

**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**



**COURSE MATERIAL
CS6503-THEORY OF COMPUTATION
III YEAR - V SEMESTER**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SYLLABUS THEORY

Sub. Code	: CS6503	Branch/Year/Sem	: CSE/III/V
Sub Name	: THEORY OF COMPUTATION	Batch	: 2017-2021
Staff Name	: G.MEGALA	Academic Year	: 2019-2020

UNIT I FINITE AUTOMATA CS6503 Syllabus Theory of Computation

Introduction- Basic Mathematical Notation and techniques- Finite State systems – Basic Definitions – Finite Automaton – DFA & NDFA – Finite Automaton with ϵ - moves – Regular Languages- Regular Expression – Equivalence of NFA and DFA – Equivalence of NDFA’s with and without ϵ - moves – Equivalence of finite Automaton and regular expressions –Minimization of DFA- – Pumping Lemma for Regular sets – Problems based on Pumping Lemma.

UNIT II GRAMMARS CS6503 Syllabus Theory of Computation TOC

Grammar Introduction– Types of Grammar – Context Free Grammars and Languages– Derivations and Languages – Ambiguity- Relationship between derivation and derivation trees – Simplification of CFG – Elimination of Useless symbols – Unit productions – Null productions – Greiback Normal form – Chomsky normal form – Problems related to CNF and GNF.

UNIT III PUSHDOWN AUTOMATA

Pushdown Automata- Definitions – Moves – Instantaneous descriptions – Deterministic pushdown automata – Equivalence of Pushdown automata and CFL – pumping lemma for CFL – problems based on pumping Lemma.

UNIT IV TURING MACHINES

Definitions of Turing machines – Models – Computable languages and functions –Techniques for Turing machine construction – Multi head and Multi tape Turing Machines – The Halting problem – Partial Solvability – Problems about Turing machine- Chomskian hierarchy of languages.

UNIT V UNSOLVABLE PROBLEMS AND COMPUTABLE FUNCTIONS

Unsolvable Problems and Computable Functions – Primitive recursive functions – Recursive and recursively enumerable languages – Universal Turing machine. MEASURING AND CLASSIFYING COMPLEXITY: Tractable and Intractable problems- Tractable and possibly intractable problems – P and NP completeness – Polynomial time reductions.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Sub. Code : CS6503
Sub Name : THEORY OF COMPUTATION
Staff Name : G.MEGALA

Branch/Year/Sem : CSE/III/V
Batch : 2017-2021
Academic Year : 2019-2020

COURSE OBJECTIVES:

Student will be able to,

- Understand various computing models like finite automata, Push down data and turning machine.
- Understand grammar for a language.
- Prove equivalence of language described by automata.
- Acquire awareness of decidability and undecidability of various problem.

COURSE OUTCOME:

S.NO	DESCRIPTION	BLOOM'S TAXONOMY (BT) KNOWLEDGE LEVEL
CO1	Student will be able to design automata and prove a statement	apply
CO2	Construct regular expression for a pattern	apply
CO3	Correlate different types of automata to real world applications	apply
CO4	Design a turning machine to solve problem on mathematical foundations	apply
CO5	Decide whether a problem is decidable or not	understand
CO6	Identify different computational complexities	understand

UNIT -1 AUTOMATA FUNDAMENTALS

INTRODUCTION

- The study of abstract devices is called automata theory.
- **Finite Automata is a mathematical model that accepts a set of inputs, processes through a set of states, generates an output.**
- Automation helps in
 - Designing and checking the behavior of digital circuits
 - Pattern searching in websites
 - Verifying systems like communication protocols for secure data transfers
 - Designing lexical analyzers in compilers
- Finite automation model is a study of machines.

History

Alan Turing introduced an abstract machine in 1930's. This machine had all the

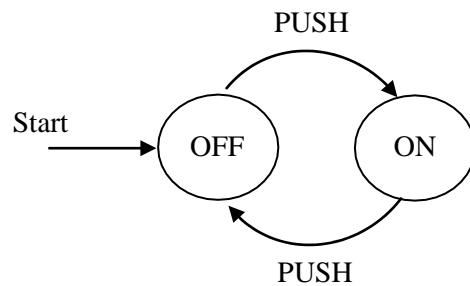
capabilities of today's computer.

In 1940's and 1950's, the machines were simplified to finite automation machines. Researchers proposed the automation model in order to model the functions of human brain.

The linguist Noam Chomsky made a study on formal grammars in late 1950's . These grammars provided the basis of compilers.

S.Cook extended Turing's study in 1969, which separated the problems as solvable and unsolvable. He classified problems as NP-Hard and as NP-Complete.

Example of a Finite state system: Switch operation

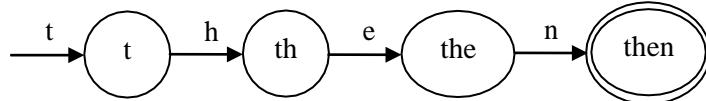


1.2 Theory of Computation

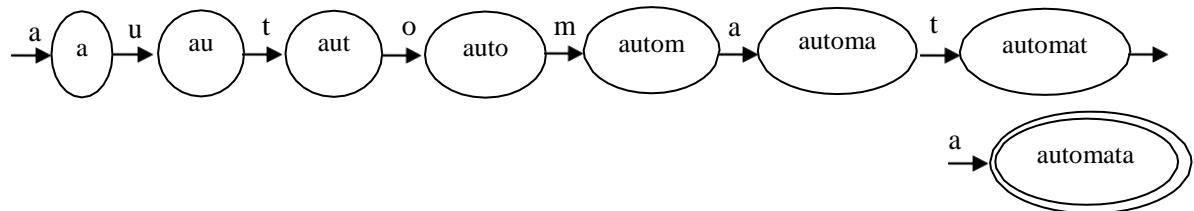
Here,

- When the electric switch = “ON” indicates logic „1“ when pushed from start state.
- Goes to „OFF“ state when pushed from „ON“ state.

Example: Pattern searching of string = “then”



Example: Pattern searching of string = “automata”



BASIC MATHEMATICAL NOTATION AND TECHNIQUES

Basic Mathematical

Objects Sets [A]

A set is collection of elements of finite number.

Example:

$$A = \{10, 01, 00, 11\}$$

$$B = \{w \mid w \text{ is a set of prime numbers less than } 100\}$$

$$\Rightarrow w \in B$$

$$C = \{r \mid r \text{ is a set of strings generated from vowels}\}$$

$$\Rightarrow r \in C$$

Subset [\subseteq]

Let A_1 and A_2 are two sets, then A_1 is a subset of A_2 [indicated as $A_1 \subseteq A_2$], if every element of A_1 is in A_2 .

Example:

$$A_1 = \{1, 2, 3, 4\}$$

$$A_2 = \{1, 2, 3, 4, 5\}$$

$$\Rightarrow A_1 \subseteq A_2$$

Complement of a Set [A']

Let A be a set of finite number of elements. Then A' is a set of elements that are not the elements of set A. [A' ≠ A]

Example:

$$A_1 = \{a, e, i, o, u\}$$

$$A' = \{ \text{ all alphabets except vowels} \}$$

Operations on Sets

Union [U]

The union of two sets results in a set containing the elements of both the sets (without repetition of elements).

Example:

$$A_1 = \{1, 3, 5, 7\}$$

$$A_2 = \{1, 2, 3, 4\}$$

$$A_1 \cup A_2 = \{1, 2, 3, 4, 5, 7\}$$

Intersection [∩]

The intersection of two sets contains a set of elements that are available in both the sets.

Example:

$$A_1 = \{1, 3, 5, 7\}$$

$$A_2 = \{1, 2, 3, 4\}$$

$$A_1 \cap A_2 = \{1, 3\}$$

Difference [-]

The difference of two sets A and B results in the set of elements that are in A and not in B.

Example:

$$A_1 = \{1, 3, 5, 7\}$$

$$A_2 = \{1, 2, 3, 4\}$$

$$A_1 - A_2 = \{5, 7\} \quad \Rightarrow [A_1 - A_2 = A_1 \cap A_2']$$

Laws on Sets

Commutative Laws

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

Associative Laws

$$A \cup (B \cup C) = (A \cup B) \cup C$$

$$A \cap (B \cap C) = (A \cap B) \cap C$$

Distributive Laws

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

Idempotent Laws

$$A \cup A = A$$

$$A \cap B = A$$

Absorptive Laws

$$A \cup (A \cap B) = A$$

$$A \cap (A \cup B) = A$$

De Morgan's Laws

$$(A \cup B)' = A' \cap B'$$

$$(A \cap B)' = A' \cup B'$$

Other Laws

$$(A')' = A$$

$$A \cap A' = \emptyset$$

$$A \cup A' = \text{Universal set, "U"}$$

$$A \cup \emptyset = A$$

$$A \cap \emptyset = \emptyset$$

$$A \cup U = U$$

$$A \cap U = A$$

Equality of sets [=]

Two sets are set to be equal if both the sets contain the same elements.

Example:

$$A = \{1, 3, 5, 7\}$$

$$B = \{\text{set of odd numbers less than } 8\}$$

$$\text{Then, } A = B$$

Empty set [ϕ]

A set containing no elements is said to be an empty set.

Example:

$$A = \{\text{set of natural numbers between 1 and 2}\}$$

$$\Rightarrow A = \phi \text{ or } \{\}$$

Power of a set [A^k]

The power of a set A, denoted by A^k is the set containing all elements of A, whose length is less than or equal to k.

Example:

$$A = \{1, 2\}$$

$$A^2 = \{\phi, 1, 2, 12\}$$

Relationship of Sets

Relation of two sets is the association of one set with other.

Types based on the properties of sets

- 1) **Reflexive:** Every element of the set is associated with itself.
- 2) **Symmetric:** Let $a, b \in A$, then a is related to b , and b is related to a [$aRb = bRa$]
- 3) **Transitive:** If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$. It is denoted as: aRb, bRc then aRc .

Logic

Logic deals with the logical proportions and connectives.

Proposition

Proposition is a meaningful, objective, declarative statement that has a truth value as true / false.

Example:

$0 \neq 1$, New Delhi is the capital of India, 5 is an even number, Chennai is in AP.

Connective

Logical connective is a compound statement formed from simple proportions using a connective such as

$\wedge \rightarrow$ and connective /conjunction

$\vee \rightarrow$ or connective /disjunction

$\neg \rightarrow$ Not / Negative connective

Example

p	Q	$p \wedge q$	$p \vee q$	$\neg p$	$\neg q$
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

Implication

It is a conditional statement of the form, “if p then q” (p implies q), where p and q are simple proportions.

It is denoted as $p \rightarrow q = \neg p \vee q$

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Proofs

A proof is single line / multiline derivation that provide a convincing argument to make a statement true.

Proofs can be derived from assumptions or facts or existing derivations.

Forms of Proofs

There are basically two forms of proofs. They are,

- 1) Deductive Proofs
- 2) Inductive Proofs

Deductive Proofs

These are sequence of statements which are derived from any assumption (hypothesis) or a given initial statement to a conclusion statement.

The proofs are generated using some facts / accepted logics.

Example: If $x \geq 4$, $2^x \geq x^2$

The hypothesis statement $\Rightarrow x \geq 4$

Conclusion statement $\Rightarrow 2^x \geq x^2$

This can be provided as substitution statement as:

When $x = 4$: $2^4 \geq 4^2 \Rightarrow 16 \geq 16 \Rightarrow \text{True}$

When $x = 5$: $2^5 \geq 5^2 \Rightarrow 32 \geq 25 \Rightarrow \text{True}$

When $x = 6$: $2^6 \geq 6^2 \Rightarrow 16 \geq 36 \Rightarrow \text{True}$

When $x = 3$: $2^3 \geq 3^2 \Rightarrow 16 \geq 9 \Rightarrow \text{False}$

Hence proved.

Inductive Proof

It has a sequence of recursive / parametrical statements that handle the statement with lower values of its parameters.

There are two types of inductions, namely -

- Mathematical Induction
- Structural Induction

Mathematical Induction

This follows induction principle that follows two steps, namely-

- **Basis of Induction:** It considers „n“ of $f(n)$ as the lowest possible integer ($n = 0$ or 1) and prove the given statement using substitution.

- **Inductive step:** Assume that the given statement is true for n^{th} value and prove the statement for $n = n+1$.

This applies for all integer related statement.

Structural Induction

Structural Induction follows the mathematical induction concept but applies for trees and expressions.

Example: Every tree has one node than it has edges, balancing parenthesis etc.

Additional forms of proofs

- Proofs about sets
- Proof by contradiction
- Proof by counter example
- Direct Proofs

Proofs about set

Proof about the set includes the proofs on the properties of the sets.

Example: Commutative law, Distributive law etc

Consider, $A = P \cup Q$, $B = Q \cup P$

Since $P \cup Q = Q \cup P$ (by Commutative law), we prove that $A = B$.

This proof follows “if and only if” type. This is $A = B$ is true if and only if $P \cup Q = Q \cup P$ is true.

Proof by contradiction

The proof $p \rightarrow q$ is true, then the method follows a contradictory assumption that $p \neq q$ and from the result, it proves the assumption is false and thus proves, $p \Rightarrow q$.

Example:

To prove $P \cap Q = Q \cap P$, we first assume that $P \cap Q \neq Q \cap P$.

Now consider an element, x from P that is also in Q .

$$\Rightarrow x \in P \cap Q$$

Thus all the element of P is taken and compared with that of Q .

Finally, $P \cap Q = Q \cap P$ is proved [contradiction to our assumption]

Proof by Counter Example

Given a statement, that can be proved to be true for a domain of input instances, and false for other inputs.

Example: $a \bmod b \neq b \bmod a$

Let $a = 5$ and $b = 10$

$$a \% b = 5 \% 10 = 5$$

$$b \% a = 10 \% 5 = 0$$

Hence $a \% b \neq b \% a$, the statement is true only if $a = b$.

Thus proved.

Inductive Proofs

These are special type of proofs used to prove recursively defined objects.

It consists of sequence of parameterized statement that uses the statement itself with lower values of its parameter.

Two steps

1. **Basis of Induction** → We start with the lowest possible value.

Example: To prove $f(n)$. We take $n = 0$ or 1 initially.

2. **Inductive step** → Here we prove if $f(n)$ is true, $f(n+1)$ is also true.

SAMPLE PROBLEMS:

1. Prove that: $1+2+3+\dots+n \text{ (n+1) } / 2$ using method of induction for ($n > 0$).

Proof:

Basis of Induction

Let $n = 1$ [We cannot take $n = 0$ since U.H.S starts from 1]

$$\text{L.H.S} = 1$$

$$\text{R.H.S} = \frac{1(1+1)}{2} = \frac{1(2)}{2} = \frac{2}{2} = 1$$

1.10 Theory of Computation

Since L.H.S = R.H.S \Rightarrow Proved.

Inductive Step

We have $1+2+3+\dots+n = \frac{n(n+1)}{2}$

Sub $n = n + 1$

$$\text{L.H.S} \quad 1 + 2 + 3 + \dots + n = \sum_{i=1}^n i$$

When we substitute $n = n+1$,

$$\begin{aligned}
 \text{L.H.S} &= \sum_{i=1}^{n+1} i & \text{Eg: } \sum_{i=1}^5 i = \sum_{i=1}^4 i + 5 \\
 &= \sum_{i=1}^n i + (n + 1) \\
 &= \frac{n(n + 1)}{2} + (n + 1) \\
 &= \frac{n(n + 1) + 2(n + 1)}{2} \\
 &= \frac{n^2 + n + 2n + 2}{2} \\
 &= \frac{n^2 + 3n + 2}{2} \quad \text{----- (1)}
 \end{aligned}$$

$$\text{R.H.S} = \frac{n(n + 1)}{2}$$

When we substitute $n=n+1$,

$$\begin{aligned}
 \text{R.H.S} &= \frac{(n + 1)((n + 1) + 1)}{2} \\
 &= \frac{(n + 1)(n + 2)}{2} \\
 &= \frac{n^2 + 2n + n + 2}{2}
 \end{aligned}$$

$$\leftarrow \qquad \qquad \qquad = \frac{n^2 + 3n + 2}{2} \qquad \qquad \qquad \text{----- (2)}$$

From (1) and (2), L.H.S = R.H.S

So the given hypothesis is proved.

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

Basis of induction

Let $n=1$ (we can't take $n=0$ since $i=1$ in L.H.S)

$$\text{L.H.S} = \sum_{i=1}^n i^2 = 1^2 = 1$$

$$\text{R.H.S} = \frac{1(1+1)(2(1)+1)}{6} = \frac{1(2)(2+1)}{6} = \frac{2(3)}{6} = \frac{6}{6} = 1$$

$$\text{L.H.S} = \text{R.H.S.}$$

Hence basis of induction is proved.

Inductive step

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

L.H.S: Sub $n=n+1$

$$\begin{aligned} &= \sum_{i=1}^{n+1} i^2 \\ &= \sum_{i=1}^n i^2 + (n+1)^2 \qquad \qquad \sum_{i=1}^3 i^2 = \sum_{i=1}^2 i^2 + (3)^2 \\ &= \frac{n(n+1)(2n+1)}{6} + (n+1)^2 \\ &= \frac{n(n+1)(2n+1) + 6(n+1)^2}{6} \\ &= \frac{n(2n^2 + n + 2n + 1) + 6(n^2 + 2n + 1)}{6} \end{aligned}$$

$$\begin{aligned}
 &= \frac{2n^3 + n^2 + 2n^2 + n + 6n^2 + 12n + 6}{6} \\
 &= \frac{2n^3 + 9n^2 + 13n + 6}{6} \quad \text{-----(1)}
 \end{aligned}$$

R.H.S: Sub $n=n+1$

$$\begin{aligned}
 \frac{n(n+1)(2n+1)}{6} &= \frac{(n+1)((n+1)+1)(2(n+1)+1)}{6} \\
 &= \frac{(n+1)(n+2)(2n+2+1)}{6} \\
 &= \frac{(n^2 + 2n + n + 2)(2n + 3)}{6} \\
 &= \frac{2n^3 + 3n^2 + 4n^2 + 6n + 2n^2 + 3n + 4n + 6}{6} \\
 &= \frac{2n^3 + 9n^2 + 13n + 6}{6}
 \end{aligned}$$

From (1) and (2) L.H.S = R.H.S

The hypothesis is proved.

3) P.T for every integer, $n \geq 0$ the number $4^{2n+1} + 3^{n+2}$ is a multiple of 13.

