{i_2}y_{i_3}.....y_{i_k}$$

Post Correspondence Problem is unsolvable.

This means it is a non computable function. No turing machine exists for PCP.

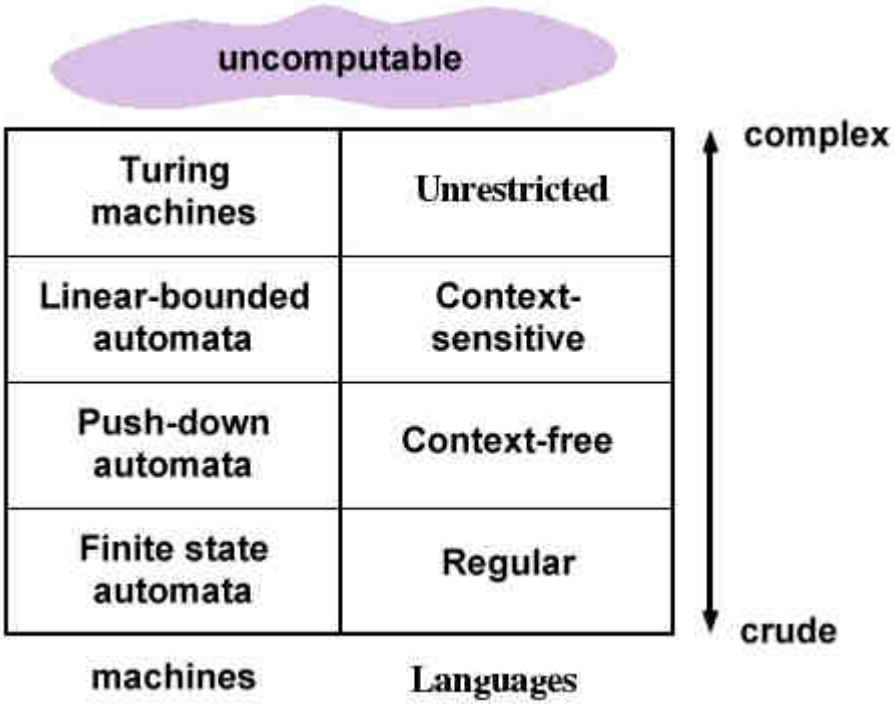
The proof of this is beyond the scope of our study.

Use of PCP

Other problems can be reduced to PCP and we can declare them as unsolvable or undecidable.

Part XI. Linear Bounded Automata

Consider the following figure:



We learned in Chomsky classification that context sensitive languages (Type-1) are produced from context sensitive grammars.

Context sensitive languages are recognised using linear bounded automata.

The following is an example for a Context Sensitive grammar:

S $\longrightarrow$ aBCT|aBC

T $\longrightarrow$ ABCT|ABC

BA $\longrightarrow$ AB

CA $\longrightarrow$ AC

CB $\longrightarrow$ BC

aA $\longrightarrow$ aa

aB $\longrightarrow$ ab

bB $\longrightarrow$ bb

bC $\longrightarrow$ bc

cC $\longrightarrow$ cc

in this grammar, LHS of a production is not longer than the RHS.

The language, $L = \{a^n b^n c^n | n > 0\}$ is a context sensitive language.

14 Linear Bounded Automata (LBA)

Context sensitive languages are recognised using linear bounded automata (LBA).

Here input tape is restricted in size. A linear function is used for restricting the length of the input tape.

Many compiler languages lie between context sensitive and context free languages.

A linear bounded automation is a non deterministic Turing machine which has a single tape whose length is not infinite but bounded by a linear function.

Formal Definition

A linear bounded automation is defined as,

$M = (Q, \sum, \Gamma, \delta, q_0, \#,$ <,>,F),

where

Q is a set of states,

$\sum$ is a set of input symbols,

Γ is a set of tape symbols, including #,

δ is the set of transition functions from

$$(Q \times \Gamma) \longrightarrow (Q \times \Gamma \times \{L, R, N\}),$$

q_0 is the start state,

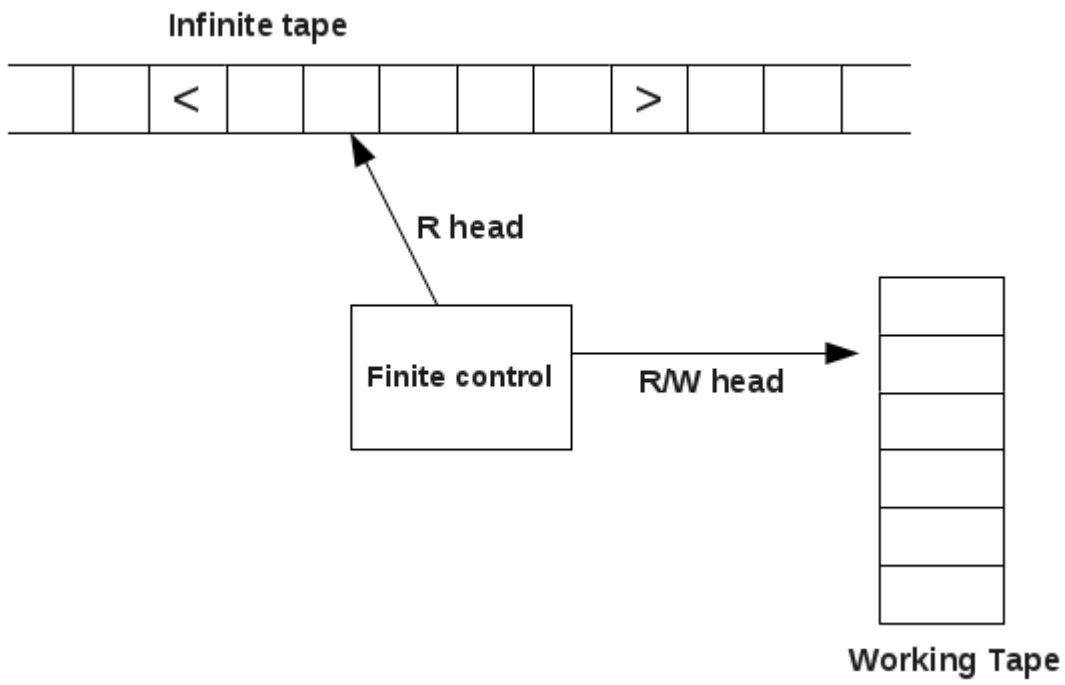
$\#$ is the blank symbol,

< is the left end marker in the input tape,

$>$ is the right end marker in the input tape,

F is the final state.

Following diagram shows a linear bounded automation,



When an input string is processed using an LBA, input string is enclosed between < and > end markers.

The end marker < prevents the R head from getting off the left end of the tape. The right end marker > prevents the R head from getting off the right end of the tape.

On the input tape, head does not write. Also on the input tape, head does not move left. All the computation is to be done between end markers < and >.

On the working tape head can read and write without any restriction.

Thus LBA is same as a Turing machine except that head can move only within the end markers.

Example:

Consider an LBA defined as,

Q= {q_0, q_1, q_2, q_3, q_4}

$\sum$= {a, b, c}

Γ= {a, b, c, #}

q_0= s

δ(q_0, [) = (q_1, [, R)	δ(q_1,]) = (q_1,],Y)	δ(q_1, #) = (q_1, #, R)
δ(q_1, a) = (q_2, #, R)	δ(q_2, a) = (q_2, a, R)	δ(q_2, #) = (q_2, #, R)
δ(q_2, b) = (q_3, #, R)	δ(q_3, b) = (q_3, b, R)	δ(q_3, #) = (q_3, #, R)
δ(q_3, c) = (q_4, #, L)	δ(q_4, c) = (q_4, c, L)	δ(q_4, b) = (q_4, b, L)
δ(q_4, a) = (q_4, a, L)	δ(q_4, #) = (q_4, #, L)	δ(q_4, [) = (q_1, [, R)

In the above, the transition $\delta(q_1, a) = (q_2, \#, R)$ means,

LBA on state q_3, head points to symbol b, remains in state q_3, replaces b with b and head turns towards right by one cell.

Questions (from Old syllabus of S7CS TOC)

MGU/Nov2011

1. Explain the instantaneous description of a Turing machine (4marks).

2. What is a universal turing machine (4marks)?

3a. Design a Turing machine that computes a function f(m,n)=m+n in addition of two integers.

OR

b. Explain the halting problem of Turing machine. Prove that it is undecidable (12marks).

MGU/April2011

1. What are the languages accepted by a Turing machine (4marks)?

2. Define Godelization (4marks).

3a.

OR

b. Design a Turing machine with the initial tape as 01110111110... and the output pattern 01111101110... (12marks).

4a. Show that the halting problem of a Turing machine is undecidable. Explain Godelization with example.