Solution

Basis of induction

Let $n = 0$

$$\begin{aligned}
 \text{L.H.S} &= 4^{2(0)+1} + 3^{(0)+2} \\
 &= 4^1 + 3^2 = 4 + 9 \\
 &= 13 \Rightarrow \text{multiple of 13}
 \end{aligned}$$

Inductive proof

Let $n=n+1$

$$\begin{aligned}
 4^{2n+1} + 3^{n+2} &= 4^{2(n+1)+1} + 3^{(n+1)+2} \\
 &= 4^{2n+2+1} + 3^{n+3}
 \end{aligned}$$



$$\begin{aligned}
 &= 4^{2n+3} + 3^{n+3} \\
 &= 4^{2n+1} \cdot 4^2 + 3^{n+2} \cdot 3^1 \\
 &= 16(4^{2n+1}) + 3(3^{n+2}) \\
 &= 13(4^{2n+1}) + 3(4^{2n+1}) + 3(3^{n+2}) \\
 &= 13(4^{2n+1}) + 3(\underbrace{4^{2n+1} + 3^{n+2}}_{\downarrow})
 \end{aligned}$$

Multiple of 13

$$\begin{aligned}
 &= 13(4^{2n+1}) + 3(13n) \\
 &= 13(4^{2n+1} + 3n) \Rightarrow \text{multiple of 13}
 \end{aligned}$$

The hypothesis is proved.

4) Show that $n^4 - 4n^2$ is divisible by 3 for $n \geq 0$

Basis of induction

Consider $n = 0$

$$n^4 - 4n^2 = 0 - 0 = 0 \Rightarrow 3 \times 0 \Rightarrow \text{Divisible by 3}$$

Hence proved.

Inductive step

Let us assume that $n^4 - 4n^2$ is divisible by 3 for n , and try to prove the same for $n=n+1$

$$\therefore n^4 - 4n^2 \Rightarrow (n+1)^4 - 4(n+1)^2$$

Using induction hypothesis,

$$\begin{aligned}
 &(n+1)^4 - 4(n+1)^2 - n^4 + 4n^2 \quad ((n+1) - n^{\text{th}} \text{ term} = \text{a multiple of 3}) \\
 &= n^4 + 4n^3 + 6n^2 + 4n + 1 - 4n^2 - 8n - 4 - n^4 + 4n^2 \\
 &= n^4 + 4n^3 + 2n^2 - 4n - 3 - n^4 + 4n^2 \\
 &= 4n^3 + 6n^2 - 4n - 3 \\
 &= 4n(n^2 - 1) + 6n^2 - 3
 \end{aligned}$$

$$= \underbrace{4n(n+1)(n-1)}_{\downarrow} + 6n^2 - 3 \Rightarrow \text{Multiple of 3 for all } n \geq 0$$

Divisible by 3

FINITE AUTOMATA

Basic Definitions

Alphabet (Σ)

An alphabet is a finite, non empty set of symbols.

Example: $\Sigma = \{0,1\}$ → Binary alphabet

$\Sigma = \{a,b,c,\dots,z\}$ → Lower - case letters set

$\Sigma = \{1, 2, 3\}$ → Digits

String (w)

A string is a finite sequence of symbols chosen from some alphabet.

Example:

01101 is a string from $\Sigma = \{0,1\}$

aa, bb, ab, ba are strings from $\Sigma = \{a, b\}$

Empty string (ϵ) / Null string / λ / \wedge

An empty string is the string with zero occurrences of symbols.

Length of a string ($|\omega|$)

Let ω be the string, then the length of the string, ω is the number of symbols composing the string.

Example:

$\omega = 010 ; |\omega| = 3$

$\omega = \epsilon ; |\omega| = 0$

$\omega = abcba ; |\omega| = 5$

Concatenation of strings

The concatenation of two strings ' ω ' and 'v' is the string obtained by appending the symbols of v to the right end of ω .

Example:

$$\omega = 010$$

$$v = 111$$

$$\omega v = 010111$$

Power of an alphabet (\sum^k)

If \sum is an alphabet, we can express the set of all string of a certain length from that alphabet by using an exponential notation.

We define, \sum^k to be the set of strings of length k, each of whose symbol is in \sum

Example:

$$\sum^0 = \{\epsilon\}$$

$$\text{Let } \sum = \{a, b, c\}$$

$$\sum^0 = \{\epsilon\}$$

$$\sum^1 = \{a, b, c\}$$

$$\sum^2 = \{aa, bb, cc, ab, ac, ba, bc, ca, cb\}$$

$$\sum^3 = \{aaa, bbb, ccc, aab, aac, aba, aca, abb, acc, abc, acb, bba, bbc, bab, bcb, baa, bcc, \dots\}$$

Reversing a string (ω^R)

The reverse of the string is obtained by writing the string in reverse order.

Example:

$$\omega = \{abc\}$$

$$\omega^R = \{cba\}$$

Kleene closure (\sum^*)

Let \sum be an alphabet. Then the kleen closure, \sum^* denotes the set of all strings over the alphabet, \sum

Example:

$$1) \text{ Let } \Sigma = \{a, b\}$$

$$\Sigma^* = \{\epsilon, a, b, aa, bb, ab, ba, aaa, aab, \dots\}$$

$$2) \text{ Let } \Sigma = \{1\}$$

$$\Sigma^0 = \{\epsilon\}$$

$$\Sigma^1 = \{1\}$$

$$\Sigma^2 = \{11\}$$

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$$

$$= \{\epsilon, 1, 11, 111, \dots\}$$

Kleene plus / Positive closure (Σ^+)

Let Σ be an alphabet. Then the positive closure, Σ^+ denotes the set of all strings over the alphabet, Σ except null string (ϵ).

Example:

$$\text{Let } \Sigma = \{1, 0\}$$

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots$$

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

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

$$\Sigma^3 = \{111, 000, 110, 101, 011, \dots\}$$

$$\therefore \Sigma^+ = \{1, 0, 11, 00, 10, 01, 111, 000, 101, \dots\}$$

$$\Sigma^+ = \Sigma^* - \{\epsilon\}$$

Substring

The string, v if it appears within another string ω , then v is called as the substring of ω .

Example:

Let $v = \{111\}$; $\omega = \{010111\}$

$\therefore v$ is a substring of ω

Let $\omega = \{1, 2, 3\}$

Prefixes = $\{\epsilon, 1, 12, 123\}$

Suffixes = $\{\epsilon, 3, 23, 123\}$

Palindrome

A palindrome is a string which is same when read in backward or forward direction.

$\omega = \omega^R \Rightarrow \omega$ is a palindrome

Example: $\omega = \{1001\}$

$\therefore \omega^R = \{1001\}$ equal $\Rightarrow \omega$ is a palindrome.

Language (L)

Alphabet \rightarrow finite set of symbols

String \rightarrow collection of alphabets

Language \rightarrow collection of appropriate strings

A set of strings taken from an alphabet is called a language.

Example:

$$1) \sum = \{a, b\}$$

$L = \{a^n b, n \geq 0\} = \{b, ab, aab, aaab, \dots\}$

$$2) \sum = \{0, 1\}$$

$L = \{\text{Set of strings ending with } 1\}$

$= \{\epsilon, 11, 011, 0011, 1011, 0111, 101111, \dots\}$

Properties of String Operations

1. Concatenation of a string is always associative.

$$(uv) w = u(vw)$$

2. If u and v are two strings, then the length of this concatenation is the sum of the lengths of the individual lengths.

$$|xy| = |x| + |y|$$

Example: Let $x = 10$

$$y=101$$

Then $xy=10101 \Rightarrow$ Concatenation.

$$|xy| = |x| + |y|$$

$$= 2 + 3$$

$$= 5$$

Operations of Languages

1. Product or concatenation ($L_1 L_2$)

$$L_1 = \{b, ab, aab, \dots\} = \{a^n b, n \geq 0\}$$

$$L_2 = \{1, 01, 001, \dots\} = \{0^n 1, n \geq 0\}$$

$$L_1 L_2 = \{a^n b 0^n 1, a^n b \in L_1 \text{ & } 0^n 1 \in L_2\}$$

2. Reversal (L^R)

The reversal of a language is a set of all string reverses.

$$L^R = \{w^R / w \in L\}$$

3. Kleen Star/Star Closure (L^*)

$$L^* = L^0 U L^1 U L^2 U \dots$$

4. Positive Closure

$$L^+ = L^* - \{\epsilon\}$$

$$= L^1 U L^2 U L^3 U \dots$$

Other Operation on Set/Language

1. Union

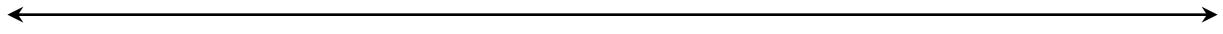
$$L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$$

2. Intersection

$$L_1 \cap L_2 = \{w \mid w \in L_1 \text{ and } w \in L_2\}$$

3. Difference:

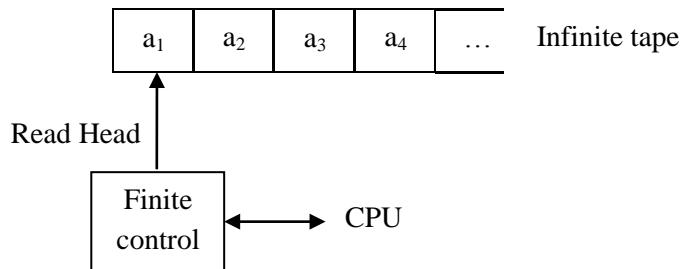
$$L_1 - L_2 = \{w \mid w \in L_1 \text{ and } w \notin L_2\}$$



Definition of Finite Automata [FA]

Finite Automata is a mathematical model of system with certain input and it is processed through various intermediate states and finally gives certain output.

Model of FA



Input Tape

- It is divided into number of cells.
- Each cell holds one symbol.

Read Head

- Reads one cell at a time and moves ahead.

Finite Control

- Acts like a CPU.
- Depending on current state and input symbol read from tape, it changes state.

Formal Definition of FA

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

FA consist of 5 Tuples,

$Q \rightarrow$ Set of finite states.

$\Sigma \rightarrow$ Set of input alphabets

$\delta \rightarrow$ Transition function $[Q \times \Sigma \rightarrow Q]$

$q_0 \rightarrow$ Start state

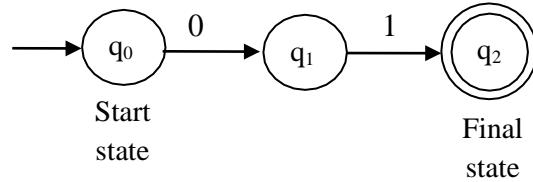
$F \rightarrow$ Set of final states. $[F \subseteq Q]$

Representation of FA

1. Transition Diagram
2. Transition Table

Transition Diagram

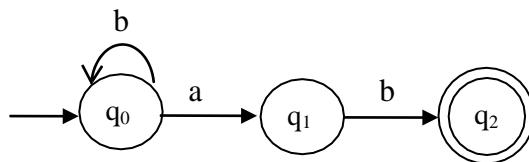
It can be directed graph with vertices of a graph corresponding to states and edges indicates transition from one state to another



- Start state indicated by circle.
- Start transition indicated by arrow.
- Final state indicated by double circle.

Transition Table

It is a tabular listing of the transition function which by implication tells us, set of states and input alphabet.



State ↓	a	b
Input →		
→ q ₀	q ₁	q ₀
q ₁	∅	q ₂
*q ₂	∅	∅

$$\begin{aligned}\delta(q_0, a) &= q_1 \\ \delta(q_0, b) &= q_0 \\ \delta(q_1, b) &= q_2\end{aligned}$$

Applications

1. Design of digital circuits.
2. String searching
3. Communication protocols for information exchange.
4. Lexical analysis phase of a compiler.

Types

1. Deterministic Finite Automata [DFA]
2. Non-Deterministic Finite Automata [NFA]

DETERMINISTIC FINITE AUTOMATA [DFA] /FA

- The word deterministic refers to the fact the transition is deterministic.
- There is only one path for specific input from current state to next state.

DFA is defined by 5 tuples,

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

where,

$Q \rightarrow$ Finite set of states.

$\Sigma \rightarrow$ Finite set of input symbols.

$q_0 \rightarrow$ Initial state

$\delta \rightarrow$ Transition function

$F \rightarrow$ Final state / Accepting state

Properties of DFA

- There is only one path for an input from current state to next state.
- It contains only one final state.

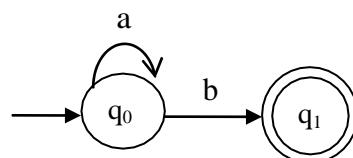
PROBLEMS

1. Design a DFA for the language, $L = \{a^n b; n \geq 0\}$

If $n=0 \Rightarrow L=a^0 b=b$ $n=3 \Rightarrow L=a^3 b=aaab$

$n=1 \Rightarrow L=a^1 b=ab$ $n=4 \Rightarrow L=a^4 b=aaaab$

$n=2 \Rightarrow L=a^2 b=aab$ $n=5 \Rightarrow L=a^5 b=aaaaab$



Here, $Q=\{q_0, q_1\}$;

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

$$q_0 = \{q_0\}$$

Transition function,

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

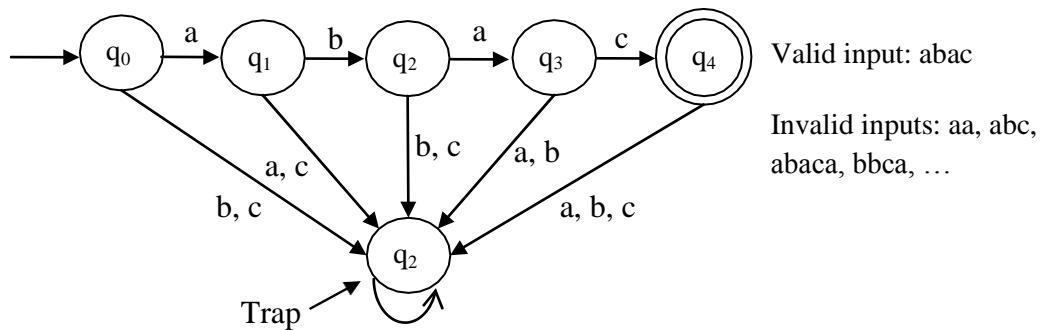
$$\delta(q_0, b) = \{q_1\}$$

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

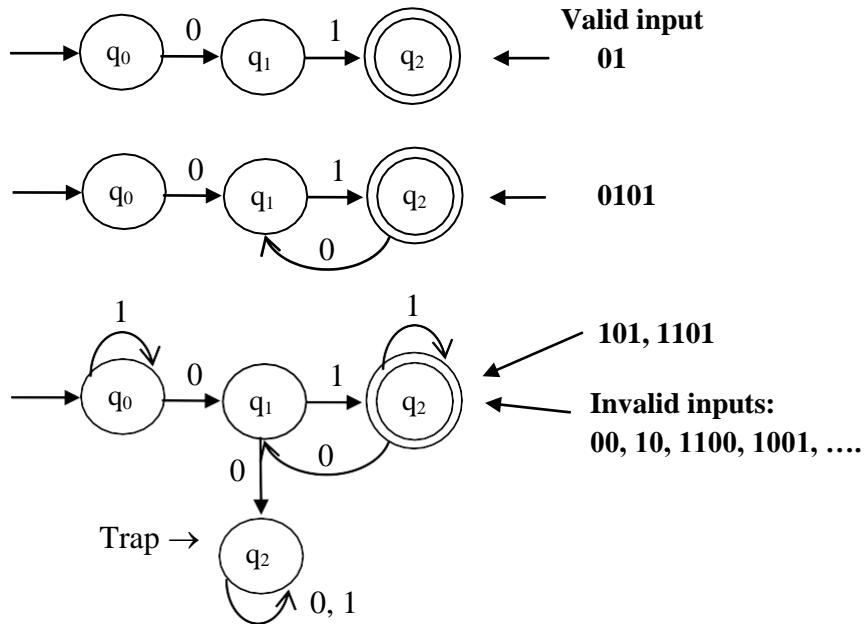
$$\delta(q_1, b) = \emptyset$$

$$F = \{q_1\}$$

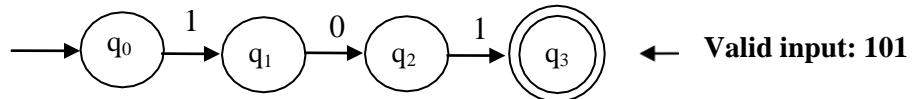
2 Design a DFA that accept $L = \{w / w \in (a, b, c), \text{ and } w \text{ contains only abac}\}$

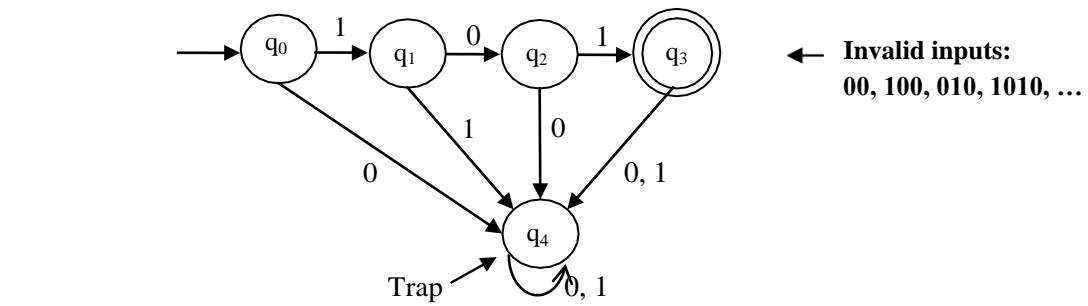


3 Design a DFA that accept $L = \{w \in (0, 1)^*, \text{ where every 0 in } w \text{ has 1 immediately to its right}\}$.

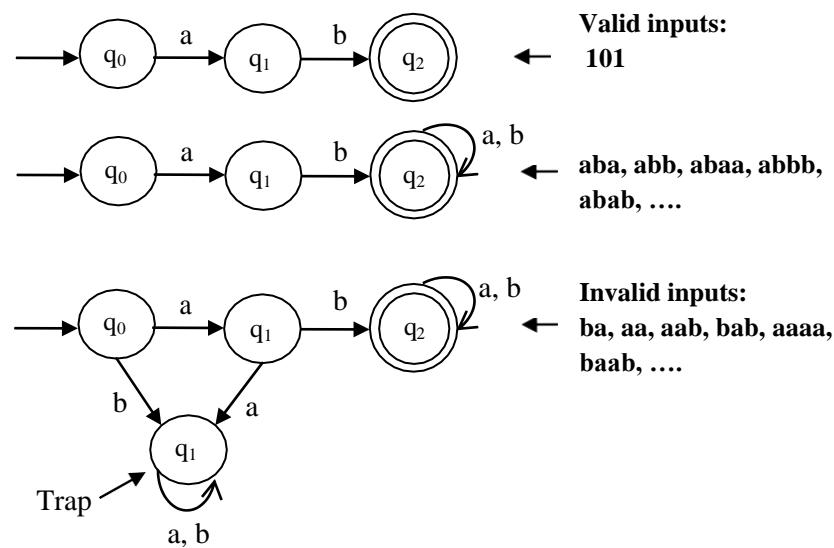


4 Design a DFA that accept „101“.

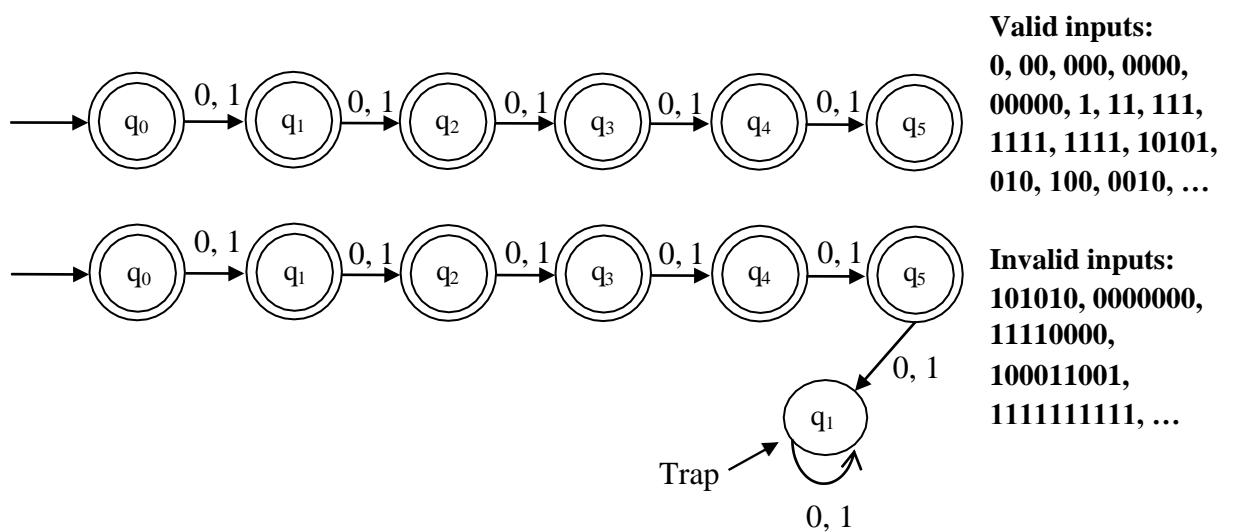




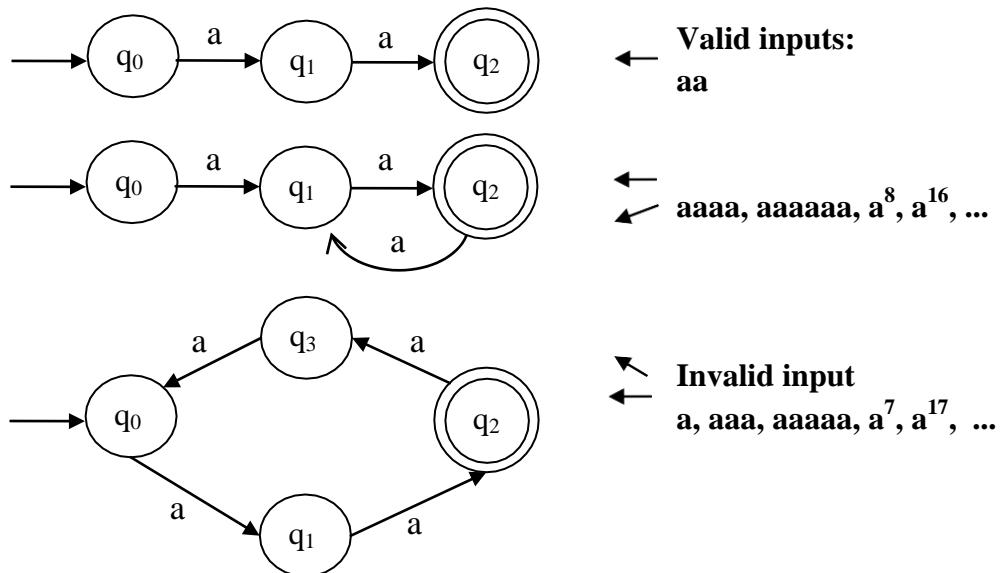
5. Design a DFA that accepts string that starts with „ab“.



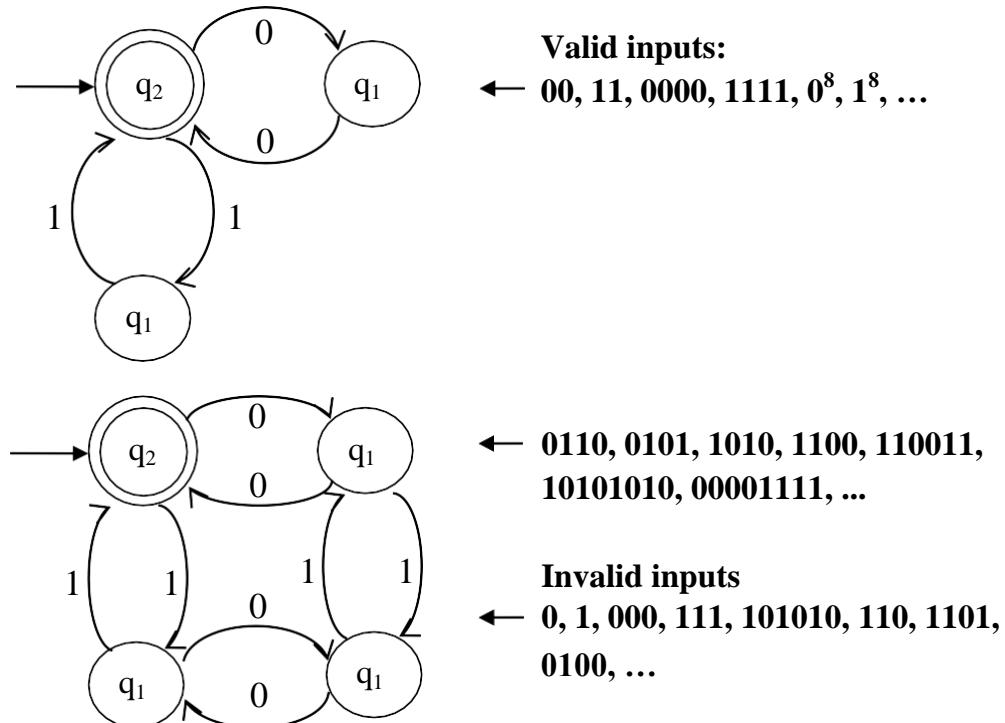
6. Design a DFA that accepts all strings of length atmost 5 over $\Sigma = \{0, 1\}$



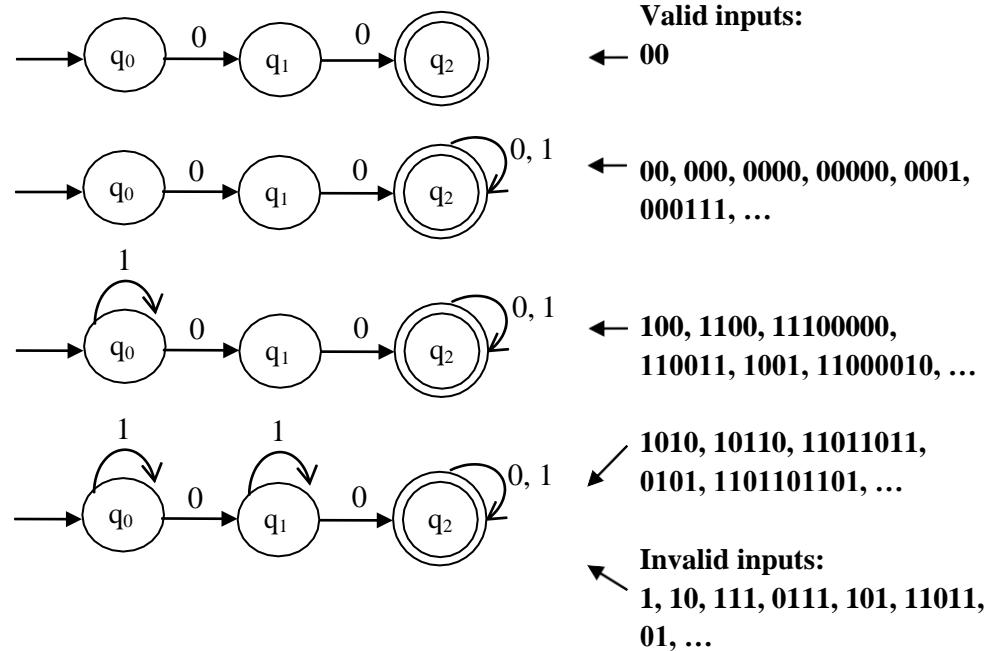
7. Design a DFA that accepts even number of a's over $\Sigma = \{a\}$



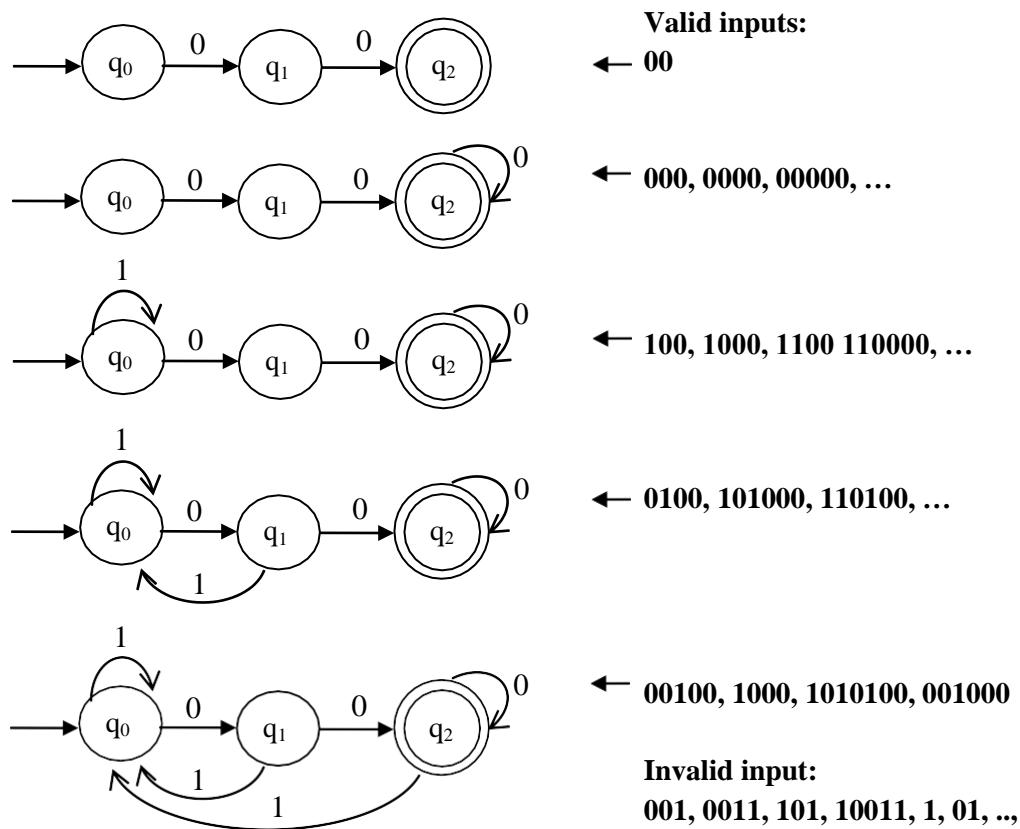
8. Design a DFA that accepts a language, L having even number of 0s and 1s.



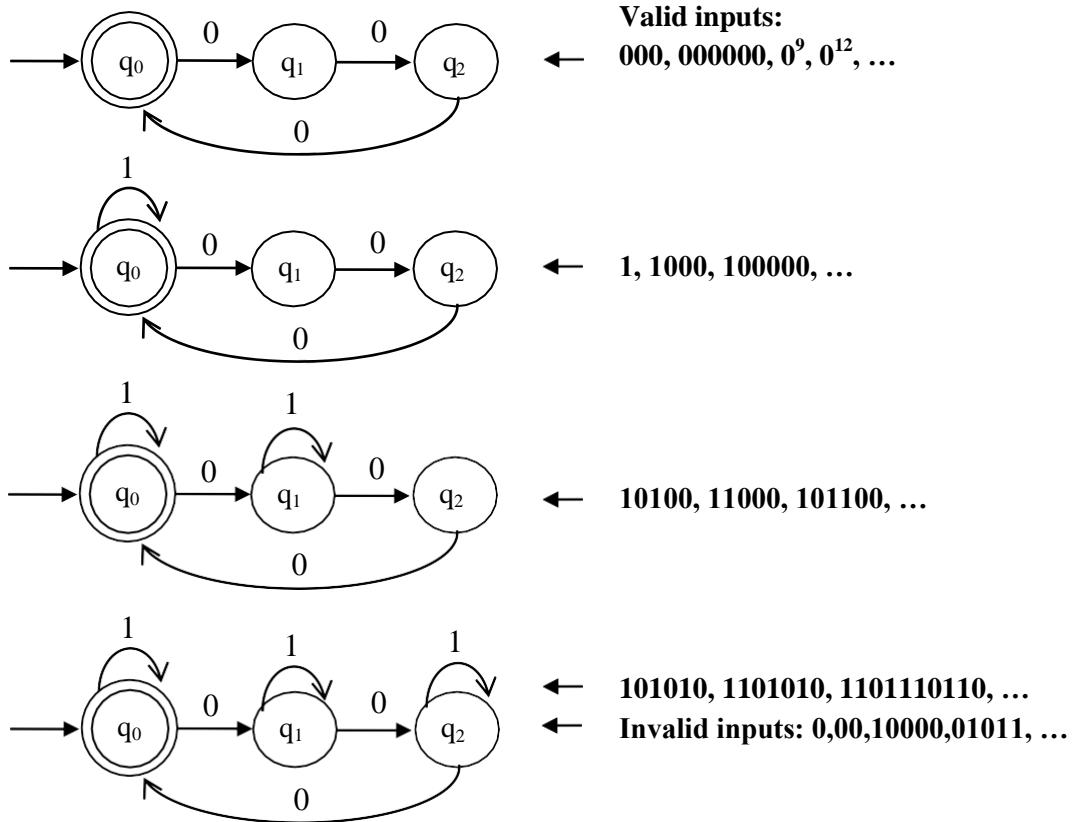
9. Design a DFA that accepts all strings containing atleast two zeroes.



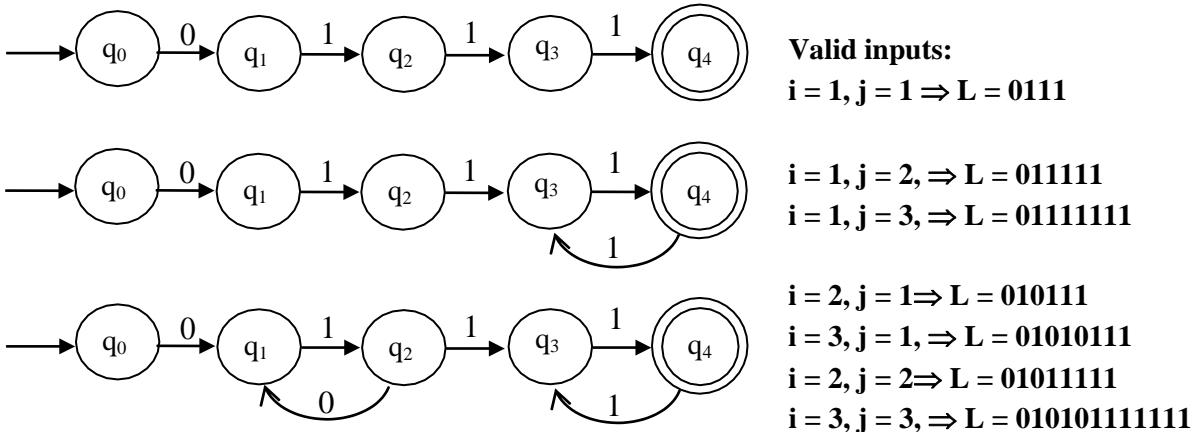
10. Design a DFA that accepts strings ending with „00“ over $\Sigma = \{0,1\}$