OR

b. Design a m-tape Turing machine that works as a copying machine (12marks).

MGU/Nov2010

1. What is meant by halting of a TM (4marks).

2a. Design a Turing machine to compute a function f where, $f : \sum_0^* \longrightarrow \sum_0^*, \sum_0 = \sum_1 = \{a, b\}, f(w) = \overline{w}, \overline{w}\, si$ the result of replacing an occurrence of a in wbyb and vice versa.

OR

b. Explain Church's thesis and its application. Also explain Godelization (12marks).

3a. Show that halting problem of a Turing machine is not NP-complete (12marks).

OR

b.

MGU/May2010

1a. What is configuration of a Turing machine ?

b. When do we say that a function is Turing computable (4marks)?

2. Explain Church's thesis (4marks).

3a.

Or

b. i. Construct a Turing machine to do the multiplication.

ii. Design a Turing machine to compute n mod 2. (12marks).

MGU/Nov2009

1. a.

b. State Post correspondence problem (4marks).

2. Describe the method of Godelization (4marks).

3a.

OR

b. Prove the equivalence of two way infinite tape with standard turing machine (12marks).

4a.

OR

b. i. Describe the action of a turing machine.

ii. Explain briefly how to enumerate all possible turing machines computations, so that a given computation can be characterised by a single natural number code C. (12marks).

5a. Show that it is not possible to compute the maximum distance travelled by the turing machine head from its initial position during halting computations as a function of code C. Any results that you use should be stated clearly (12marks).

Or

b.

MGU/Nov2008

1. What is Turing machine (4marks)?

2a. Design a Turing machine that recognise the language $L = \{ww^R|w$ is in $(a+b)^*\}$.

OR

b. What do you mean by a universal turing machine? Explain its applications (12marks).

MGU/May2008

1. Explain Church's thesis (4marks).

2. What is multi-head turing machine (4marks)?

3a. Construct a turing machine that accepts the language given by $\{ww^R|w$ is in (0+1)}.

OR

b. Explain universal turing machine and explain its applications (12marks).

MGU/Dec2007

1. What is halting problem of turing machine (4marks)?

2. What is turing computability (4marks)?

3a. Design a turing machine that recognises the language {w| w is in $(a+b)^*\}$.

OR

b. What is church's thesis and Godelization (12marks)?

MGU/July2007

1. Define turing machine (4marks).

2. Briefly explain church's thesis (4marks).

3a. i. Construct a turing machine that increments a binary number.

ii. Write short notes on any two variants of turing machines (12marks).

OR

b. i. Construct a Turing machine that decrements a binary number.

ii. Construct a turing machine that adds two unary numbers (12marks).

4a. State turing machine halting problem. Show that it is undecidable (12marks).

OR

b.

MGU/Jan2007

1. What is a universal turing machine (4marks)?

2. Construct a turing machine which computes the function f(n)=n+2 over unary numbers (4marks).

3a. i. Construct a turing machine that decides the language $L = \{a^n b^n | n \geq 0\}$.

ii. Construct a turing machine that shifts the input string one position to the left.

OR

b. i.

ii. Construct a turing machine that accepts the language $L = \{w \in (a, b)^* | w$ has equal number of a's and b's} (12marks).

MGU/July2006

1. Write short notes on halting problem of turing machine (4marks).

2. Explain church's hypothesis (4marks).

3a. Design a turing machine to recognise the language $L = \{0^n 1^n 0^n | n \geq 1\}$.

OR

b. i. Construct a turing machine that will compute f(x,y)=x+y.

ii. Write a note on universal turing machines (12marks).

MGU/Nov2005

1. What is Church's hypothesis? Explain (4marks).

2. Define a turing machine (4marks).

3a. Prove that a language L is recognised by a Turing machine with a two way infinite tape iff it is recognised by a turing machine with a one way infinite tape.

OR

b. Design a turing machine M to recognise the language $L = \{ww^R | w$ is in $(a + b)^*$ (12marks).

References

.

Nagpal, C, K (2011). Formal Languages and Automata Theory. Oxford University Press.

Pandey, A, K (2006). An Introduction to automata Theory and Formal Languages. Kataria & Sons.

Mishra, K, L, P; Chandrasekaran, N (2009). Theory of Computer Science. PHI.

Linz, P (2006). An Introduction to Formal Languages and Automata. Narosa.

Hopcroft, J, E; Motwani, J; Ullman, J, D (2002). Introduction to Automata Theory, Languages and Computation. Pearson Education.

website: http://sites.google.com/site/sjcetcssz

St. Joseph's College of Engineering & Technology Palai

Department of Computer Science & Engineering

S4 CS

CS010 406 Theory of Computation

Module 5

Theory of Computation - Module 5

Syllabus

Complexity classes- Tractable problems– Class P –P Complete-Reduction problem- Context grammar nonempty-

Intractable problems- Class NP – NP Complete- Cooks theorem-

Reduction problems-SAT-Clique-Hamiltonian-TSP-Vertex Cover-NP Hard problems.

Contents

Part I. Complexity

In this module, we will learn different complexity classes, such as P and NP.

In data structures, we already learned about time complexity and space complexity. Here we will be concerned about time complexity of various problems.

We learned computable and non computable functions.

A computable function is a function for which we can construct a Turing machine. Or an algorithm can be formulated.

A non computable function is one in which no Turing machine can be constructed. Here we cannot devise an algorithm.

1 Tractable and Intractable Problems

Consider computable functions. These functions are computable or solvable. We can construct a TM for them. We can devise an algorithm for them.

But many of these problems can be solved only in principle, not in practice. This is because some of the computable functions may take 1000s of years to find a solution using a computer system. Such problems are termed intractable problems. They come under complexity class NP.

Most of the problems we are familiar with can be solved within a reasonable amount of time. Such problems are said to be tractable. They come under complexity class P.

Part II. Complexity Class P

Consider some of the problems we learned in data structures, such as bubble sort, quick sort, merge sort, binary search, matrix multiplication etc..

Also we learrned that,

The time complexity of bubble sort is $\Theta(n^2)$.

The time complexity of quick sort is $\Theta(n\, log\, n)$.

The time complexity of merge sort is $\Theta(n\, log\, n)$.

The time complexity of heap sort is $\Theta(n\, log\, n)$.

The time complexity of binary search is $\Theta(log\, n)$.

The time complexity of matrix multiplication algorithm is $\Theta(n^3)$.

Consider time complexity values of all these algorithms. They are of the order of $n^2, n\, log\, n, log\, n, n^3$ etc.. If we calculate exact value, we get polynomials such as $5n^2 + 2n + 4,\ n\, log\, n + 5n + 9,\ 7n^3 + 3n^2 + 9n + 3$ etc.. They are polynomials. This means that above algorithms are polynomial time algorithms.

These polynomial time algorithms can be solved within a reasonable amount of time. This means these problems can be solved in practice.

Consider the problem, sorting using bubble sort. The time complexity of bubble sort is $\Theta(n^2)$.

This means if there are 10 numbers in the list, a machine will take $\Theta(10^2)$ time to sort. If there are 100 numbers in the list, a machine will take $\Theta(100^2)$ time to sort. If there are 1000 numbers in the list, a machine will take $\Theta(1000^2)$ time to sort.

All these time values are reasonable. A computing machine can solve this sorting problem within a small amount of time.

Almost all the algorithms we learned have time complexity values n^3, n^2, $n\,log\,n$, n, $log\,n$, etc.. That is, for most of these algorithms, the exponent of n is at most 3.

These problems can be solved within a reasonable amount of time by a computing machine or a TM. These problems come in complexity class P.

But if an algorithm for a problem has time complexity $\Theta(n^{1000})$, it is not a reasonable amount of time. This is also a polynomial time. But for such a problem, it has observed that somebody will invent a new algorithm that has time complexity of the order of a small value of exponent such as n^4 or n^3.

2 Complexity Class P

The class P consists of those problems that are solvable in polynomial time.

This means these problems can be solved in time $\Theta(n^k)$, where

n is the size of the input,

k is a constant.

Another defintion is,

A problem is in class P, if there exists a deterministic Turing Machine of polynomial time complexity.

Examples for problems in class P are sorting using bubble sort, quick sort, heap sort etc.. ; searching using binary search, sequential search, ... ; matrix multiplication algorithm etc..

Most of the problems we are familiar with come under class P.

P - Hard Problems

A problem A, is said to be P-hard if,

every P problem can be reduced to A.

P - Complete Problems

A problem A, is said to be P-complete if,

A is P, and

A is P-hard.

Examples for P-complete problems :

Emptiness problem for context free grammars.