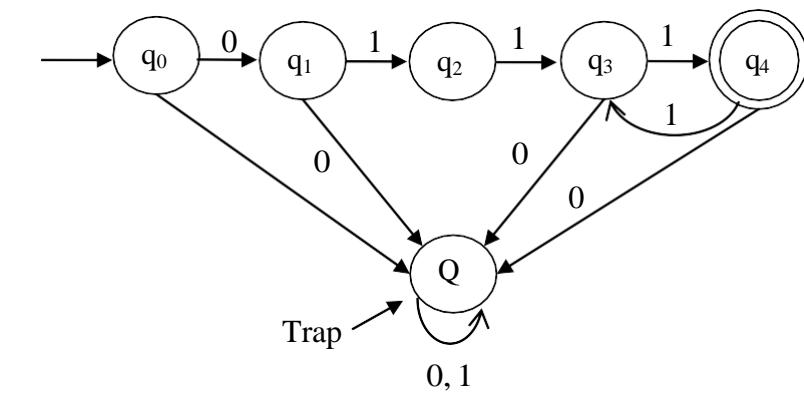


11. Design a DFA that accepts a language, L which has the number of zeroes is of multiples of 3.



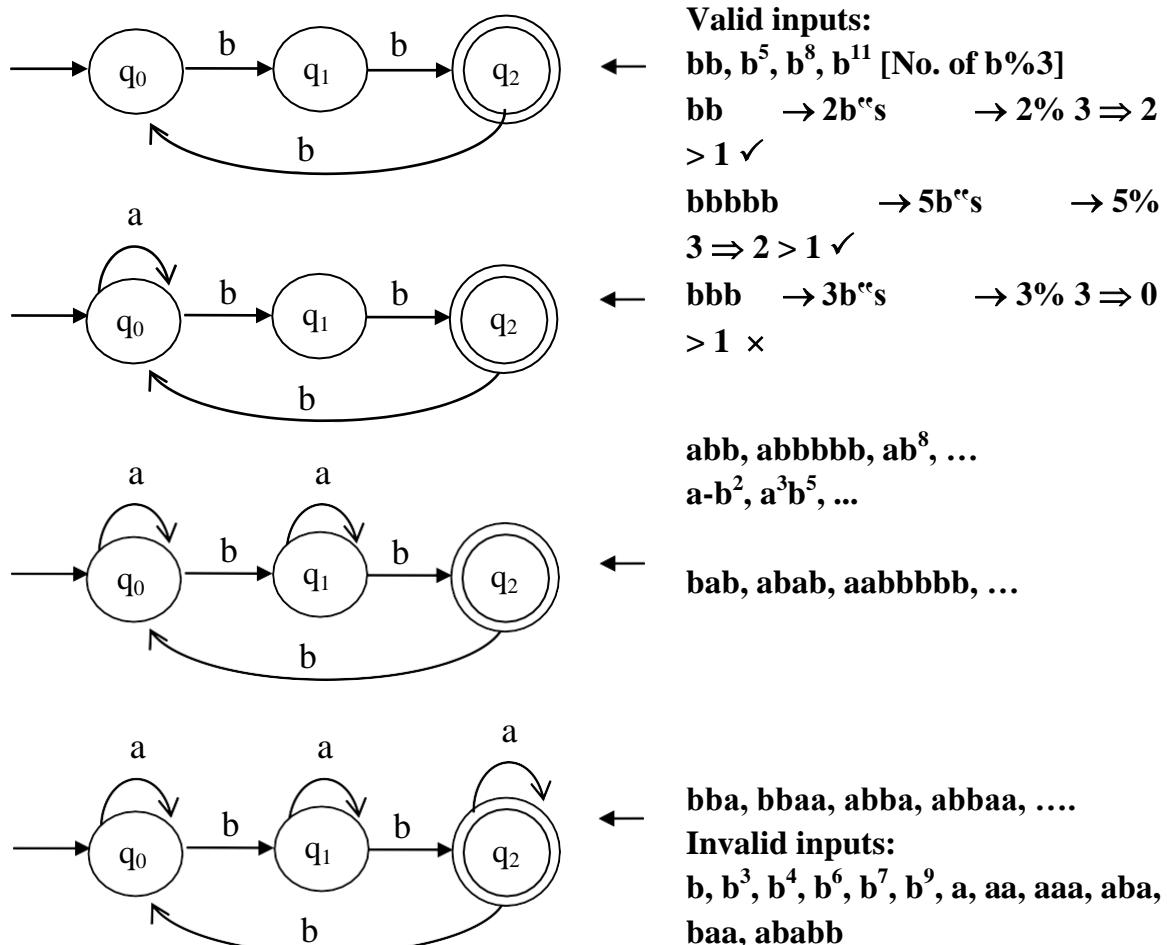
12. Design a DFA that accepts $L = \{(01)^i 1^{2j}, i \geq 1; j \geq 1\}$



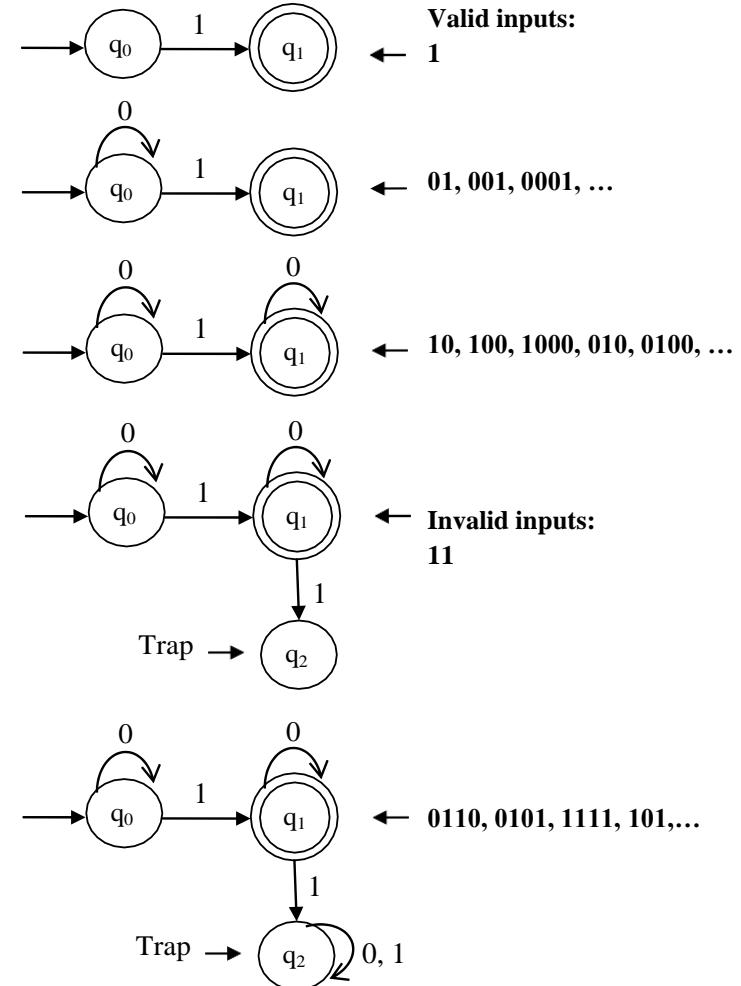


Invalid inputs:
1, 00, 0100, 0110, 01110,
011110, 1000, 101, ...

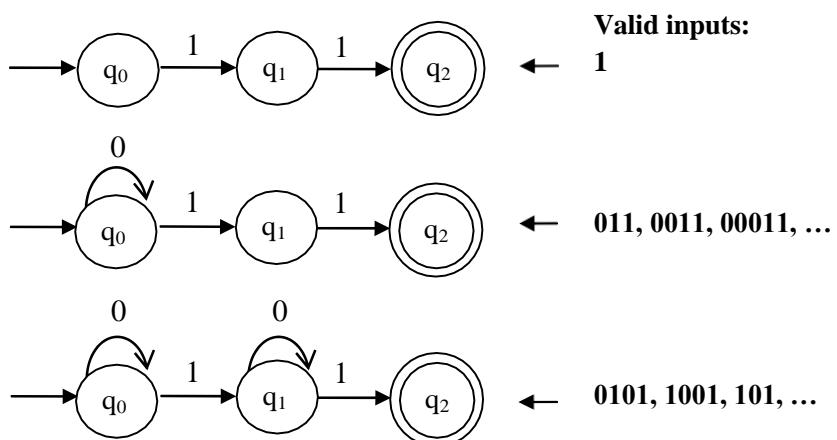
13. Design a DFA that accepts $L = \{\omega \in (a, b)^* / n_b(\omega) \bmod 3 > 1\}$

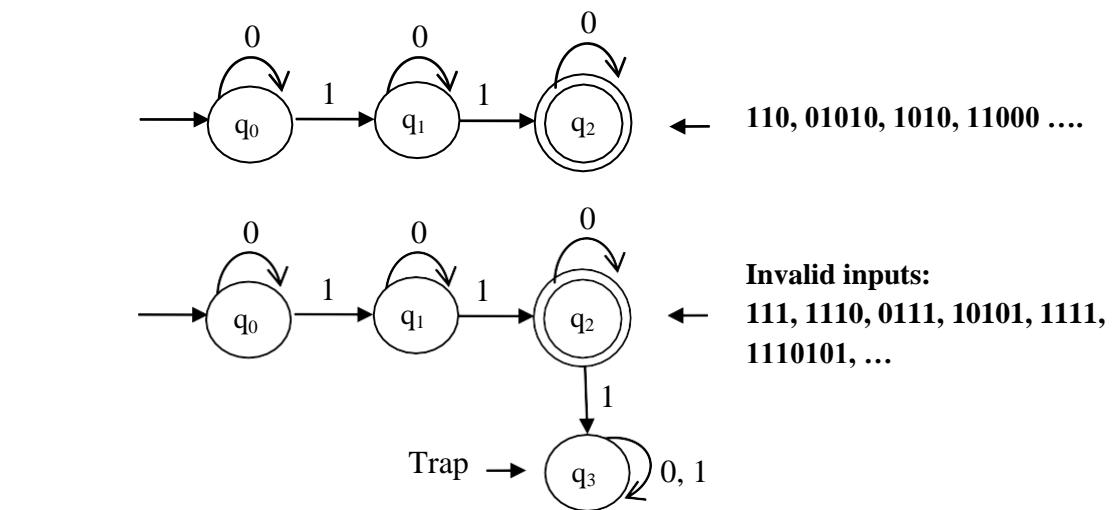


14. Design a DFA accepting strings containing exactly one 1 over $\Sigma = \{0, 1\}$

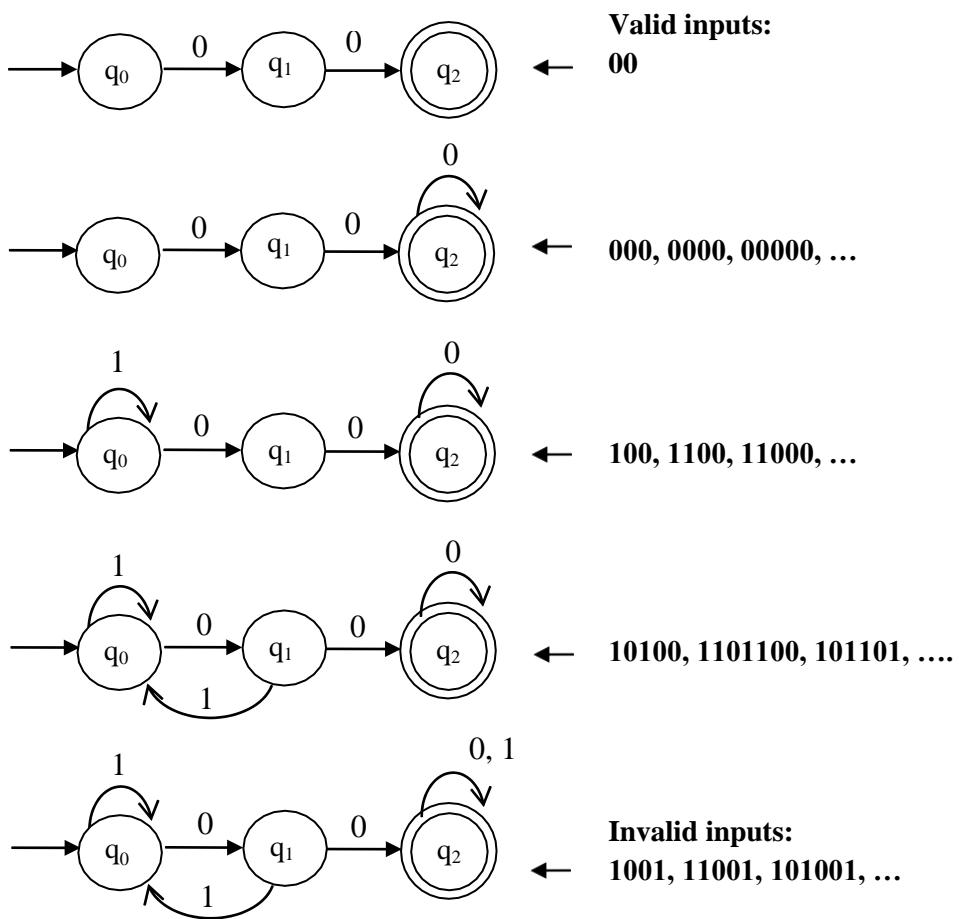


15. Design a DFA that accepts all strings with exactly two 1s over $\Sigma = \{0, 1\}$

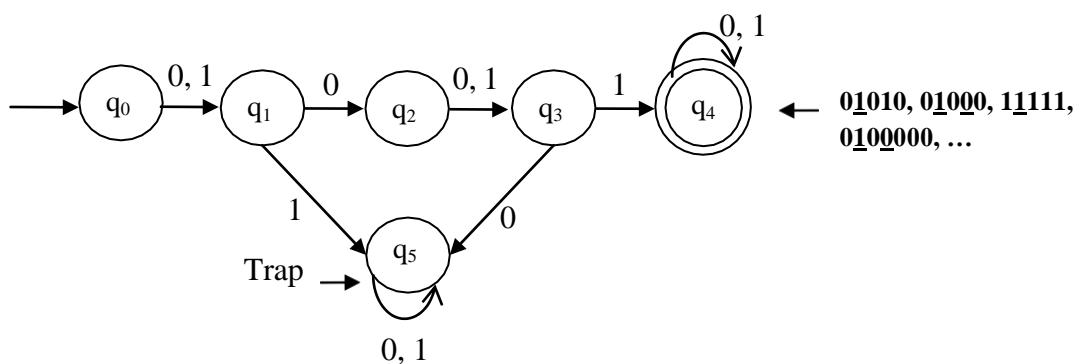
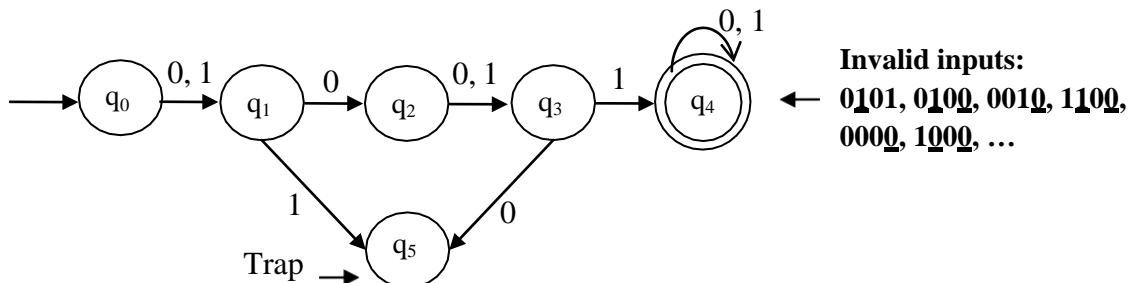
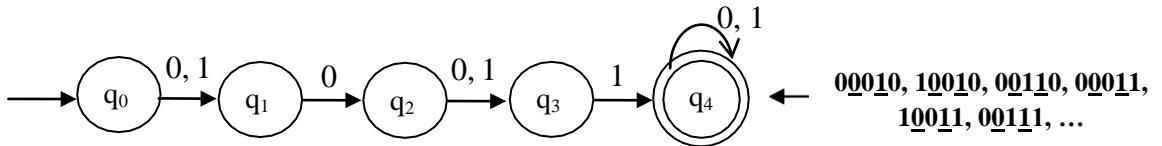
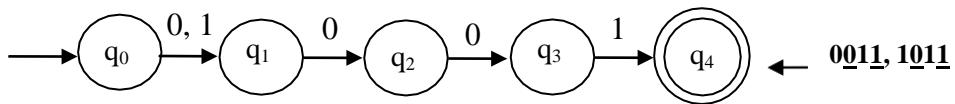
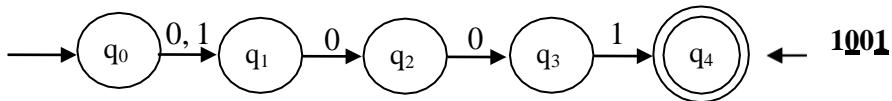
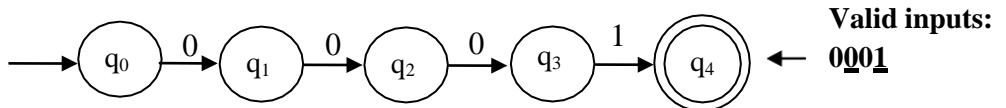




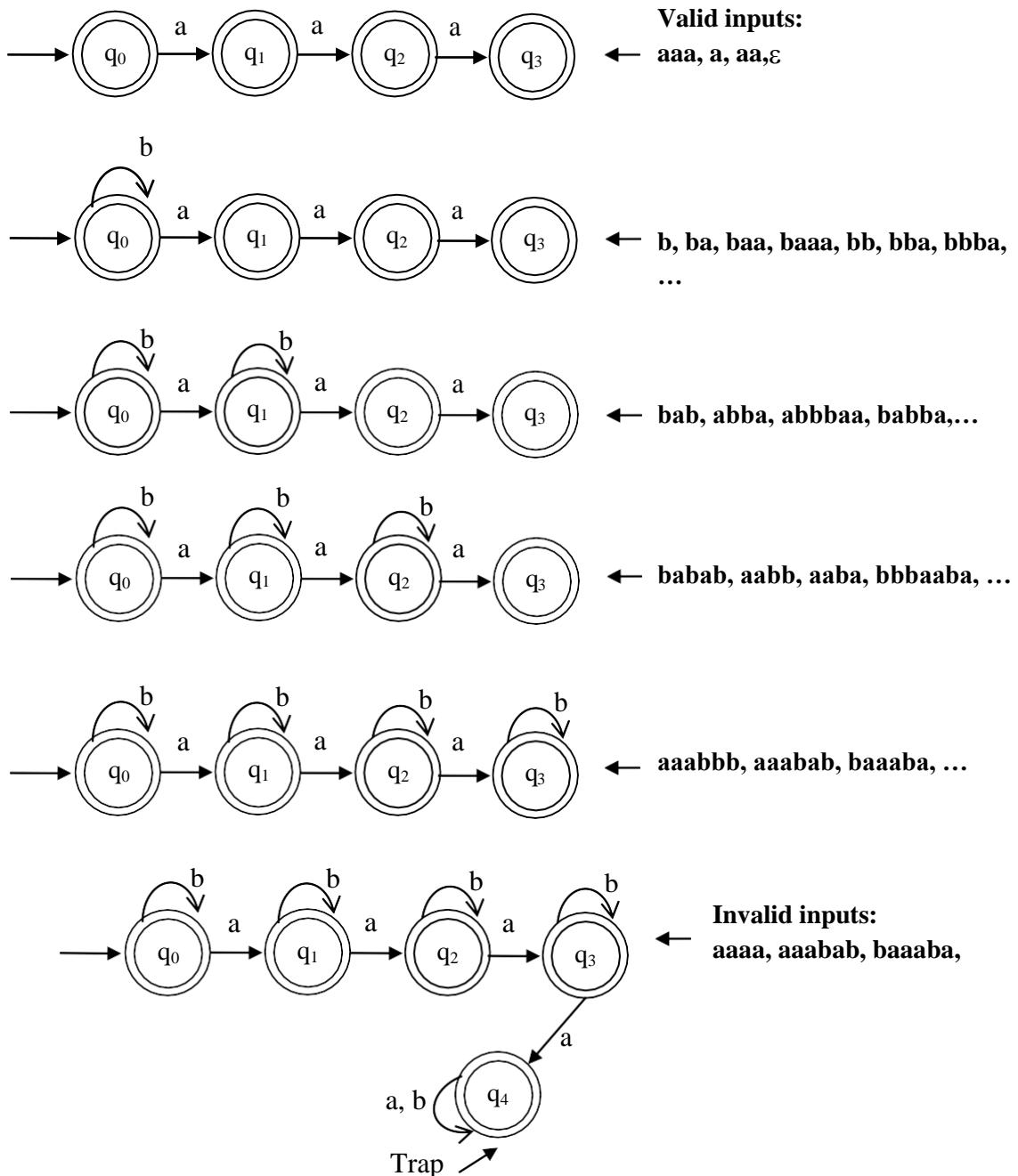
16. DFA that accepts strings containing „00“ as substring.



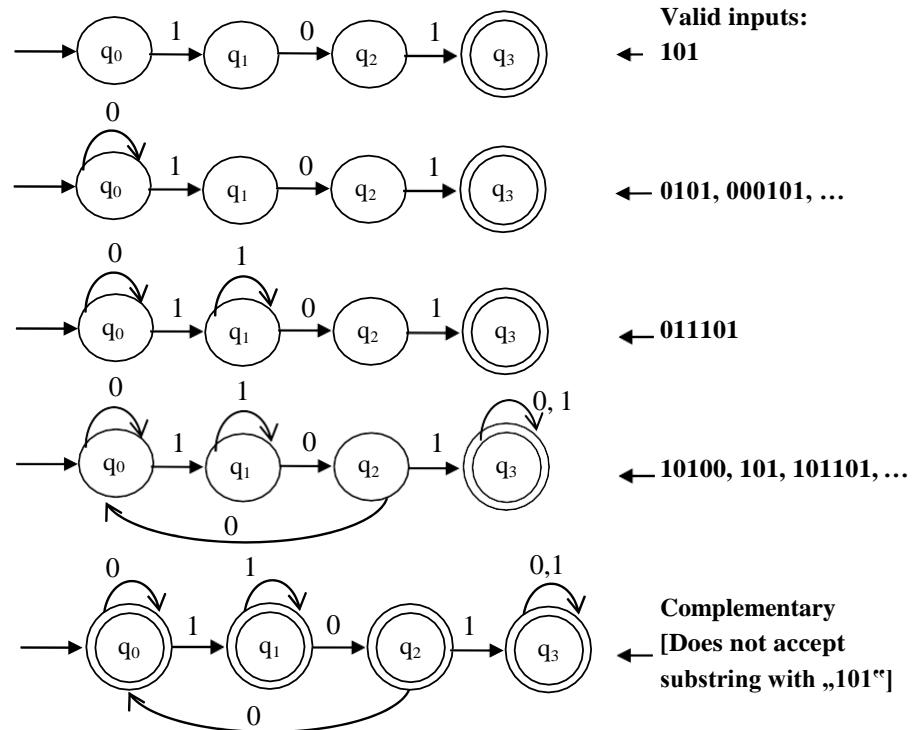
17. Design a DFA that accepts string, ω such that its second symbol is zero and fourth symbol is 1.



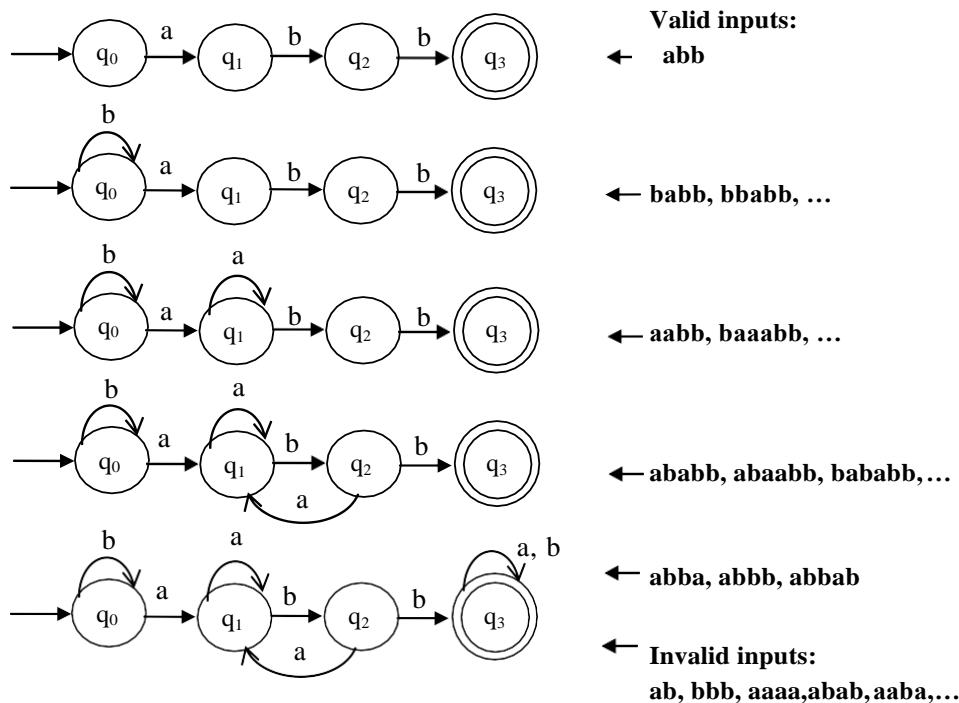
18. Design a DFA that accepts atmost 3 a's.



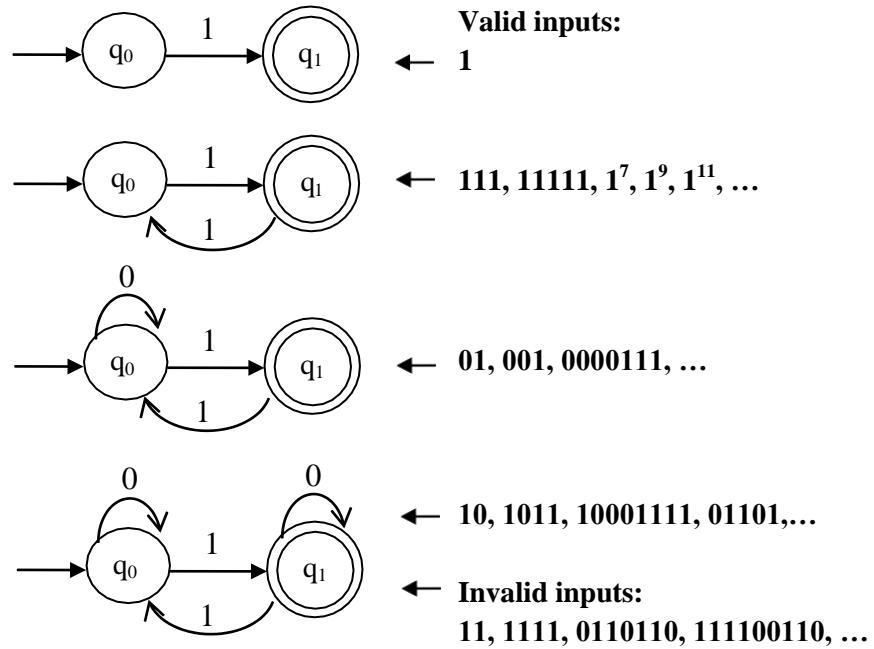
19. Design a DFA that accepts all strings containing substring „101“.



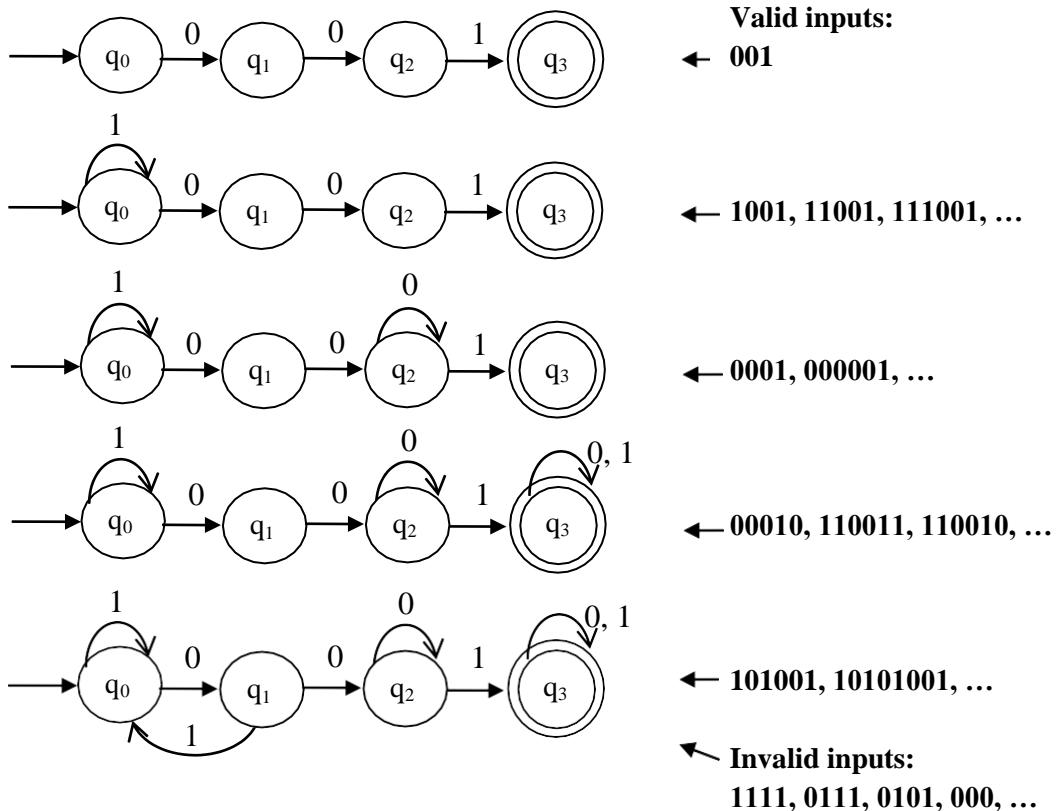
20. Design a DFA to check whether a string over $\Sigma = \{a,b\}$ contains „abb“ is accepted.



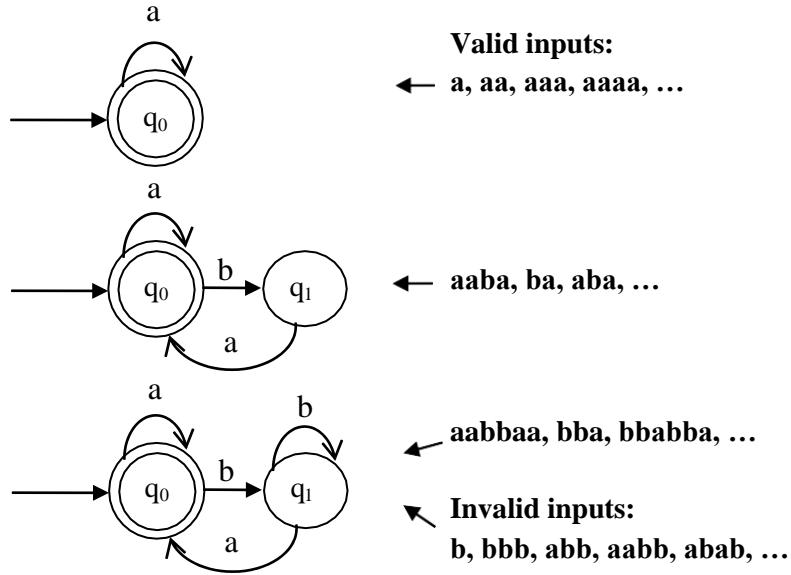
21. Design a DFA that accepts odd number of ones over $\Sigma = \{0, 1\}$



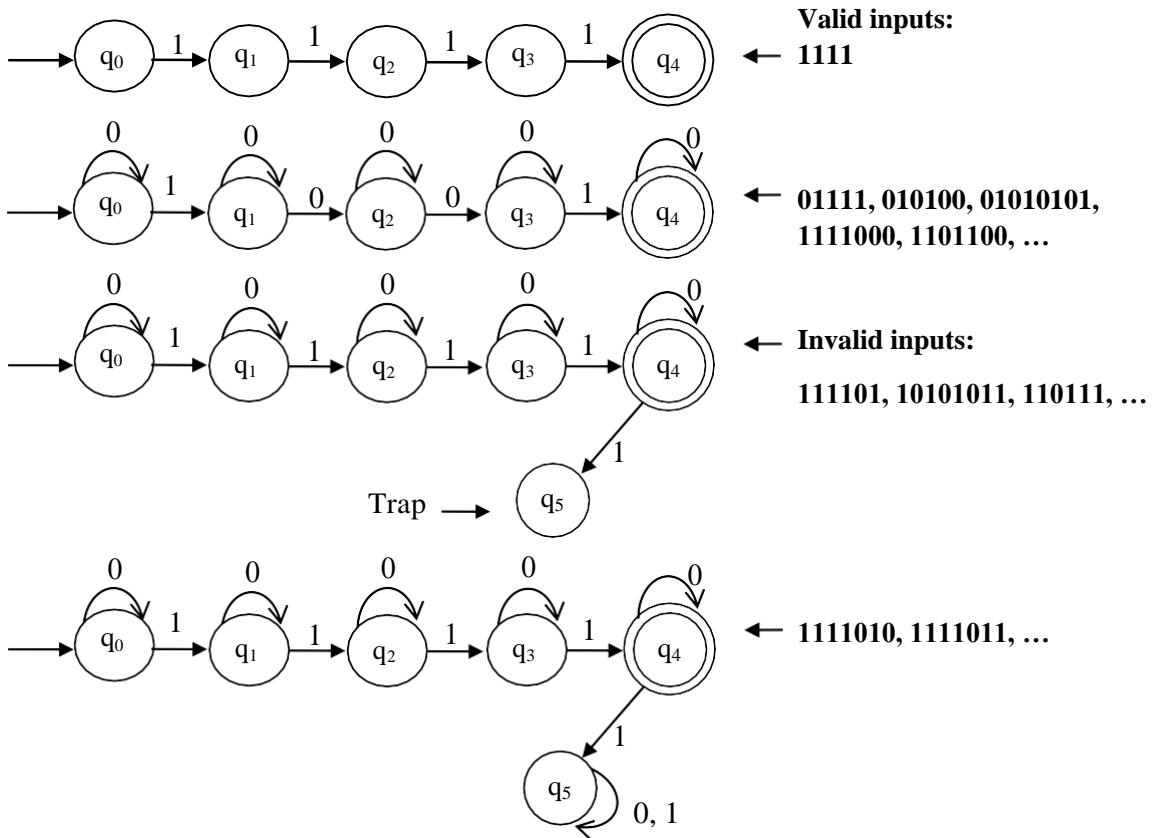
22. Design a DFA that accepts a substring “001”.



23. Design a DFA that recognizes words which do not end in „b“ over $\Sigma = \{a, b\}$



24. Design a DFA that accepts exactly 4 ones over $\Sigma = \{0, 1\}$

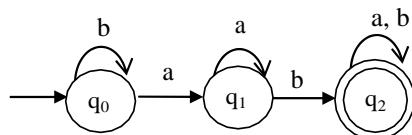


Extended transition function: $(\delta) \leftarrow \Delta \text{ Cap/Hat}$

- The extended transition function, $\hat{\delta}$ takes two parameters, state and string.
- It is a mapping from $Q \times \Sigma^* \rightarrow Q$
- Formally $\hat{\delta}$ can be defined as
 - $\hat{\delta}(q_0, \epsilon) = q_0$
 - $\hat{\delta}(q_0, \omega a) = \hat{\delta}(\hat{\delta}(q_0, \omega), a)$

PROBLEMS

1) Consider the transition diagram

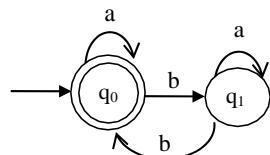


Check whether the input string, "aab" is accepted by the finite automata.

$$\begin{aligned}
 \hat{\delta}(q_0, \epsilon) &= q_0 \\
 \hat{\delta}(q_0, aab) &= \hat{\delta}(\hat{\delta}(q_0, aa), b) \\
 &= \hat{\delta}(\hat{\delta}(\hat{\delta}(q_0, \epsilon), a), b) \\
 &= \hat{\delta}(\hat{\delta}(\hat{\delta}(q_0, a), a), b) \\
 &= \hat{\delta}(\hat{\delta}(q_1, a), b) \\
 &= \hat{\delta}(q_1, b) \\
 &= q_2 \Rightarrow \text{Final state}
 \end{aligned}$$

Hence the given string "aab" is accepted by FA.

2) Consider the FA



Show how the string $\omega_1 = aabba$ and $\omega_2 = aba$ is processed.

1) $\omega_1 = aabba$

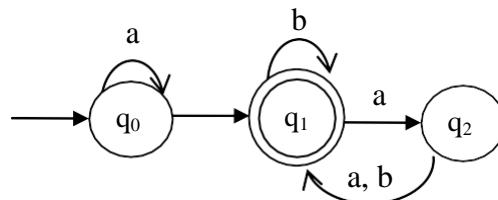
$$\begin{aligned}\hat{\delta}(q_0, aabba) &= \hat{\delta}(q_0, abba) \\ &= \hat{\delta}(q_0, bba) \\ &= \hat{\delta}(q_1, ba) \\ &= \hat{\delta}(q_0, a) \\ &= q_0 \Rightarrow \text{Final state} \Rightarrow \text{string is accepted}\end{aligned}$$

2) $\omega_2 = aba$

$$\begin{aligned}\hat{\delta}(q_0, aba) &= \hat{\delta}(q_0, ba) \\ &= \hat{\delta}(q_1, a) \\ &= q_1 \Rightarrow \text{Not a final state} \Rightarrow \text{string not accepted}\end{aligned}$$

3) Check whether the strings "ababba", "baab" are accepted by the DFA?