Circuit Value Problem (CVP) - Given a circuit, the inputs to the circuit, and one gate in the circuit, calculate the output of that gate

Linear programming - Maximize a linear function subject to linear inequality constraints

Lexicographically First Depth First Search Ordering - Given a graph with fixed ordered adjacency lists, and nodes u and v, is vertex u visited before vertex v in a depth-first search induced by the order of the adjacency lists?

Context Free Grammar Membership - Given a context-free grammar and a string, can that string be generated by that grammar?

Horn-satisfiability: given a set of Horn clauses, is there a variable assignment which satisfies them? This is P's version of the boolean satisfiability problem.

LZW Data Compression - given strings s and t, will compressing s with an LZ78 method add t to the dictionary? (Note that for LZ77 compression such as gzip, this is much easier, as the problem reduces to "Is t in s?".)

3 Emptiness of CFG

Emptiness Problem for Context Free Grammars

The emptiness problem is whether the grammar generates any terminal strings at all.

Emptiness Problem for CFGs is P - Complete

Theorem

The emptiness problem for context-free grammars is P-complete.

Proof

Consider any context-free grammar G $=\{V, \sum, P, S\}$. The emptiness of L(G) can be determined by the following algorithm.

Step 1:

Prove that the problem is P.

Mark each of the terminal symbols in $\sum$.

Search P for a production $A \longrightarrow \alpha$, in which α consists only of marked symbols and A is unmarked. If such a production rule $A \longrightarrow \alpha$ exists, then mark A and repeat the process.

If the start symbol S is unmarked, then declare L(G) to be empty. Otherwise, declare L(G) to be nonempty.

The number of iterations of Step 2 is bounded above by the number of nonterminal symbols in N. Consequently, the algorithm requires polynomial time and the problem is in P.

Step 2:

Prove that the problem is P-hard

To show that the emptiness problem for context-free grammars is P-hard, consider any problem K in P.

[This part of the proof is beyond the scope of this class.]

Part III. Complexity Class NP

Complexity class NP consists of following types of problems:

NP problems,

NP-hard problems,

NP-complete problems.

Consider computable functions. For these functions, a TM exists or they are solvable.

But some of the computable functions can be solved only in principle, not in practice. This is because a computing machine may take 1000s of years to solve such problems.

For these problems, a polynomial time algorithm has not yet been invented. We may wish somebody will invent a polynomial time algorithm for these problems in the future.

For these problems, time complexity is found to be $\Theta(2^n)$. This is not polynomial time, it is superpolynomial time complexity.

When the value of n is 10, time complexity value of such a problem will be $\Theta(2^{10})$, which is a manageable number.

When the value of n is 100, time complexity value of such a problem will be $\Theta(2^{100})$. This value is greater than the number of molecules in the universe. This means, such problems cannot be solved by a a computer in practice (when n is large). They are solvable in principle only. These problems come under complexity class NP.

Some examples for such problems in NP are,

Circuit satisfiability problem,

Boolean Formula satisfiability problem (SAT),

3-CNF satisfiability problem,

Clique problem,

Vertex cover problem,

Hamiltonian cycle problem,

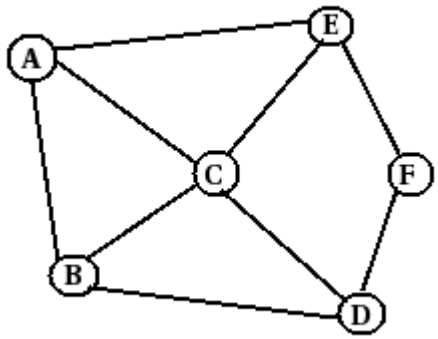
Traveling salesman problem (TSP).

4 Complexity Class NP

The class NP consists of those problems that are “verifiable” using a polynomial time algorithm.

What do you mean by “verifiable”.

This means if somebody gives a ’certificate’ of solution for such a problem, then we can verify that the certifcate is correct in polynomial time. For example, consider the hamiltonian cycle problem,



Let somebody gives us a certificate that a hamiltonian cycle, A-C-B-D-F-E-A exists in the above graph, it can be verified very easily in polynomial time. But if we are asked to find a hamiltonial cycle from the above graph, it cannot be solved in polynomial time.

Another definition is,

A problem is in class NP if there exists a non- deterministic Turing machine of polynomial time complexity.

All problems in P are also in NP. This is because all problems in P are verifiable in polynomial time. That is, $P \subseteq NP$.

Polynomial Time Reducibility

In some cases, a problem can be reduced to another problem.

Consider the problem of solving the linear equation, $bx + c = 0$.

We may transform this to the quadratic equation $0x^2 + bx + c = 0$. Solution of this quadratic equation is same as the solution of the given linear equation.

A problem A is reducible to another problem B, if it is possible to convert every instance of A to a corrsponding instance of B. If this reduction is possible in polynomial time, then we say that A is polynomial time reducible to B.

This is denoted as,

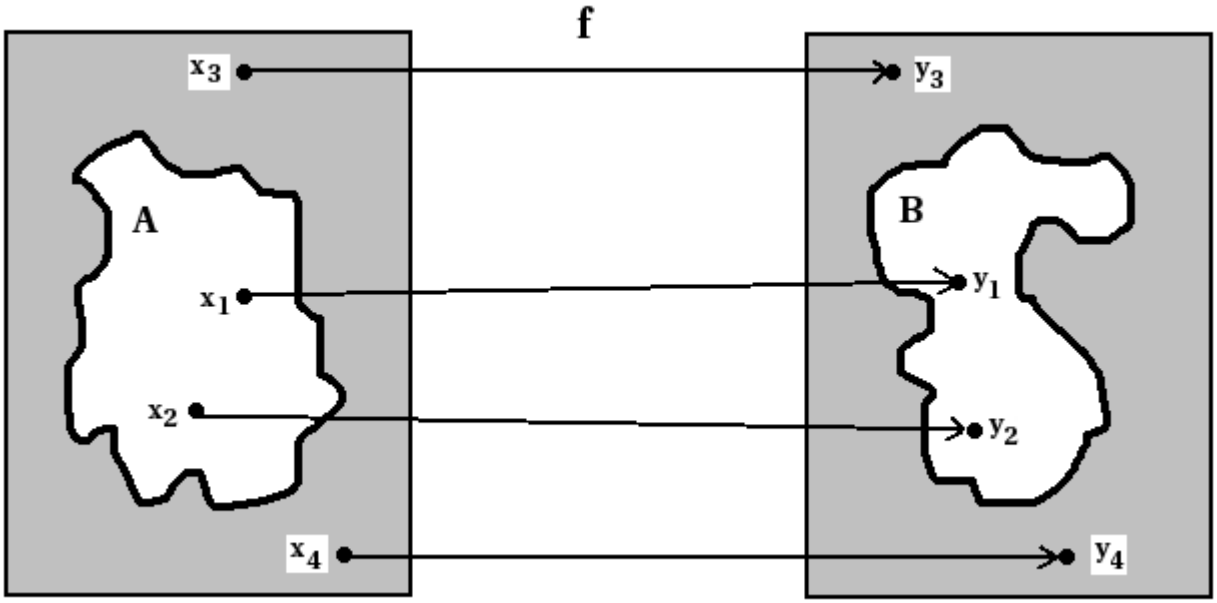
$$A \leq_p B.$$

This means A can be reduced to B in polynomial time.

If the algorithm used for reduction is f, then if $x \in A$, iff $f(x) \in B$, and

if $x \notin A$, iff $f(x) \notin B$.

This is shown below:



From the above figure, $x_1 \in A$,

$f(x_1) = y_1$,

$y_1 \in B$.

Also, $x_3 \notin A$,

$f(x_3) = y_3$,

$y_3 \notin B$.

5 NP - Hard Problems

A problem A, is said to be NP-hard if,

every NP problem can be reduced to A in polynomial time.

Let A be a problem. We say that A is NP-hard, if

$L \leq_p A$, for every $L \in NP$.

6 NP - Complete Problems

A problem A, is said to be NP-complete if,

A is NP, and

A is NP-hard.

Thus if we want to prove that a problem is NP complete, we need to prove that

it is NP, and

all NP problems can be reduced to this problem (NP-hard).

Examples for NP complete problems are,

Circuit satisfiability problem,

Boolean Formula satisfiability problem (SAT),

3-CNF satisfiability problem (3-CNF-SAT),

Clique problem,

Vertex cover problem,

Hamiltonian cycle problem,

Traveling salesman problem (TSP).

Proving that a problem is NP complete

We will prove that above problems are NP complete. The approach we use is as follows:

We will prove that Boolean Formula satisfiability problem (SAT) is NP complete.

This is done by proving that SAT is NP, and

All problems in NP are reduced to SAT.

To prove that 3-CNF-SAT is NP complete,

Prove that 3-CNF-SAT is NP.