AU – May / June, Nov / Dec 2009



1) $\omega_1 = ababba$

$$\begin{aligned}\hat{\delta}(q_0, ababba) &= \hat{\delta}(\hat{\delta}(q_0, ababb), a) \\ &= \hat{\delta}(\hat{\delta}(\hat{\delta}(q_0, abab), b)a) \\ &= \hat{\delta}(\hat{\delta}(\hat{\delta}(\hat{\delta}(q_0, aba), b), b), a) \\ &= \hat{\delta}(\hat{\delta}(\hat{\delta}(\hat{\delta}(\hat{\delta}(q_0, ab), a), b), b), a) \\ &= \hat{\delta}(\hat{\delta}(\hat{\delta}(\hat{\delta}(\hat{\delta}(q_0, a), b), a), b), b), a) \\ &= \hat{\delta}(\hat{\delta}(\hat{\delta}(\hat{\delta}(\hat{\delta}(\hat{\delta}(q_0, \varepsilon), a), b), a), b), b), a) \\ &= \hat{\delta}(\hat{\delta}(\hat{\delta}(\hat{\delta}(\hat{\delta}(\hat{\delta}(q_0, a), b), a), b), b), a)\end{aligned}$$



$$\begin{aligned}
 &= \delta(\delta(\delta(\delta(\delta(q_0, b), a), b), b), a) \\
 &= \delta(\delta(\delta(\delta(q_1, a), b), b), a) \\
 &= \delta(\delta(\delta(\delta(q_2, b), b), b), a) \\
 &= \delta(\delta(q_1, b), a) \\
 &= \delta(q_1, a) \\
 &= \delta(q_3, \epsilon) \\
 &= q_3 \Rightarrow \text{Not a Final state} \Rightarrow \text{Not accepted}
 \end{aligned}$$

2) $\omega_2 = baab$

$$\begin{aligned}
 \hat{\delta}(q_0, baab) &= \hat{\delta}(\hat{\delta}(q_0, baa), b) \\
 &= \hat{\delta}(\hat{\delta}(\hat{\delta}(q_0, ba), a), b) \\
 &= \hat{\delta}(\hat{\delta}(\hat{\delta}(\delta(q_0, b), a), a), b) \\
 &= \hat{\delta}(\hat{\delta}(\hat{\delta}(\delta(\delta(q_0, \epsilon), b), a), a), b) \\
 &= \hat{\delta}(\hat{\delta}(\delta(\delta(q_0, b), a), a), b) \\
 &= \hat{\delta}(\delta(q_1, a), a), b) \\
 &= \delta(q_2, a), b) \\
 &= \delta(q_1, b) \\
 &= q_1 \Rightarrow \text{Final state} \Rightarrow \text{string accepted}
 \end{aligned}$$

NON DETERMINISTIC FINITE AUTOMATA

AU - Nov/Dec 2013

A NFA has 5 tuples defined as,

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

where,

$Q \rightarrow$ a finite set of states

$\Sigma \rightarrow$ A finite set of inputs

$\delta \rightarrow$ Transition function from $Q \times \Sigma \rightarrow Q$

$q_0 \rightarrow$ Start / initial state

$F \rightarrow$ A set of final / accepting states $[F \subseteq Q]$



Subset

Difference between DFA and NFA

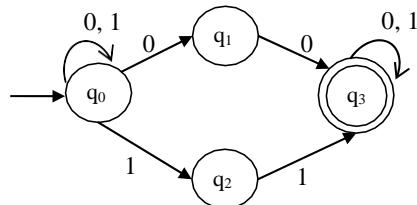
S.NO	DFA	NFA
1.	The transition is deterministic	The transition is non-deterministic
2.	It has more number of states than NFA	It has less number of states. $\lceil \leq 2^Q \rceil$
3.	It provides only one state transition for a given input	It may lead to more than one states for given input
4.	It has only one final state	It has more than one final states
5.	More difficult to design	It is easy to design
6.	It has a single path for an input from a state	It may have several paths for a single input from a state. Backtracking is required.

PROBLEMS

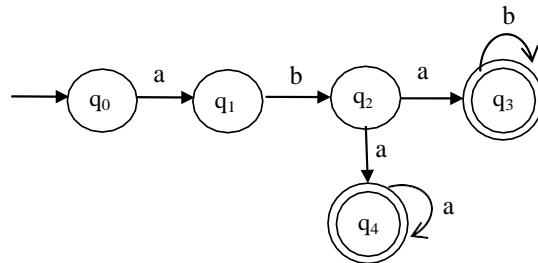
- 1) Sketch the NFA state diagram for $M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_3\})$ with δ given as

δ	Σ	0	1
Q			
$\rightarrow q_0$	(q_0, q_1)	(q_0, q_2)	
q_1	q_3	\emptyset	
q_2	\emptyset	q_3	
$* q_3$	q_3	q_3	

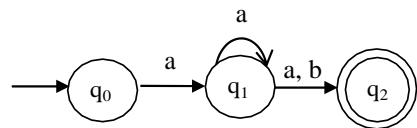
Solution:



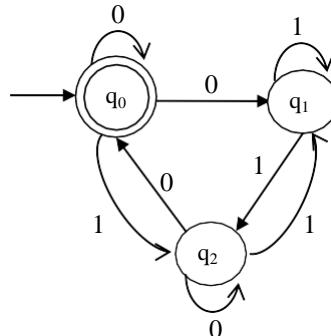
2 Design a NFA with no more than five states for the set $\{abab^n : n \geq 0\} \cup \{aba^n : n > 0\}$.



3 Determine a NFA that accepts $L = (aa^*(a + b))$.



4 For NFA shown, check whether „0100“ is accepted or not.



$\delta :$

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

Solution:

$$\hat{\delta}(q_0, 0100)$$

$$\delta(q_0, 0) = \{q_0, q_1\} = \delta(\{q_0, q_1\}, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$$

$$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1)$$

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

$$\begin{aligned}
 &= \delta(q_0, 1) \cup \delta(q_1, 1) \\
 &= \{q_2\} \cup \{q_1, q_2\} \\
 &= \{q_1, q_2\} \\
 \delta(q_0, 010) &= \delta(\delta(q_0, 01), 0) \\
 &= \delta(\{q_1, q_2\}, 0) \\
 &= \delta(q_1, 0) \cup \delta(q_2, 0) \\
 &= \{\emptyset\} \cup \{q_0, q_2\} \\
 &= \{q_0, q_2\} \\
 \delta(q_0, 0100) &= \delta(\delta(q_0, 010), 0) \\
 &= \delta(\{q_0, q_2\}, 0) \\
 &= \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \{q_0, q_2\} \\
 &= \{q_0, q_1, q_2\} \Rightarrow \text{Since } q_0 \text{ is final state} \Rightarrow \text{string accepted}
 \end{aligned}$$

5 NFA with states $\{1, 2, 3, 4, 5\}$ and $\Sigma = \{a, b\}$ has δ as,

Calculate 1) $\delta(1, abab)$

2) $\delta(2, baba)$

Σ	a	b
$\rightarrow 1$	$\{1, 2\}$	$\{1\}$
2	$\{3\}$	$\{3\}$
3	$\{4\}$	$\{4\}$
4	$\{5\}$	\emptyset
5	\emptyset	$\{5\}$

Solution:

1) $\delta(1, abab)$

$$\delta(1, a) = \{1, 2\}$$

$$\delta(1, ab) = \delta(\delta(1, a), b)$$

$$= \delta(\{1, 2\}, b)$$

← →

$$= \delta(1, b) \cup \delta(2, b)$$

$$= \{1\} \cup \{3\}$$

$$= \{1, 3\}$$

$$\delta(1, aba) = \delta(\delta(1, ab), a)$$

$$= \delta(\{1, 3\}, a)$$

$$= \delta(1, a) \cup \delta(3, a)$$

$$= \{1, 2\} \cup \{4\}$$

$$= \{1, 2, 4\}$$

$$\delta(1, abab) = \delta(\delta(1, aba), b)$$

$$= \delta(\{1, 2, 4\}, b)$$

$$= \delta(1, b) \cup \delta(2, b) \cup \delta(4, b)$$

$$= \{1\} \cup \{3\} \cup \{\phi\} = \{1, 3\}$$

2) $\delta(2, baba) :$

$$\delta(2, b) = \{3\}$$

$$\delta(2, ba) = \delta(\delta(2, b), a)$$

$$= \delta(3, a)$$

$$= \{4\}$$

$$\delta(2, bab) = \delta(\delta(2, ba), b)$$

$$= \delta(4, b)$$

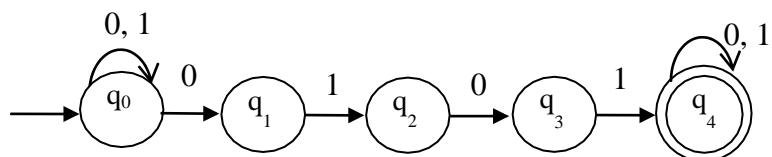
$$= \phi$$

$$\delta(2, baba) = \delta(\delta(2, bab), a)$$

$$= \delta(\phi, a)$$

$$= \phi$$

6 Design a NFA to accept strings containing the substring, "0101".



NFA: $M = (Q, \Sigma, \delta, q_0, F)$ where,

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

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

$$q_0 = \{q_0\}$$

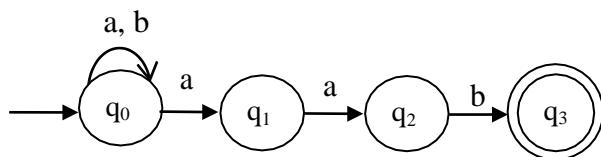
$$F = \{q_4\}$$

δ is given as →

δ

$\Sigma \backslash Q$	0	1
$Q \rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	\emptyset	$\{q_2\}$
q_2	$\{q_3\}$	\emptyset
q_3	\emptyset	$\{q_4\}$
$* q_4$	$\{q_4\}$	$\{q_4\}$

7) Construct a NFA that accepts $L = \{x \in \{a, b\}^* / x \text{ ends with 'aab'}\}$



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

δ is given as →

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

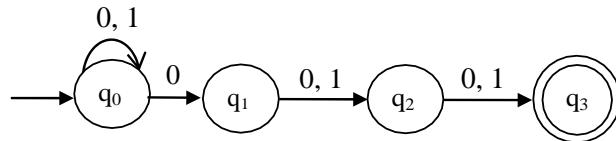
$$\Sigma = \{a, b\}$$

$$q_0 = \{q_0\}$$

$$F = \{q_3\}$$

$\Sigma \backslash Q$	a	b
$Q \rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{\emptyset\}$
q_2	\emptyset	$\{q_3\}$
$* q_3$	\emptyset	\emptyset

- 8 Design a NFA to accept strings over alphabet $\{0, 1\}$, such that the third symbol from right end is 0.



$$Q = \{q_0, q_1, q_2, q_3\} \quad \text{The transition function, } \delta \text{ is given as}$$

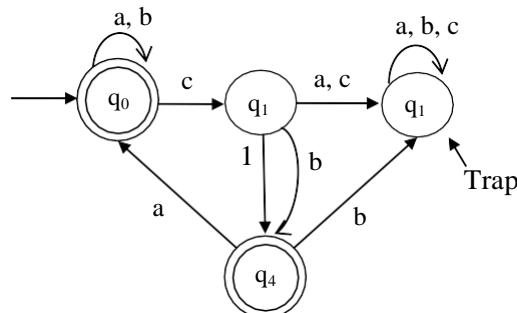
$$\Sigma = \{0, 1\}$$

$$q_0 = \{q_0\}$$

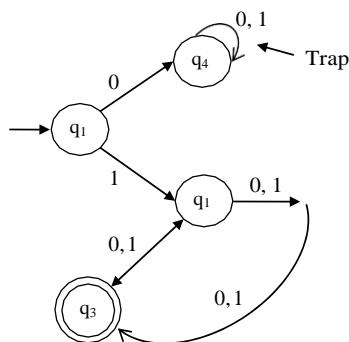
$$F = \{q_3\}$$

$\Sigma \backslash Q$	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
* q_3	\emptyset	\emptyset

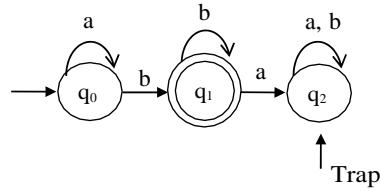
- 9 Construct a NFA for $L = \{x \in \{a, b, c\}^*: x \text{ contains exactly one } b \text{ immediately following } c\}$.



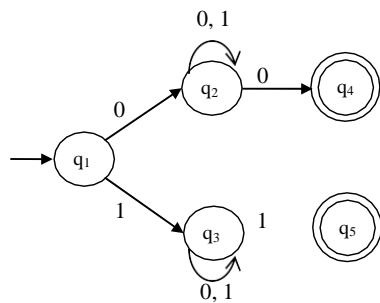
- 10 Construct a NFA that accept $L = \{x \in \{0, 1\}^*: x \text{ is starting with } 1 \text{ and } |x| \text{ is divisible by } 3\}$.



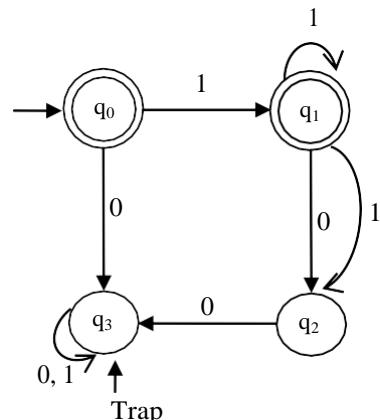
- 1) Design a NFA for $L = \{x \in \{a, b\}^* / x \text{ contains any number of } a\text{'s followed by atleast one } b\}$.



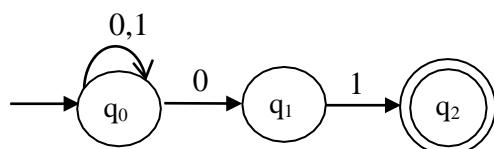
- 2) Design a NFA for a binary number where the first and the last digits are same.



- 3) Construct a NFA over $\{0, 1\}$ such that each „0“ is immediately preceded and immediately followed by 1.



- 4) Construct NFA for the RE: $(0+1)01$.



EQUIVALENCE OF DFA AND NFA

Every language that can be described by some NFA can also be described by some DFA. It involves subset construction process for conversion.

Let $N = (Q_N, \Sigma, \delta_N, \{q_0\}, F_N)$ be the given NFA.

We can construct DFA,

$$D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$$

where,

$Q_D \rightarrow$ Set of all subsets of Q_N

$F_D \rightarrow$ Set of subsets, S of Q_N such that $S \cap F_N \neq \emptyset$

$\delta_D \rightarrow$ For each set, S contained in Q_N , i.e., $S \subseteq Q_N$ and each input a in Σ be defined as $\delta_D(S, a) = \delta_N(p, a); p \in S$

Procedure

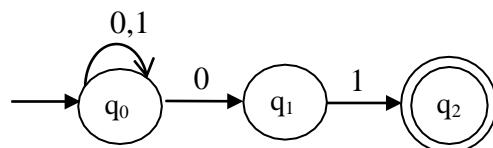
1. If $Q = \{q_0, q_1, q_2\}$, then create subsets, $\{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}$ and \emptyset . Thus if no of states = n ; the no of subsets = 2^n .
2. Construct the transition table for the subsets over the given language, Σ .
3. Minimise the table by considering only the reachable states from the start state.
4. Draw the DFA from the minimised transition table.

PROBLEMS

- 1) Construct a DFA from the following NFA that accepts all strings of 0's and 1's that end with „01”. AU - NOV/DEC 2013, NOV/DEC 2003, MAY/JUNE 2006

Solution:

NFA



Here ,

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

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

$$q_0 = \{q_0\}$$

$$F = \{q_2\}$$

δ

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

Subsets

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

Transition table: δ_D

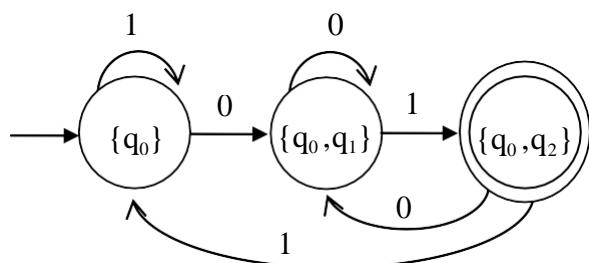
$Q \backslash \Sigma$	0	1
0	\emptyset	\emptyset
1	\emptyset	\emptyset
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	\emptyset	$\{q_2\}$
$\{q_2\}$	\emptyset	\emptyset
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1, q_2\}$	\emptyset	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Minimised transition table

$Q \backslash \Sigma$	0	1
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

$\leftarrow \{q_0\}$ Processed
 $\leftarrow \{q_0, q_1\}$ Processed
 $\leftarrow \{q_0\} \{q_0, q_1\}$ both are processed

Minimised DFA



2) Convert the given NFA to DFA.

$$Q_N = \{p, q, r, s\}$$

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

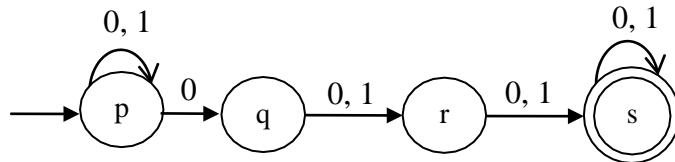
$$q_0 = \{p\}$$

$$F_N = \{s\}$$

NFA for the above specification

$$\delta_N:$$

$\Sigma \backslash Q$	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r\}$	$\{r\}$
r	$\{s\}$	\emptyset
$*s$	$\{s\}$	$\{s\}$



Subsets

$$Q_D = \{\emptyset, \{p\}, \{q\}, \{r\}, \{s\}, \{p, q\}, \{p, r\}, \{p, s\}, \{q, r\}, \{q, s\}, \{r, s\}, \{p, q, r\}, \{p, q, s\}, \{q, r, s\}, \{p, r, s\}, \{p, q, r, s\}\}$$

$$|Q_D| = 2^4 = 16$$

Transition table

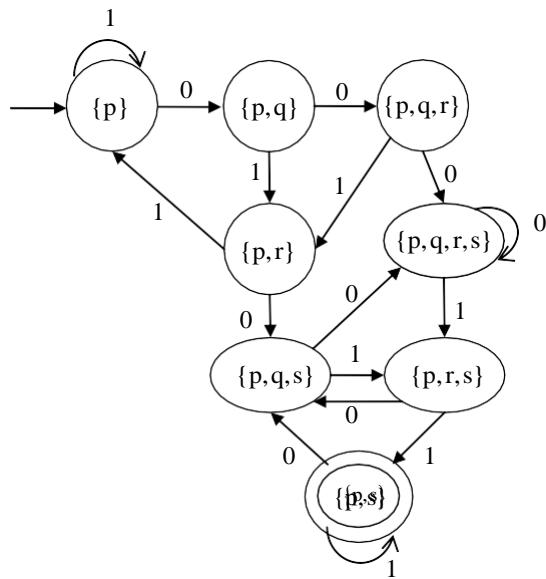
$\Sigma \backslash Q$	0	1
\emptyset	\emptyset	\emptyset
$\rightarrow \{p\}$	$\{p, q\}$	$\{p\}$
$\{q\}$	$\{r\}$	$\{r\}$
$\{r\}$	$\{s\}$	\emptyset
$*\{s\}$	$\{s\}$	$\{s\}$
$\{p, q\}$	$\{p, q, r\}$	$\{p, r\}$
$\{p, r\}$	$\{p, q, s\}$	$\{p\}$
$*\{p, s\}$	$\{p, q, s\}$	$\{p, s\}$
$\{q, r\}$	$\{r, s\}$	$\{r\}$
$*\{q, s\}$	$\{r, s\}$	$\{r, s\}$
$*\{r, s\}$	$\{s\}$	$\{s\}$

$\{p, q, r\}$	$\{p, q, r, s\}$	$\{p, r\}$
$*\{p, q, s\}$	$\{p, q, r, s\}$	$\{p, r, s\}$
$*\{p, r, s\}$	$\{p, q, s\}$	$\{p, s\}$
$*\{q, r, s\}$	$\{r, s\}$	$\{r, s\}$
$*\{p, q, r, s\}$	$\{p, q, r, s\}$	$\{p, r, s\}$

Minimised transition table

$Q \setminus \Sigma$	0	1
$\rightarrow\{p\}$	$\{p, q\}$	$\{p\}$
$\{p, q\}$	$\{p, q, r\}$	$\{p, r\}$
$\{p, q, r\}$	$\{p, q, r, s\}$	$\{p, r\}$
$\{p, r\}$	$\{p, q, s\}$	$\{p\}$
$\{p, q, r, s\}$	$\{p, q, r, s\}$	$\{p, r, s\}$
$\{p, q, s\}$	$\{p, q, r, s\}$	$\{p, r, s\}$
$\{p, r, s\}$	$\{p, q, s\}$	$\{p, s\}$
$*\{p, s\}$	$\{p, q, s\}$	$\{p, s\}$

DFA



↔ 3) Convert the following NFA to DFA. AU NOV/DEC 2004

$\Sigma \backslash Q$	0	1
$\rightarrow \{p\}$	$\{q, s\}$	$\{q\}$
q^*	$\{r\}$	$\{q, r\}$
r	$\{s\}$	$\{p\}$
$* s$	\emptyset	$\{p\}$

Solution:-

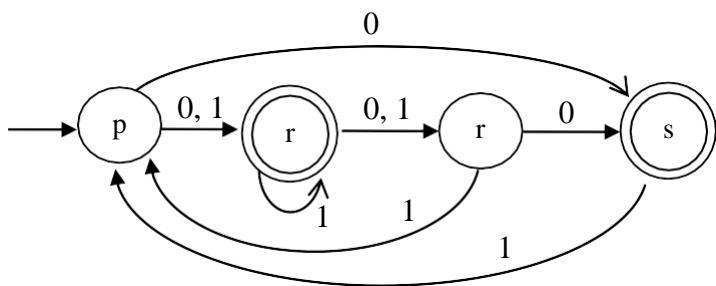
Here,

$$Q_N = \{p, q, r, s\}$$

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

$$q_0 = \{p\}$$

$$F_N = \{q, s\}$$



Subsets:

$$Q_D = \{\emptyset, \{p\}, \{q\}, \{r\}, \{s\}, \{p, q\}, \{p, r\}, \{p, s\}, \{q, r\}, \{q, s\}, \{r, s\}, \{p, q, r\}, \{p, q, s\}, \{p, r, s\}, \{q, r, s\}, \{p, q, r, s\}\}$$

$$F_D = \{\{q\}, \{s\}, \{p, s\}, \{q, r\}, \{q, s\}, \{r, s\}, \{p, q, r\}, \{p, q, s\}, \{p, r, s\}, \{q, r, s\}, \{p, q, r, s\}\}$$

Transition table:-

$\Sigma \backslash Q$	0	1
\emptyset	\emptyset	\emptyset
$\rightarrow \{p\}$	$\{q, s\}$	$\{q\}$
$* \{q\}$	$\{r\}$	$\{q, r\}$

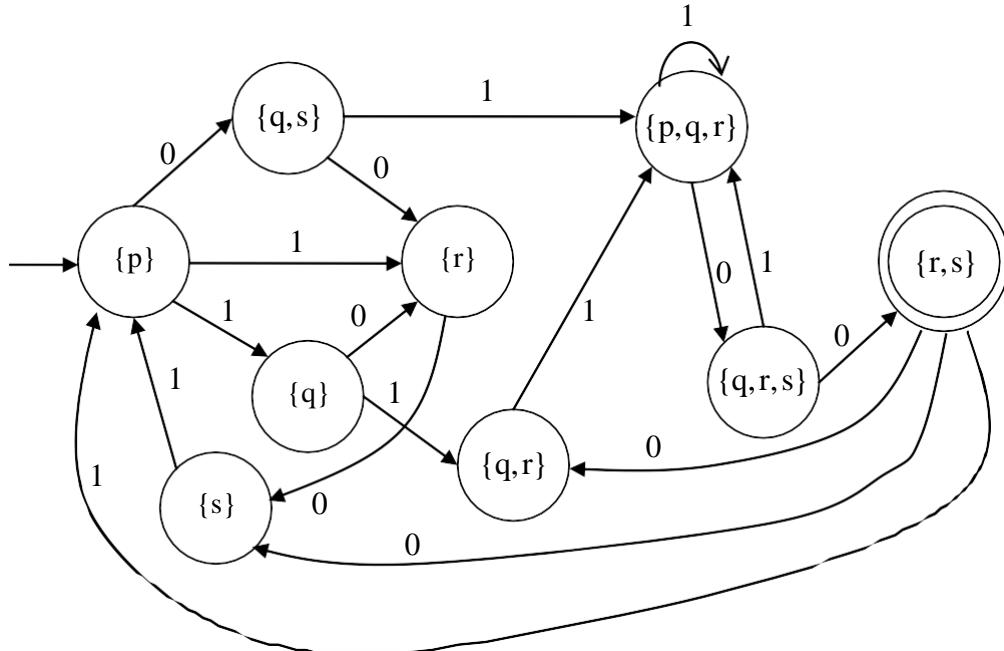
$\cdot\{r\}$	$\{s\}$	$\{p\}$
$*\{s\}$	ϕ	$\{p\}$
$\{p, q\}$	$\{q, r, s\}$	$\{p, q, r\}$
$\{p, r\}$	$\{q, s\}$	$\{p, q\}$
$*\{p, s\}$	$\{q, s\}$	$\{p, q\}$
$\{q, r\}$	$\{r, s\}$	$\{p, q, r\}$
$*\{q, s\}$	$\{r\}$	$\{p, q, r\}$
$*\{r, s\}$	$\{s\}$	$\{p\}$
$\{p, q, r\}$	$\{q, r, s\}$	$\{p, q, r\}$
$*\{p, q, s\}$	$\{q, r, s\}$	$\{p, q, r\}$
$*\{p, r, s\}$	$\{q, s\}$	$\{p, q\}$
$*\{q, r, s\}$	$\{r, s\}$	$\{p, q, r\}$
$*\{p, q, r, s\}$	$\{q, r, s\}$	$\{p, q, r\}$

Minimised transition table:-

$\Sigma \backslash Q$	0	1
$\rightarrow\{p\}$	$\{q, s\}$	$\{q\}$
$\{q, s\}$	$\{r\}$	$\{p, q, r\}$
$\{q\}$	$\{r\}$	$\{q, r\}$
$\{r\}$	$\{s\}$	$\{p\}$
$\{p, q, r\}$	$\{q, r, s\}$	$\{p, q, r\}$
$\{q, r\}$	$\{r, s\}$	$\{p, q, r\}$
$\{s\}$	ϕ	$\{p\}$
$\{q, r, s\}$	$\{r, s\}$	$\{p, q, r\}$
$*\{r, s\}$	$\{s\}$	$\{p\}$

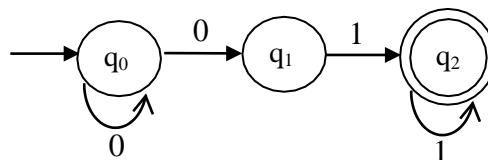


DFA diagram:-



4) Construct a DFA for the following NFA.

AU MAY/JUN 2009



Solution:-

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

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

$$q_0 = \{q_0\}$$

$$F_N = \{q_2\}$$

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

Construction of DFA:-

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

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

$$q_0 = \{q_0\}$$

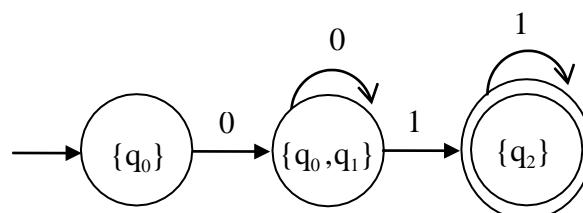
Transition table:-

Σ Q	0	1
ϕ	ϕ	ϕ
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	ϕ
$\{q_1\}$	ϕ	$\{q_2\}$
$*\{q_2\}$	ϕ	$\{q_2\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_2\}$
$*\{q_1, q_2\}$	ϕ	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_2\}$

Minimized transition table:-

Σ Q	0	1
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	ϕ
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_2\}$
$*\{q_2\}$	ϕ	$\{q_2\}$

Minimized DFA:-



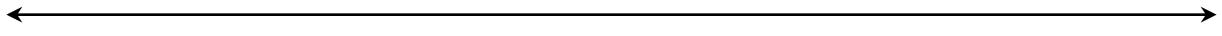
5) Convert $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \delta(q_1, \{q_3\}), \text{ where } \delta \text{ is given by,}$

$$\delta(q_1, 0) = \{q_2, q_3\} \quad \delta(q_1, 1) = \{q_1\}$$

$$\delta(q_2, 0) = \{q_1, q_2\} \quad \delta(q_2, 1) = \phi$$

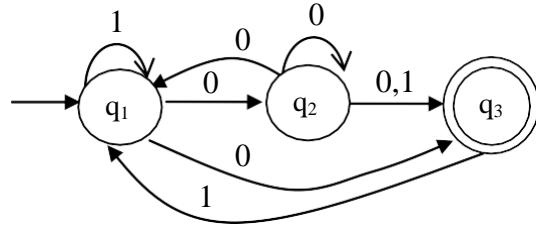
$$\delta(q_3, 0) = \{q_2\} \quad \delta(q_3, 1) = \{q_1, q_2\}$$

Construct an equivalent DFA



Solution:

Given NFA,



$\Sigma \backslash Q$	0	1
$\rightarrow q_1$	$\{q_2, q_3\}$	$\{q_1\}$
q_2	$\{q_1, q_2\}$	\emptyset
$* q_3$	$\{q_2\}$	$\{q_1, q_2\}$

DFA Construction:-

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

$$Q_D = \{\emptyset, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \{q_1, q_3\}, \{q_2, q_3\}, \{q_1, q_2, q_3\}\}$$

$$q_0 = \{q_1\}$$

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

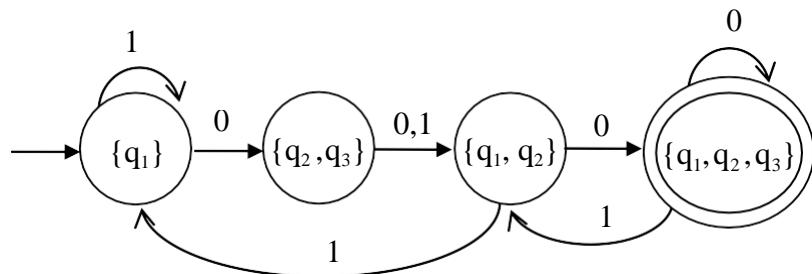
Transition Table:-

$\Sigma \backslash Q$	0	1
\rightarrow	\emptyset	\emptyset
$\{q_1\}$	$\{q_2, q_3\}$	$\{q_1\}$
$\{q_2\}$	$\{q_1, q_2\}$	\emptyset
$\{q_3\}$	$\{q_2\}$	$\{q_1, q_2\}$
$\{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1\}$
$\{q_1, q_3\}$	$\{q_2, q_3\}$	$\{q_1, q_2\}$
$\{q_2, q_3\}$	$\{q_1, q_2\}$	$\{q_1, q_2\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$

Minimized DFA table:-

Q	Σ	
	0	1
{q ₁ }	{q ₂ , q ₃ }	{q ₁ }
{q ₂ , q ₃ }	{q ₁ , q ₂ }	{q ₁ , q ₂ }
{q ₁ , q ₂ }	{q ₁ , q ₂ , q ₃ }	{q ₁ }
{q ₁ , q ₂ , q ₃ }	{q ₁ , q ₂ , q ₃ }	{q ₁ , q ₂ }

Minimized DFA:-



THEOREM

AU – MAY 2005, DEC 2006, DEC 2013

A Language, L is accepted by some NFA if and only if it is accepted by some DFA

(or)

For every NFA, there exists an equivalent DFA

Proof:-

This theorem has two parts:-

- 1) If L is accepted by a DFA (M2), then L is accepted by some NFA (M1)
- 2) If L is accepted by a NFA (M1), then L is accepted by some DFA (M2)

To prove the theorem, it is enough to prove that

$$\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(\{q_0\}, w)$$

For an arbitrary string w, we prove this statement, we use induction principle on |w|.

Basis of Induction:-

Let w ≠ 0

Then w = ε

By the definition of extended transition function,

$$\hat{\delta}_D(\{q_0\}, \epsilon) = \{q_0\} \quad \text{----- (1)}$$

$$\hat{\delta}_N(\{q_0\}, \epsilon) = \{q_0\} \quad \text{----- (2)}$$

$\therefore \hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(\{q_0\}, w)$ is true for $w = \epsilon$ Hence proved that $L[D] = L[N]$.

Inductive Step:-

Let us assume that the theorem is true for all the strings, w of length „n“. (i.e.) $|w| = n$.

Now let us prove that the same is true for a string of length, $n+1$

Let $w = xa$, where

$a \rightarrow$ last symbol of w .

By assumption, we have,

$$\begin{aligned}\hat{\delta}_D(\{q_0\}, x) &= \hat{\delta}_N(\{q_0\}, x) \\ &= \{P_1, P_2, \dots, P_k\} \rightarrow \text{subsets} \left[\bigcup_{i=1}^k P_i \right]\end{aligned}$$

Now, by the definition of extended transition function of DFA,

$$\begin{aligned}\hat{\delta}_D(\{q_0\}, x) &= \hat{\delta}_D(\{q_0\}, x_a) \\ &= \delta_D(\hat{\delta}(\{q_0\}, x), a) \\ &= \delta_D(\{P_1, P_2, \dots, P_k\}) \\ &= U_{i=1}^k \delta_N(P_i, a) \rightarrow \text{By subset construction} \\ &= \hat{\delta}_N(q_0, w) \rightarrow \text{By definition of } \hat{\delta} \text{ of NFA}\end{aligned}$$

\therefore By induction principle, the theorem is true for all the strings, w of the given language.

Thus, $L[D] = L[N]$

NFA WITH ϵ TRANSITIONS

The ϵ [Epsilon] used to indicate null string. The string is used for transition from one state to other without any input. **AU DEC 2007**

Formal Notation for ϵ – NFA:

A non deterministic FA, M with ϵ –transition is given by $E = (Q, \Sigma, \delta, q_0, F)$, where

$Q \rightarrow$ finite set of states

$\Sigma \rightarrow$ finite set of input alphabets

$q_0 \rightarrow$ Initial/ start state

$F \rightarrow F \subseteq Q \rightarrow$ set of final/accepting states

$\delta \rightarrow$ Transition/ Mapping function from

$$Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$$

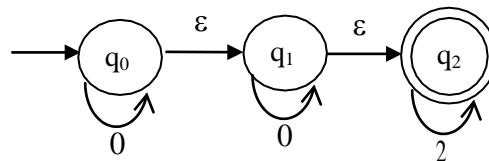
EPSILON CLOSURE

AU MAY 2012, DEC 2010, MAY 2013

Epsilon closure of a state, q is defined as the set of states reachable from q on ϵ moves. ϵ - closure of q_0 is denoted as $\epsilon\text{CLOSE}(q_0)$.

Example: Obtain ϵ closure of each state in the following ϵ – NFA with ϵ – move.

AU DEC 2005



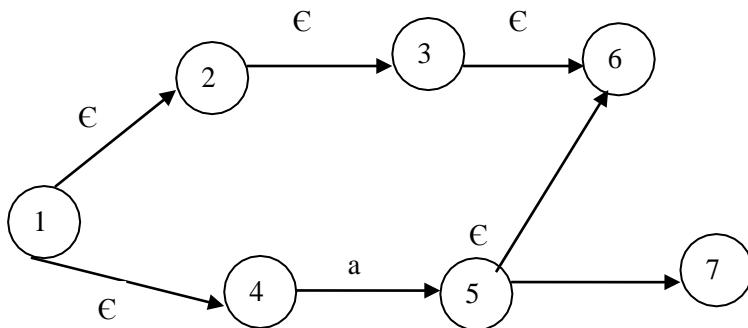
ϵ CLOSURE $\{q_0\} = \{q_0, q_1, q_2\}$

ϵ CLOSURE $\{q_1\} = \{q_1, q_2\}$

ϵ CLOSURE $\{q_2\} = \{q_2\}$

Find ϵ - CLOSURE of States 1,2 and 4 from the following transition diagram.

AU MAY 2008



ϵ –Closure $\{1\} = \{1,2,4,3,6\}$

ϵ –Closure $\{2\} = \{2,3,6\}$

ϵ –Closure $\{4\} = \{4\}$

Eliminating Epsilon transitions

Given any epsilon NFA with ϵ moves, we can find a DFA that accept the same language as E.