SAT is reduced to 3-CNF-SAT. [All NP problems can be reduced to SAT. SAT can be reduced to 3-CNF-SAT. This means all NP problems can be reduced to 3-CNF-SAT.] This means 3-CNF-SAT is NP hard.

To prove that Clique problem is NP complete,

Prove that Clicque problem is NP.

3-CNF-SAT is reduced to Clique problem. [All NP problems can be reduced to SAT. SAT can be reduced to 3-CNF-SAT. 3-CNF-SAT can be reduced to Clique problem. This means all NP problems can be reduced to Clique problem.] This means Clique problem is NP hard.

To prove that vertex cover problem is NP complete,

Prove that vertex cover problem is NP.

Clique problem is reduced to vertex cover problem. [All NP problems can be reduced to SAT. SAT can be reduced to 3-CNF-SAT. 3-CNF-SAT can be reduced to Clique problem. Clique problem can be reduced to vertex cover problem. This means all NP problems can be reduced to vertex cover problem.] This means vertex cover problem is NP hard.

To prove that Hamiltonian cycle problem is NP complete,

Prove that Hamiltonian cycle problem is NP.

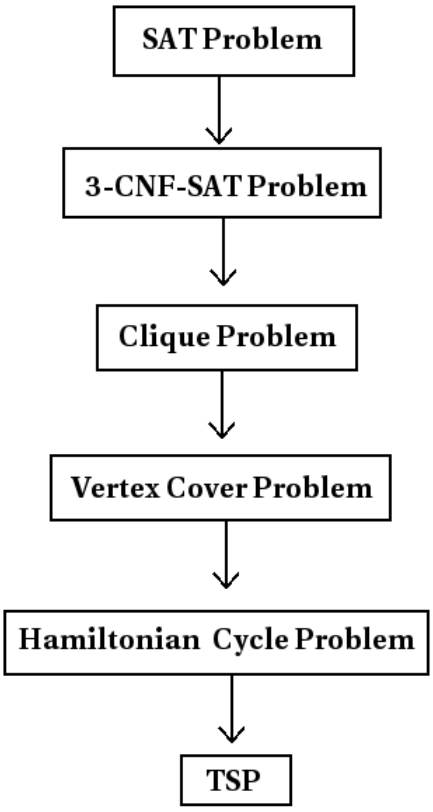
Vertex cover problem is reduced to Hamiltonian cycle problem. [All NP problems can be reduced to SAT. SAT can be reduced to 3-CNF-SAT. 3-CNF-SAT can be reduced to Clique problem. Clique problem can be reduced to vertex cover problem. Vertex cover problem can be reduced to Hamiltonian cycle problem. This means all NP problems can be reduced to Hamiltonian cycle problem.] This means Hamiltonian cycle problem is NP hard.

To prove that Traveling Salesman(TSP) problem is NP complete,

Prove that TSP is NP.

Hamiltonian cycle problem is reduced to TSP. [All NP problems can be reduced to SAT. SAT can be reduced to 3-CNF-SAT. 3-CNF-SAT can be reduced to Clique problem. Clique problem can be reduced to vertex cover problem. Vertex cover problem can be reduced to Hamiltonian cycle problem. Hamiltonian cycle problem can be reduced to TSP. This means all NP problems can be reduced to TSP.] This means TSP is NP hard.

Thus approach we use for proving a problem is NP-complete is shown below.



Part IV. SAT (Boolean Formula Satisfiability Problem)

7 SAT

Satisfiability

An example for a boolean formula is

$$(\neg x_1 \cup x_2) \cap (x_1 \cup \neg x_2) \cap (\neg x_2 \cup x_3) \cap (\neg x_1 \cup \neg x_2 \cup x_3).$$

This formula is in conjunctive normal form (CNF) (intersection of unions).

This boolean formula is satisfiable if there is some assignment of the values 0 and 1 to its variables that causes it to evaluate to 1 (true).

If we put, $x_1 = 0, x_2 = 0, x_3 = 1$, we can see that the above formula evaluates to 1 (true).

This means above formula is satisfiable.

Consider the boolean formula,

$$(x_1 \cap x_2) \cap (\neg x_1 \cup \neg x_2)$$

If we put $x_1 = 0, x_2 = 0$, this formula evaluates to 0. If we put $x_1 = 0, x_2 = 1$, this formula evaluates to 0. If we put $x_1 = 1, x_2 = 0$, this formula evaluates to 0. If we put $x_1 = 1, x_2 = 1$, this formula evaluates to 0. This means for any values assigned to the variables the above boolean formula always evaluate to 0 (false).

This means the formula is not satisfiable.

SAT Problem

SAT (Boolean Formula Satisfiability) problem is defined as:

Is there a set of values for the boolean variables that assigns the value 1 (true) to the boolean expression given in CNF?

8 Cook's Theorem

Cook's theorem states that

satisfiability of boolean formulas (SAT problem) is NP-Complete.

SAT Problem is NP - Complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

Satisfiability of boolean formulas is NP-Complete.

Proof:

Step 1: Prove that SAT problem is in class NP.

A boolean variable can have two possible values. Hence, for an n variable boolean expression in CNF, there are 2^n possible combinations of values of these variables.

If we want to check whether a boolean formula is satisfiable, we need to apply each combination to the formula until we get a 1 as the result of evaluation. In the worst case, we need to apply all the 2^n possible combinations.

Consider the formula, $(\neg x_1 \cup x_2) \cap (x_1 \cup \neg x_2) \cap (\neg x_2 \cup x_3) \cap (\neg x_1 \cup \neg x_2 \cup x_3)$. There are 3 variables. Each variable can have two possible values, 0 or 1. There are 2^3 combinations.

So the time complexity of this problem is $\Theta(2^n)$. (not polynomial time)

If we are given a certificate containing a satisfiable assignment for a formula, then we can easily verify it in polynomial time.

For example, for the above formula, if we are given a certificate saying that if $x_1 = 0, x_2 = 0, x_3 = 1$, then just applying these values to the formula, we get,

$(\neg x_1 \cup x_2) \cap (x_1 \cup \neg x_2) \cap (\neg x_2 \cup x_3) \cap (\neg x_1 \cup \neg x_2 \cup x_3)$

=$(\neg 0 \cup 0) \cap (0 \cup \neg 0) \cap (\neg 0 \cup 1) \cap (\neg 0 \cup \neg 0 \cup 1)$

=$(1 \cup 0) \cap (0 \cup 1) \cap (1 \cup 1) \cap (1 \cup 1 \cup 1)$

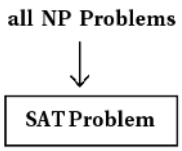
=$1 \cap 1 \cap 1 \cap 1$=1

This can be done in linear time.

So the SAT problem is in class NP.

Step 2: Prove that SAT problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.



A lot of details are involved in this proof. [This proof is beyond the scope of this class].

[Refer the text book 'Introduction to the Theory of computation' by Michael Sipser - Chapter 7- Theorem 7.37]

3-CNF Satisfiability Problem (3-CNF-SAT)

3-CNF

We know that a boolean formula is in conjunctive normal form(CNF) if it is expressed as an AND of clauses, each clause is the OR of one or more variables.

Then, a formula is in 3-conjunctive normal form (3-CNF) is each clause has exactly three distinct literals.

For example, the following boolean formula is in 3-CNF.

$(x_1 \cup \neg x_1 \cup \neg x_2) \cap (x_3 \cup x_2 \cup x_4) \cap (\neg x_1 \cup \neg x_3 \cup \neg x_4) \cap (x_2 \cup x_1 \cup \neg x_3)$

The clauses in the above formula contain exactly 3 literals. The formula is in 3-CNF.

3-CNF Satisfiability Problem

3-CNF-SAT problem is defined as:

Is there a set of values for the boolean variables that assigns the value 1 (true) to the boolean expression given in 3-CNF?

3-CNF Satisfiability Problem is NP-complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

Satisfiability of boolean formulas in 3-CNF is NP-complete.

Proof:

Step 1: Prove that 3-CNF-SAT is in class NP.

A boolean variable can have two possible values. Hence, for an n variable boolean expression in CNF, there are 2^n possible combinations of values of these variables.

If we want to check whether a boolean formula is satisfiable, we need to apply each combination to the formula until we get a 1 as the result of evaluation. In the worst case, we need to apply all the 2^n possible combinations.

Consider the 3-CNF formula, $(x_1 \cup \neg x_1 \cup \neg x_2) \cap (x_3 \cup x_2 \cup x_4) \cap (\neg x_1 \cup \neg x_3 \cup \neg x_4) \cap (x_2 \cup x_1 \cup \neg x_3)$. There are 4 variables. Each variable can have two possible values, 0 or 1. There are 2^4 combinations.

So the time complexity of this problem is $\Theta(2^n)$. (not polynomial time)

If we are given a certificate containing a satisfiable assignment for a formula, then we can easily verify it in polynomial time.

For example, for the above formula, if we are given a certificate saying that if $x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1$, then just applying these values to the formula, we get,

$(x_1 \cup \neg x_1 \cup \neg x_2) \cap (x_3 \cup x_2 \cup x_4) \cap (\neg x_1 \cup \neg x_3 \cup \neg x_4) \cap (x_2 \cup x_1 \cup \neg x_3)$

=$(0 \cup \neg 0 \cup \neg 1) \cap (0 \cup 1 \cup 1) \cap (\neg 0 \cup \neg 0 \cup \neg 1) \cap (1 \cup 0 \cup \neg 0)$

=$(0 \cup 1 \cup 0) \cap (0 \cup 1 \cup 1) \cap (1 \cup 1 \cup 0) \cap (1 \cup 0 \cup 1)$

=$1 \cap 1 \cap 1 \cap 1 \cap 1$=1

This can be done in linear time.

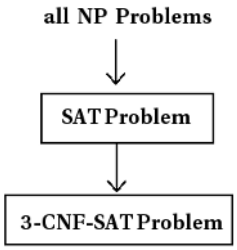
So the 3-CNF-SAT problem is in class NP.

Step 2: Prove that 3-CNF-SAT problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.

The approach we used is as follows:

In the previous section, we already proved that all NP problems can be reduced to SAT. Then, if we can reduce SAT to 3-CNF-SAT in polynomial time, we can say that 3-CNF-SAT is NP-hard.



We learned in the subject, 'logic design' that any boolean formula can be expressed in CNF. Then, a boolean formula in CNF can be rewritten in 3-CNF.

Consider a boolean formula as follows:

$$a \cup b \cup c \cup d$$

This can be converted to 3-CNF as,

$$(a \cup b \cup x) \cap (c \cup d \cup \overline{x})$$

Thus any boolean formula can be transformed to a 3-CNF boolean formula. This can be done using De Morgan's laws.

That means SAT problem can be reduced to 3-CNF-SAT problem. That is, 3-CNF-SAT is NP-hard.

In step 1, we proved that 3-CNF-SAT is in NP.

In step 2, we proved that 3-CNF-SAT is NP-hard.

So, 3-CNF-SAT is an NP-complete problem.

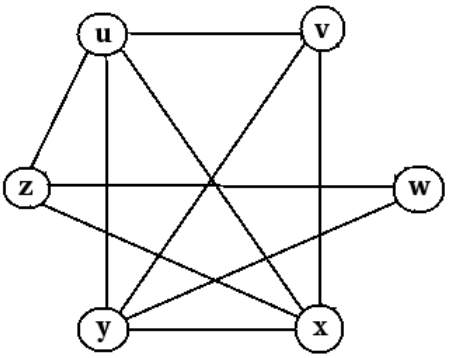
Part V. Clique Problem

9 Clique Problem

Clique

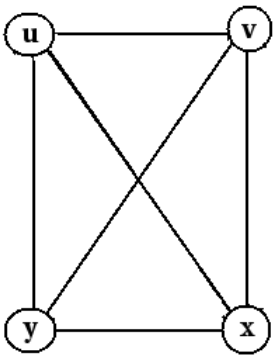
Let we are given a graph with a number of vertices. A clique is a subset of vertices, such that an edge is present between every pair of vertices.

Consider the following graph:



In the above graph, a clique is {u,v,x,y}. The size of this clique is 4.

A clique is a subgraph of the above graph which is shown below:



The above set is a clique, because uv, ux, uy, vx, vy, xy are edges in the given graph.

Clique Problem

Clique problem is to check whether a clique of size k is present in the graph.

Clique Problem is NP-Complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

Clique problem is NP-complete.

Proof:

Step 1: Prove that clique problem is in class NP.

A set with n elements has 2^n possible subsets. Then a graph with n vertices has 2^n possible subgraphs.

So an algorithm needs to check whether any one of these subgraphs form a clique. It has worst case time complexity $\Theta(2^n)$. (not polynomial time, but exponential time).

Let we are given a graph and a 'certificate' telling that a subset of vertices form a clique. An algorithm can verify this in polynomila time.

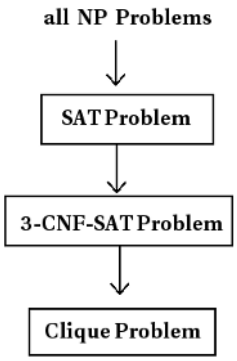
So clique problem is in class NP.

Step 2: Prove that clique problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.

The approach we used is as follows:

In the previous section, we already proved that all NP problems can be reduced to SAT. Then, again we found that SAT problem can be reduced to 3-CNF-SAT problem. Then, if we can reduce 3-CNF-SAT problem to clique problem in polynomial time, we can say that clique is NP-hard.

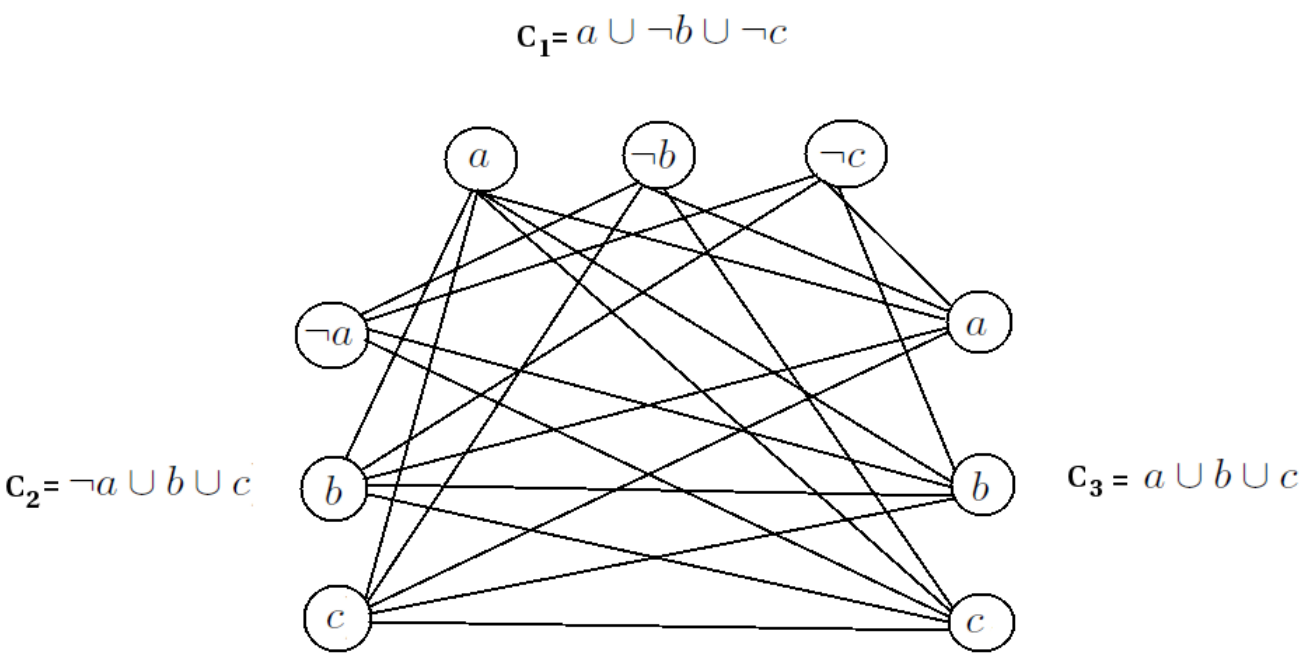


We need to reduce 3-CNF-SAT problem to clique problem.

Consider a 3-CNF boolean formula given below:

$$(a \cup \neg b \cup \neg c) \cap (\neg a \cup b \cup c) \cap (a \cup b \cup c)$$

A graph is produced from the above formula as:

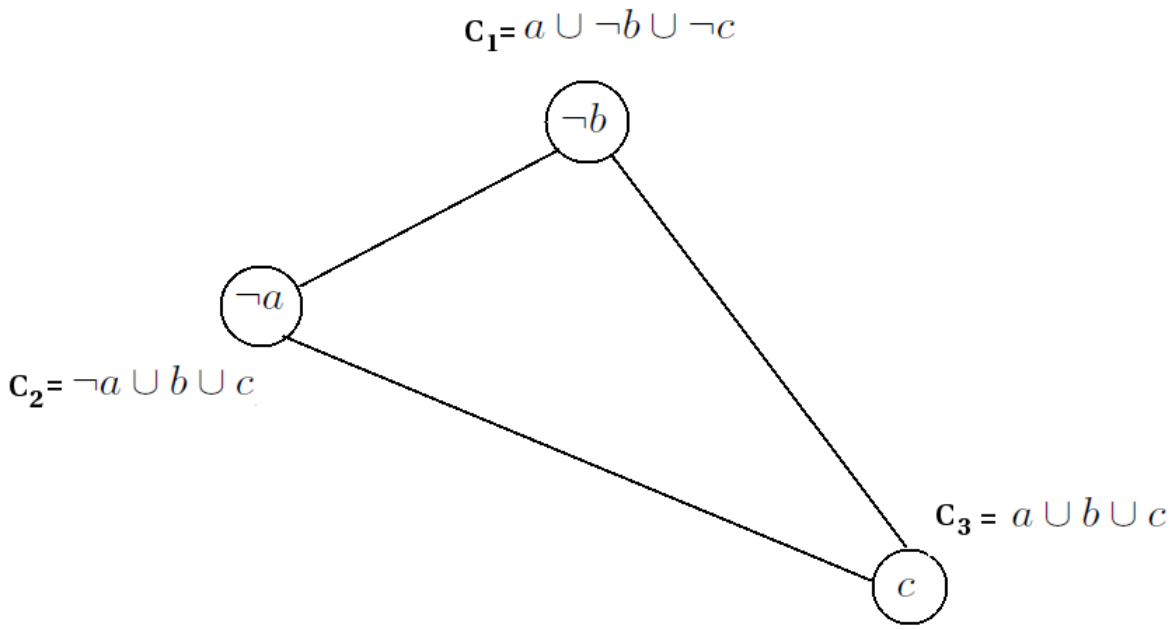


The graph is produced as follows:

Every vertex in a clause is connected to every other vertex in another clause. But a vertex x should not be connected to vertex $\overline{x}$.

This graph corresponding to 3-CNF-SAT has a solution if the corresponding vertices form a clique.

In the above graph, vertices $\neg b$ of C_1, $\neg a$ of C_2, c of C_3 form a clique.



In the above clique, C_1has the vertex $\neg b$. So put b=0.

C_2has the vertex $\neg a$. So put a=0.

C_3has the vertex c. So put c=1.

Thus a satisfying assignment to the given boolean formula is a=0, b=0, c=1.

Consider the given boolean formula,

$$(a \cup \neg b \cup \neg c) \cap (\neg a \cup b \cup c) \cap (a \cup b \cup c)$$

Put a=0, b=0, c=1, we get,

$(0 \cup \neg 0 \cup \neg 1) \cap (\neg 0 \cup 0 \cup 1) \cap (0 \cup 0 \cup 1)$

$=(0 \cup 1 \cup 0) \cap (1 \cup 0 \cup 1) \cap (0 \cup 0 \cup 1)$=$1 \cap 1 \cap 1$=1

Thus the given formula in 3-CNF gives true for a=0, b=0, c=1.

Thus the given 3-CNF-SAT problem is reduced to clique problem. This means clique problem is NP-hard.

In step 1, we proved that clique is in NP.

In step 2, we proved that clique is NP-hard.

So, clique is an NP-complete problem.

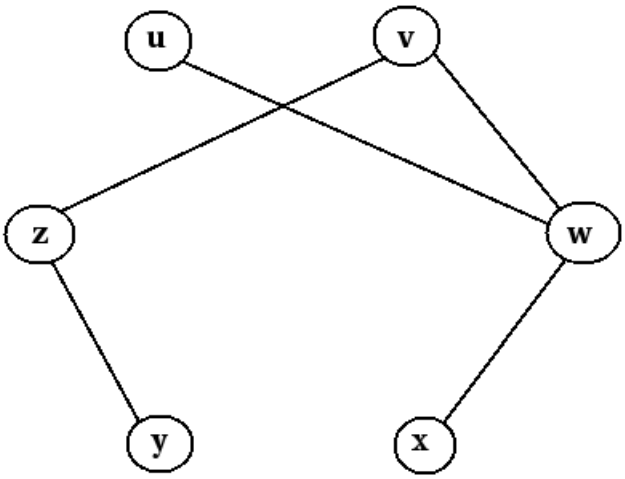
Part VI. Vertex Cover Problem

10 Vertex Cover Problem

Vertex Cover

A vertex cover of a graph is a set of vertices, V such that if (a, b) is an edge of a graph, then a or b or both must be present in V.

Consider the following graph:



In the above graph, vertex cover is {z, w}.

Let V= {z, w}

The edges of the above graph are uw, vw, xw, vz, yz. Note that in every edge, at least one vertex is a member of V.

For example,

Consider the edge uw. Here, vertex $w \in V$.

Consider the edge vz. Here, vertex $z \in V$.

Thus, {z, w} form a vertex cover.

Vertex Cover Problem

The vertex cover problem is to check whether a graph has a vertex cover of size k.

Vertex Cover Problem is NP-Complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

Vertex cover problem is NP-complete.

Proof:

Step 1: Prove that vertex cover problem is in class NP.

A set with n elements has 2^n possible subsets. Then a graph with n vertices has 2^n possible subsets of vertices.

So an algorithm needs to check whether any one of these subsets form a vertex cover. It has worst case time complexity $\Theta(2^n)$. (not polynomial time, but exponential time).

Let we are given a graph and a 'certificate' telling that a subset of vertices form a vertex cover. An algorithm can verify whether at leat one vertex of every edge in the graph is an element of this vertex cover in polynomial time.

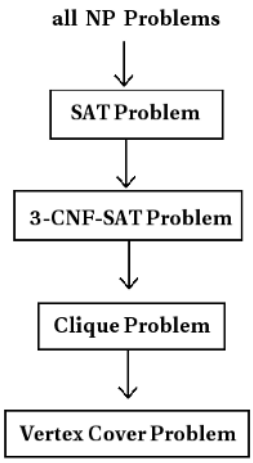
So vertex cover problem is in class NP.

Step 2: Prove that vertex cover problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.

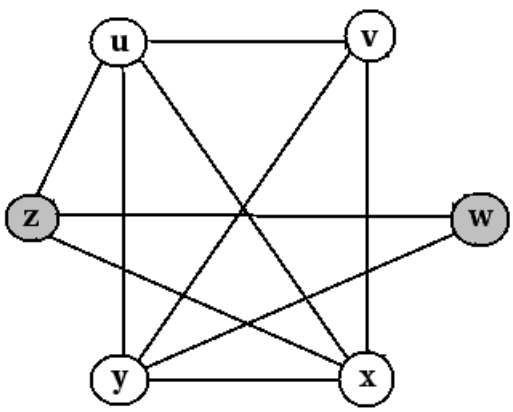
The approach we used is as follows:

In the previous section, we already proved that all NP problems can be reduced to SAT. Then, again we found that SAT problem can be reduced to 3-CNF-SAT problem. Again, we reduced 3-CNF-SAT problem to clique problem. Then, if we can reduce clique problem to vertex cover problem in polynomial time, we can say that vertex cover problem is NP-hard.



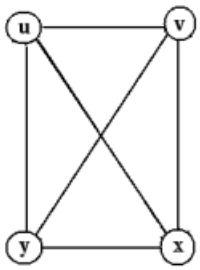
We need to reduce clique problem to vertex cover problem.

Consider the following graph with clique {u, v, x, y}.



Graph, G

V={u,v,w,x,y,z}

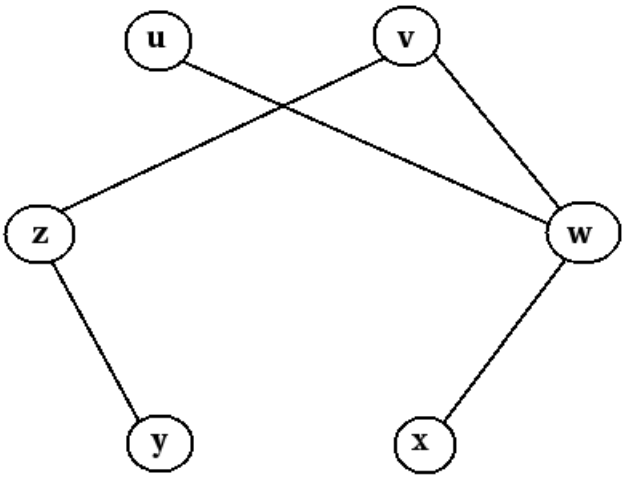


Clique of G = {u,v,x,y}

V ' ={u,v,xy}

Take the complement of the above graph, G. That is $\overline{G}$.

$\overline{G}$ is



Graph, $\overline{G}$

For the graph $\overline{G}$, vertex cover is {z,w}.

This vertex cover is found by V - V'.

That is, {u,v,w,x,y,z} - {u,v,x,y} = {z,w}

So from the clique, we can find out the vertex cover of a graph using the above mechanism.

Thus the given clique problem is reduced to vertex cover problem. This means vertex cover problem is NP-hard.

In step 1, we proved that vertex cover problem is in NP.

In step 2, we proved that vertex cover problem is NP-hard.

So, vertex cover problem is an NP-complete problem.

Part VII. Hamiltonian Cycle Problem

11 Hamiltonian Cycle Problem

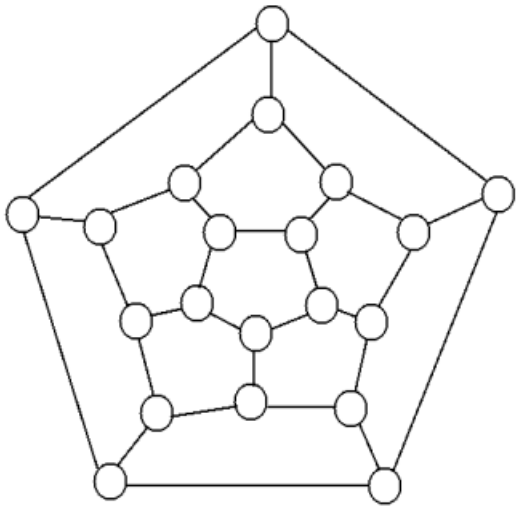
Hamiltonian Cycle

A hamiltonian cycle in an undirected graph is a simple cycle which contains every vertex of the graph.

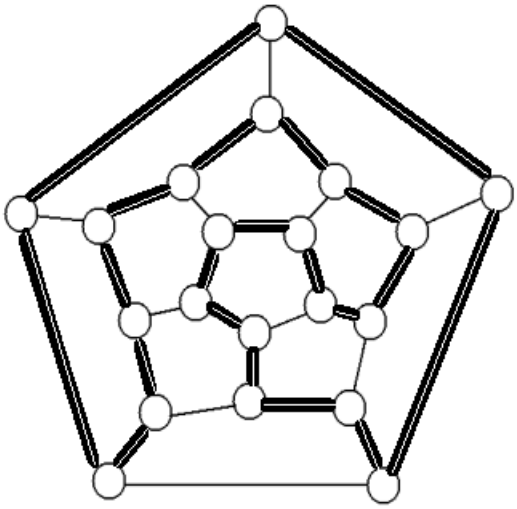
A graph which contains a hamiltonian cycle is called hamiltonian. A graph which does not contain a hamiltonian cycle is called nonhamiltonian.

Example:

Consider the following graph:



Graph



Ham Cycle is shown in thick lines

Note that the hamiltonian cycle path passes through every vertex.

Hamiltonian Cycle Problem

The hamiltonian cycle problem is to check whether a graph has a hamiltonian cycle..

Hamiltonian Cycle Problem is NP-Complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

Hamiltonian cycle problem is NP-complete.

Proof:

Step 1: Prove that hamiltonian cycle problem is in class NP.

A set with n elements has 2^n possible subsets. Then a graph with n vertices has 2^n possible permutations of vertices.

So an algorithm needs to check whether any one of these permutation form a hamiltonian path. If the graph is represented as an adjacency matrix, it will take $n!$ comparisons. It has worst case time complexity $\Theta(2^{\sqrt{n}})$. (not polynomial time, but exponential time).

Let we are given a graph and a 'certificate' telling that a sequence of vertices form a hamiltonian cycle. An algorithm can verify whether this path is hamiltonian in polynomial time.

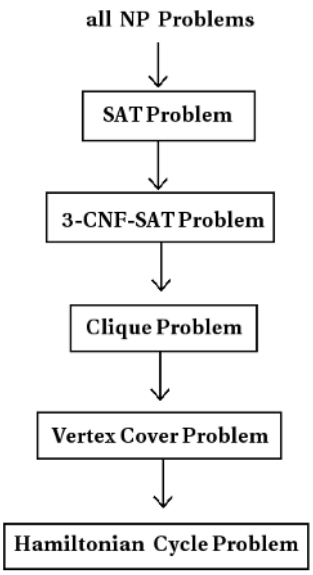
So hamiltonian cycle problem is in class NP.

Step 2: Prove that hamiltonian cycle problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.

The approach we used is as follows:

In the previous section, we already proved that all NP problems can be reduced to SAT. Then, again we found that SAT problem can be reduced to 3-CNF-SAT problem. Again, we reduced 3-CNF-SAT problem to clique problem. again, we reduced clique problem to vertex cover problem. Then, if we can reduce vertex cover problem to hamiltonian cycle problem in polynomial time, we can say that hamiltonian cycle problem is NP-hard.



We need to reduce vertex cover problem to hamiltonian cycle problem.

* [The proof of this part is beyond the scope of this class]
* [Refer the text book: Introduction to Algorithms by Cormen, Chapter 34-Section 34.5.3]

Part VIII. Travelling Salesman Problem (TSP)

12 Travelling Salesman Problem (TSP)

Travelling Salesman

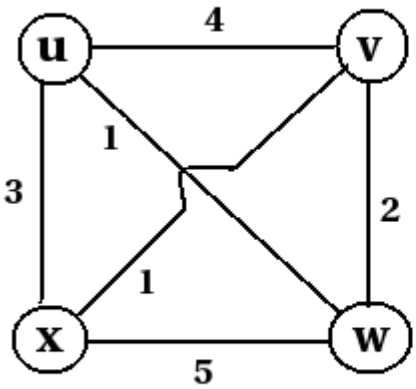
A salesman begins his tour from a city. He visits a set of cities and he finishes at the city he starts from. Each city must be visited exactly once.

Travelling Salesman Problem (TSP)

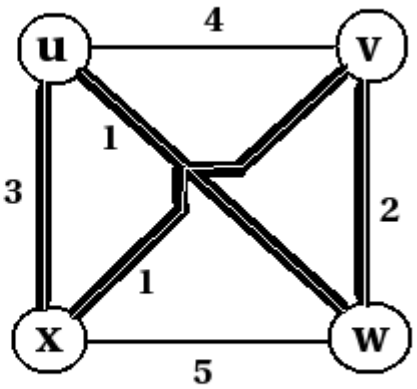
A graph of a set of cities is given. A salesman starts from a city. He must visit each city exactly once and should finish at the city he starts from. The path he follows should be the shortest one or should have a cost value less than k.

Thsu TSP is a special case of Hamiltonian cycle problem where the path cost should be minimum.

Consider the following graph containing cities u, v, w, x.



Graph



cycle x-u-w-v-x

Here the traveling salesman can follow the path x-u-w-v-x which forms the shortest path with cost 7km.

TSP is NP-Complete

We know that a problem is in class NP-Complete if: it is in NP and it is NP Hard.

Theorem:

TSP is NP-complete.

Proof:

Step 1: Prove that TSP is in class NP.

A set with n elements has 2^n possible subsets. Then a graph with n vertices has 2^n possible permutations of vertices.

So an algorithm needs to check whether any one of these permutation form a hamiltonian path with cost value less than k. If the graph is represented as an adjacency matrix, it will take $n!$ comparisons. It has worst case time complexity $\Theta(2^{\sqrt{n}})$. (not polynomial time, but exponential time).

Let we are given a graph and a 'certificate' telling that a sequence of vertices form a hamiltonian cycle with cost value less than k. An algorithm can verify whether this cycle has cost less than k in polynomial time.

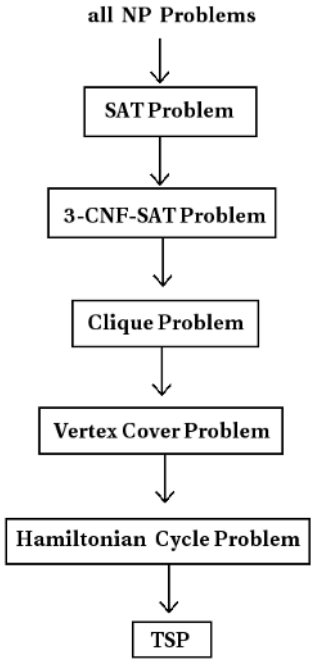
So TSP is in class NP.

Step 2: Prove that hamiltonian cycle problem is NP-hard.

We know that a problem is NP-Hard, if every problem in NP can be reduced to this problem in polynomial time.

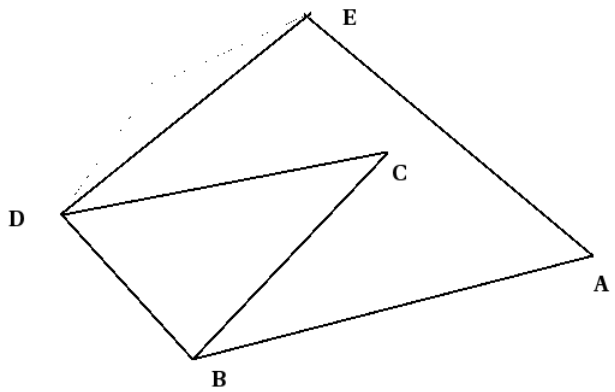
The approach we used is as follows:

In the previous section, we already proved that all NP problems can be reduced to SAT. Then, again we found that SAT problem can be reduced to 3-CNF-SAT problem. Again, we reduced 3-CNF-SAT problem to clique problem. Again, we reduced clique problem to vertex cover problem. Again we reduced vertex cover problem to hamiltonian cycle problem. Then, if we can reduce hamiltonian cycle problem to TSP in polynomial time, we can say that TSP is NP-hard.

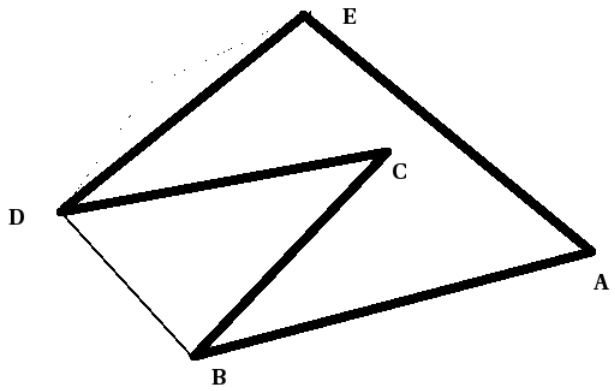


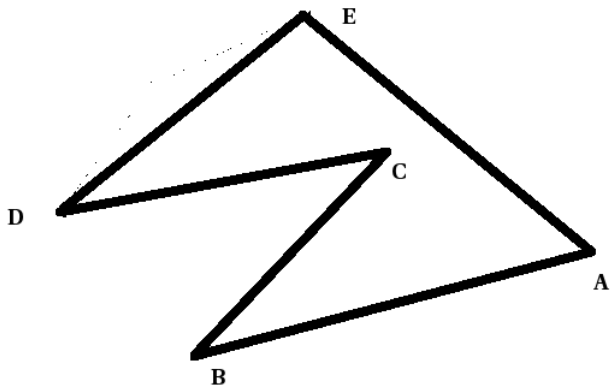
We need to reduce hamiltonian cycle problem to TSP.

Consider a graph,

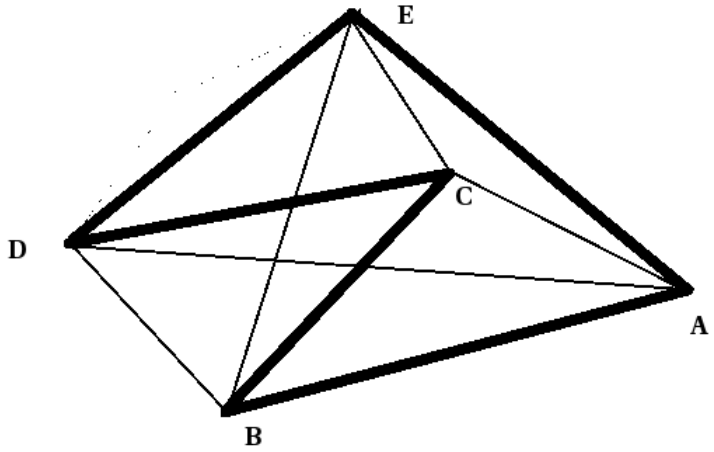


Following is an instance of hamiltonian cycle.

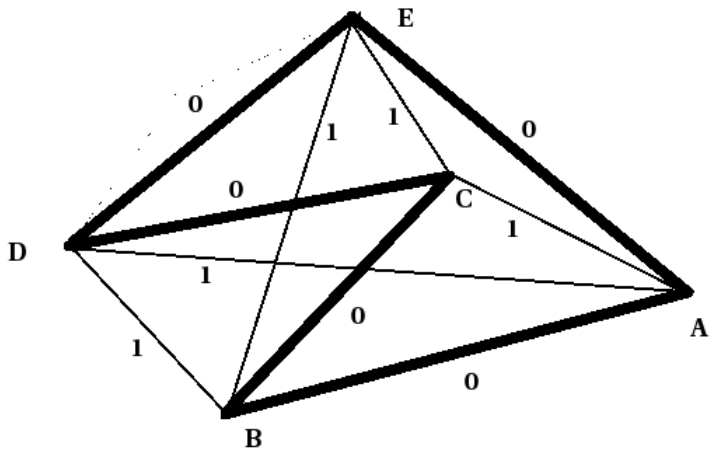




Now form a complete graph from the above istance of Hamiltonian cycle.



Next we assign cost value to every edge of this graph. If the edge is present in hamiltonian cycle, assign cost value 0 to it. If it is not, assign cost value 1 to it.



From this, an instance of TSP can be produced from the above graph by traversing through the edges which have cost value 0.

Thus hamiltonian cycle problem is reduced to TSP. So TSP is NP-hard.

In step 1, we proved that TSP is in NP.

In step 2, we proved that TSP is NP-hard.

So, TSP is an NP-complete problem.

Questions (from Old syllabus of S7CS TOC)

MGU/Nov2011

1. What is meant by intractable problem (4marks)?

2. Briefly explain NP hard problem (4marks).

3a. Explain the different classification of problems based on their complexity.

OR

b. Prove that the satisfiability problem in NP complete (12marks).

MGU/April2011

1. What are complexity classes (4marks)?

2a. Is travelling salesman problem is NP-complete or not? Prove (12marks).

OR

b.

MGU/Nov2010

1. Differentiate tractable and intractable problems (4marks).

2. What is an NP hard problem? Give example (4marks).

3a. Show that halting problem of a Turing machine is not NP complete.

OR

b. What are complexity classes and tractable and intractable problems (12marks).

MGU/May2010

1a. Define the class NP (4marks).

b.

2a.

OR

b. i. Let L be an NP complete language. Then P=NP if and only if $L \sum P$.

ii. Construct truth tables for the following formula:- $(A \longleftrightarrow (A \longleftrightarrow B))$. (12marks).

3a. Show that Travelling salesman problem is NP-complete.

OR

b. Show that the following formula of propositional calculus is a Tautology (12marks).

MGU/Nov2009

1. What are tractable and intractable problems (4marks)?

2a.

OR

b. Explain in detail about class P, NP complete and NP hard problems (12marks).

MGU/May2009

1. Give an example of an NP complete problem (4marks).

2a. Prove that Travelling Salesman's problem is NP complete.

OR

b. State the halting problem. Show that it is not NP complete (12marks).

MGU/Nov2008

1. Explain tractable and intractable problems (4marks).

2a. Explain P, NP, NP-hard and NP complete problems with example (12marks).

OR

b.

MGU/May2008

1. Cite example for NP hard problem (4marks).

2a. Prove that Traveleing Salesman's problem is NP complete (12marks).

OR

b.

MGU/Dec2007

1. Explain classes P, NP and NP completeness (4marks).

2a. Explain algorithmic complexity and NP hard problems.

OR

b. What is integer programming and show how it is a NP-complete problem (12marks).

MGU/July2007

1. When do you call a decision problem as Np-hard (4marks)?

2. a.

OR

b. Show that the problem of checking whether a graph has a clique of size k is NP-complete (12marks).

MGU/Jan2007

1. What will happen when somebody finds a deterministic polynomial time algorithm for an NP complete problem? Comment on its consequence on the complexity classes (4marks).

2. Define class P and class NP (4marks).

3a. Prove that 3SAT problem is NP-complete.

OR

b. State Cook's theorem and give an outline of its proof (12marks).

MGU/July2006

1. What is an NP problem? Give an example (4marks).

2a. Discuss any two NP hard graph problems in detail.

OR

b. i. What are NP-complete problems? Give two examples.

ii. Briefly explain the satisfiability problem. Is it an NP problem (12marks)?

MGU/Nov2005

1. Differentiate between tractable and intractable problems (4marks).

2a. Define NP, NP-hard, NP-complete and P problems. Explain with examples.

OR

b. State the halting problem. Show that it is not NP-complete (12marks).

References

.

Cormen, T, H; Leiserson, C, E; Rivest, R, L; (2001). Introduction to Algorithms. PHI.

Nagpal, C, K (2011). Formal Languages and Automata Theory. Oxford University Press.

Sipser, M (2007). Introduction to the Theory of Computation. Thomson Course Technology.

Hopcroft, J, E; Motwani, J; Ullman, J, D (2002). Introduction to Automata Theory, Languages and Computation. Pearson Education.

website: http://sites.google.com/site/sjcetcssz