Let $E = (Q_E, \Sigma, \delta_E, q_E, F_E)$



Then the equivalent DFA,

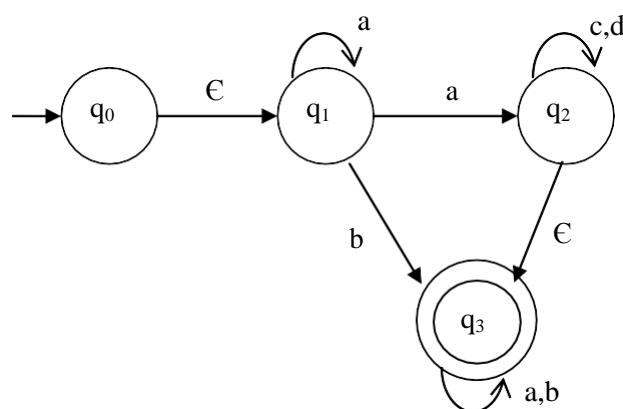
$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ is defined as follows.

- (i) Q_D is the set of subsets, s of Q_E . $S \subseteq Q_E$ such that $s = ECLOSE(S)$
- (ii) $q_D = ECLOSE(q_E)$
- (iii) $F_D = \{s / s \text{ is in } Q_D \text{ and } S \cap F_E \neq \emptyset\}$
- (iv) $\delta_D(s, a)$ is computed for all a in Σ and sets, S in Q_D by
 - a) Let $S = P_1, P_2, \dots, P_k$
 - b) $\bigcup_{i=1}^k \delta(P_i, a) = \{r_1, r_2, \dots, r_m\}$
 - c) $\delta_D(s, a) = \bigcup_{j=1}^m ECLOSE(r_j)$

PROBLEMS

- 1) Design a DFA that eliminates ϵ transition from the following ϵ -NFA.

Σ Q_E	ϵ	a	b	c	d
$\rightarrow q_0$	q_0, q_1	ϕ	ϕ	ϕ	ϕ
q_1	q_1	q_1, q_2	q_3	ϕ	ϕ
q_2	q_2, q_3	ϕ	ϕ	q_2	q_2
$*q_3$	q_3	q_3	q_3	ϕ	ϕ



Solution :-

Start state of DFA:-

$$ECLOSE(q_0) = \{q_0, q_1\} \rightarrow \text{Start State of DFA}$$

Successors of $\{q_0, q_1\}$ for the input symbols, {a,b,c,d}:-

$$\delta_D(\{q_0, q_1\}, a) : -$$

$$\delta_E(q_0, a) = \phi$$

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

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

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

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

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

$$\delta_D(\{q_0, q_1\}, b) : -$$

$$\delta_E(q_0, b) = \phi$$

$$\delta_E(q_1, b) = \{q_3\}$$

$$\delta_E(\{q_0, q_1\}, b) = \{q_3\}$$

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

$$\delta_D(\{q_0, q_1\}, b) = \{q_3\}$$

$$\delta_D(\{q_0, q_1\}, c) : -$$

$$\delta_E(q_0, c) = \phi$$

$$\delta_E(q_1, c) = \phi$$

$$\delta_E(\{q_0, q_1\}, c) = \phi$$

$$\delta_D(\{q_0, q_1\}, c) = ECLOSE(\phi)$$

$$\therefore \delta_D(\{q_0, q_1\}, c) = \phi$$

$$\delta_D(\{q_0, q_1\}, d) : -$$

$$\delta_E(q_0, d) = \phi$$

$$\delta_E(q_1, d) = \phi$$

$$\delta_E(\{q_0, q_1\}, d) = \phi$$

$$\delta_D(\{q_0, q_1\}, d) = ECLOSE(\phi)$$

$$\therefore \delta_D(\{q_0, q_1\}, d) = \phi$$

Successors of $\{q_1, q_2, q_3\}$ for the input symbols {a,b,c,d}:-

$$\delta_D(\{q_1, q_2, q_3\}, a) : -$$

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

$$\delta_E(q_2, a) = \emptyset$$

$$\delta_E(\{q_1, q_2, q_3\}, a) = \{q_1, q_2, q_3\}$$

$$\delta_D(\{q_1, q_2, q_3\}, a) = \text{CLOSE}\{q_1\} \cup \text{CLOSE}\{q_3\}$$

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

$$\delta_D(\{q_1, q_2, q_3\}, a) = \{q_1, q_2, q_3\}$$

$$\delta_D(\{q_1, q_2, q_3\}, b) : -$$

$$\delta_E(q_1, b) = \{q_3\}$$

$$\delta_E(q_2, b) = \emptyset$$

$$\delta_E(q_3, b) = \{q_3\}$$

$$\delta_E(\{q_1, q_2, q_3\}, b) = \{q_3\}$$

$$\delta_D(\{q_1, q_2, q_3\}, b) = \text{CLOSE}\{q_3\}$$

$$\delta_D(\{q_1, q_2, q_3\}, b) = \{q_3\}$$

$$\delta_D(\{q_1, q_2, q_3\}, c) : -$$

$$\delta_E(q_1, c) = \emptyset$$

$$\delta_E(q_2, c) = \{q_2\}$$

$$\delta_E(q_3, c) = \emptyset$$

$$\delta_E(\{q_1, q_2, q_3\}, c) = \{q_2\}$$

$$\delta_D(\{q_1, q_2, q_3\}, c) = \text{CLOSE}\{q_2\}$$

$$\delta_D(\{q_1, q_2, q_3\}, d) : -$$

$$\delta_E(q_2, d) = \emptyset$$

$$\delta_E(q_3, d) = \{q_2\}$$

$$\delta_E(q_3, d) = \emptyset$$

$$\delta_E(\{q_1, q_2, q_3\}, d) = \{q_2\}$$

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

$$\delta_D(\{q_1, q_2, q_3\}, d) = \{q_2, q_3\}$$

Successors of $\{q_3\}$ for the input symbols, {a,b,c,d}:-

$$\delta_D(\{q_3\}, a) : -$$

$$\delta_E(\{q_3\}, a) = \{q_3\}$$

$$\delta_D(q_3, a) = ECLOSE\{q_3\}$$

$$\delta_D(q_3, a) = \{q_3\}$$

$$\delta_D(\{q_3\}, b) : -$$

$$\delta_E(\{q_3\}, b) = \{q_3\}$$

$$\delta_D(q_3, b) = ECLOSE\{q_3\}$$

$$\delta_D(\{q_3\}, b) = \{q_3\}$$

$$\delta_D(\{q_3\}, c) : -$$

$$\delta_E(\{q_3\}, c) = \phi$$

$$\delta_D(\{q_3\}, c) = \phi$$

$$\delta_D(\{q_3\}, c) = \phi$$

$$\delta_D(\{q_3\}, d) : -$$

$$\delta_E(\{q_3\}, d) = \phi$$

$$\delta_D(\{q_3\}, d) = \phi$$

$$\delta_D(\{q_3\}, d) = \phi$$

Successors of $\{q_2, q_3\}$ over the input symbols {a,b,c,d}:-

$$\delta_D(\{q_2, q_3\}, a) : -$$

$$\delta_E(\{q_2\}, a) = \phi$$

$$\delta_E(\{q_3\}, a) = q_3$$

$$\delta_E(\{q_2, q_3\}, q) = \{q_3\}$$

↔

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

$$\delta_D(\{q_2, q_3\}, a) = \{q_3\}$$

$\delta_D(\{q_2, q_3\}, b) :-$

$$\delta_D(\{q_3\}, b) = \emptyset$$

$$\delta_E(\{q_3\}, b) = \{q_3\}$$

$$\delta_E(\{q_2, q_3\}, b) = \{q_3\}$$

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

$$\delta_D(\{q_2, q_3\}, b) = \{q_3\}$$

$\delta_D(\{q_2, q_3\}, c) :-$

$$\delta_E(\{q_2\}, c) = \{q_2\}$$

$$\delta_D(\{q_3\}, c) = \emptyset$$

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

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

$$\delta_D(\{q_2, q_3\}, c) = \{q_2, q_3\}$$

$\delta_D(\{q_2, q_3\}, d) :-$

$$\delta_E(\{q_2\}, d) = \{q_2\}$$

$$\delta_D(\{q_3\}, d) = \emptyset$$

$$\delta(\{q_2, q_3\}, d) = \{q_2\}$$

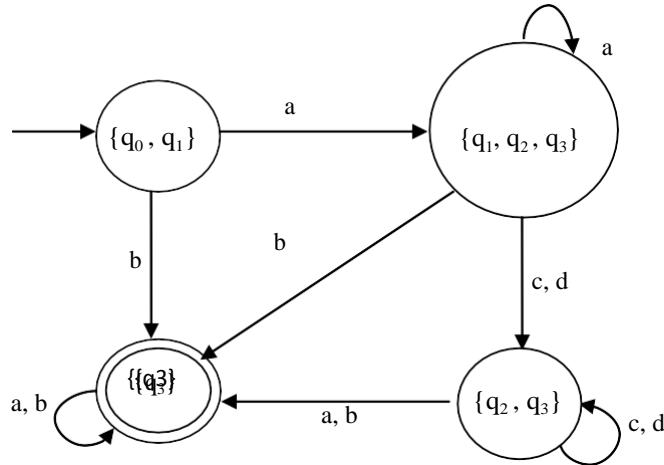
$$\delta_D(\{q_2, q_3\}, d) = ECLOSE\{q_2\}$$

$$\delta_D(\{q_2, q_3\}, d) = \{q_2, q_3\}$$

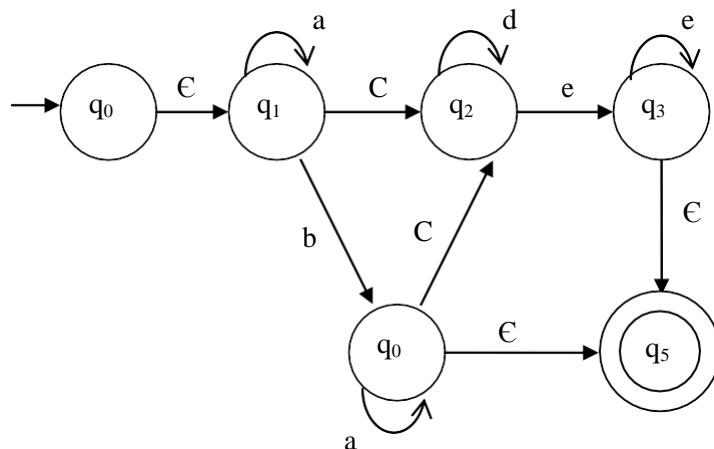
Transition Table:-

Σ Q_D	a	b	c	d
$\rightarrow \{q_0, q_1\}$	$\{q_1, q_2, q_3\}$	$\{q_3\}$	\emptyset	\emptyset
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_3\}$	$\{q_2, q_3\}$	$\{q_2, q_3\}$
$* \{q_3\}$	$\{q_3\}$	$\{q_3\}$	\emptyset	\emptyset
$\{q_2, q_3\}$	$\{q_3\}$	$\{q_3\}$	$\{q_2, q_3\}$	$\{q_2, q_3\}$

Equivalent DFA:-



2) Design a DFA that eliminates ϵ transitions from the following ϵ -NFA.



Solution:-

Start of DFA:-

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

↓

Start State of DFA

Successors of $\{q_0, q_1\}$ over $\Sigma = \{a, b, c, d, e\}$:

$$\delta_D(\{q_0, q_1\}, a) :=$$

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

$$= \emptyset \cup \{q_4\} = \{q_4\}$$

Σ Q_D	ϵ	a	b	c	d	e
$\rightarrow q_0$	q_0, q_1	ϕ	ϕ	ϕ	ϕ	ϕ
q_1	q_1	q_1	q_4	q_2	ϕ	ϕ
q_2	q_2	ϕ	ϕ	ϕ	q_2	q_3
q_3	q_3, q_5	ϕ	ϕ	ϕ	ϕ	q_3
q_4	q_4, q_5	q_4	ϕ	q_2	ϕ	ϕ
$*q_5$	q_5	ϕ	ϕ	ϕ	ϕ	ϕ

$$\delta_D(\{q_0, q_1\}, a) = \text{ECLOSE}(q_1)$$

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

$$\delta_D(\{q_0, q_2\}, b) :=$$

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

$$= \phi \cup \{q_4\}$$

$$= \{q_4\}$$

$$\delta_D(\{q_0, q_1\}, b) = \text{ECLOSE}(q_4)$$

$$\delta_D(\{q_0, q_1\}, b) = \{q_4, q_5\}$$

$$\delta_D(\{q_0, q_2\}, c) :=$$

$$\delta_E(\{q_0, q_1\}, c) = \delta_E(\{q_0\}, c) \cup \delta_E(\{q_1\}, c)$$

$$= \phi \cup \{q_2\}$$

$$= \{q_2\}$$

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

$$\delta_D(\{q_0, q_1\}, c) = \{q_2\}$$

$$\delta_D(\{q_0, q_2\}, d) :=$$

$$\delta_E(\{q_0, q_1\}, d) = \underbrace{\delta_E(\{q_0\}, d)}_{\phi} \cup \underbrace{\delta_E(\{q_1\}, d)}_{\phi}$$

$$\phi \quad \phi$$

$$= \phi$$

$$\delta_D(\{q_0, q_1\}, d) = \phi$$

Successors of $\{q_1\}$ over $\Sigma = \{a, b, c, d, e\}$:

$$\delta_D(\{q_1\}, a) : -$$

$$\delta_E(\{q_1\}, a) = \{q_1\}$$

$$\delta_D(\{q_1\}, a) = ECLOSE(q_1)$$

$$\delta_D(\{q_1\}, a) = \{q_1\}$$

$$\delta_D(\{q_1\}, b) : -$$

$$\delta_E(\{q_1\}, b) = \{q_4\}$$

$$\delta_D(\{q_1\}, b) = ECLOSE(q_4)$$

$$\delta_D(\{q_1\}, b) = \{q_4, q_5\}$$

$$\delta_D(\{q_1\}, c) : -$$

$$\delta_E(\{q_1\}, c) = \{q_2\}$$

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

$$\delta_D(\{q_1\}, c) = \{q_2\}$$

$$\delta_D(\{q_1\}, d) : -$$

$$\delta_E(\{q_1\}, d) = \emptyset$$

$$\delta_D(\{q_1\}, d) = \emptyset \quad \leftarrow ECLOSE(\emptyset) = \emptyset$$

$$\delta_D(\{q_1\}, e) : -$$

$$\delta_E(\{q_1\}, e) = \emptyset$$

$$\delta_D(\{q_1\}, e) = \emptyset \quad \leftarrow ECLOSE(\emptyset) = \emptyset$$

Successors of $\{q_4, q_5\}$ over $\Sigma = \{a, b, c, d, e\}$:

$$\delta_D(\{q_4, q_5\}, a) : -$$

$$\delta_E(\{q_4, q_5\}, a) = \delta_E(\{q_4\}, a) \cup \delta_E(\{q_5\}, a)$$

$$\{q_1\} \cup \emptyset$$

$$= \{q_4\}$$

$$\delta_D(\{q_4, q_5\}, a) = \{q_4, q_5\}$$

← →

$\delta_D(\{q_4, q_5\}, b) :-$

$$\delta_E(\{q_4, q_5\}, b) = \delta_E(\{q_4\}, b) \cup \delta_E(\{q_5\}, b)$$

$$= \emptyset$$

$$\delta_D(\{q_4, q_5\}, b) = \emptyset \quad \leftarrow \text{ECLOSE}(\emptyset) = \emptyset$$

$\delta_D(\{q_4, q_5\}, c) :-$

$$\delta_E(\{q_4, q_5\}, c) = \delta_E(\{q_4\}, c) \cup \delta_E(\{q_5\}, c) \cup \{q_2\} \cup \emptyset$$

$$= \{q_2\}$$

$$\delta_D(\{q_4, q_5\}, c) = \text{ECLOSE}(q_2)$$

$$\delta_D(\{q_4, q_5\}, c) = \{q_2\}$$

$\delta_D(\{q_4, q_5\}, d) :-$

$$\delta_E(\{q_4, q_5\}, d) = \delta_E(\{q_4\}, d) \cup \delta_E(\{q_5\}, d)$$

$$= \emptyset$$

$$\delta_D(\{q_4, q_5\}, d) = \emptyset \quad \leftarrow \text{ECLOSE}(\emptyset) = \emptyset$$

$\delta_D(\{q_4, q_5\}, e) :-$

$$\delta_E(\{q_4, q_5\}, e) = \delta_E(\{q_4\}, e) \cup \delta_E(\{q_5\}, e)$$

$$= \emptyset$$

$$\delta_D(\{q_4, q_5\}, e) = \emptyset \quad \leftarrow \text{ECLOSE}(\emptyset) = \emptyset$$

Successors of $\{q_2\}$ over $\Sigma = \{a, b, c, d, e\}$:

$\delta_D(\{q_2\}, a) :-$

$$\delta_E(\{q_2\}, a) = \delta \emptyset$$

$$\delta_D(\{q_2\}, a) = \emptyset$$

$$\leftarrow \text{ECLOSE}(\emptyset) = \emptyset$$

$\delta_D(\{q_2\}, b) :-$

$$\delta_D(\{q_2\}, b) = \emptyset$$

$$\delta_D(\{q_2\}, b) = \emptyset$$

$$\leftarrow \text{ECLOSE}(\emptyset) = \emptyset$$

$\delta_D(\{q_2\}, c) :-$

$$\delta_D(\{q_2\}, c) = \phi$$

$$\delta_D(\{q_2\}, c) = \phi \quad \leftarrow \text{ECLOSE}(\phi) = \phi$$

$\delta_D(\{q_2\}, d) :-$

$$\delta_E(\{q_2\}, d) = \{q_2\}$$

$$\delta_D(\{q_2\}, d) = \text{ECLOSE}(q_2)$$

$$\delta_D(\{q_2\}, d) = \{q_2\}$$

$\delta_D(\{q_2\}, e) :-$

$$\delta_E(\{q_2\}, e) = \{q_3\}$$

$$\delta_D(\{q_2\}, e) = \text{ECLOSE}(q_3)$$

$$\delta_D(\{q_2\}, e) = \{q_3\}$$

Successors of $\{q_3, q_5\}$ over $\Sigma = \{a, b, c, d, e\}$:

$\delta_D(\{q_3, q_5\}, a) :-$

$$\delta_E(\{q_3, q_5\}, a) = \delta_E(\{q_3\}, a) \cup \delta_E(\{q_5\}, a)$$

$$= \phi \quad \leftarrow \phi \cup \phi = \phi$$

$$\delta_D(\{q_3, q_5\}, a) = \phi \quad \leftarrow \text{ECLOSE}(\phi) = \phi$$

$\delta_D(\{q_3, q_5\}, b) :-$

$$\delta_E(\{q_3, q_5\}, b) = \delta_E(\{q_3\}, b) \cup \delta_E(\{q_5\}, b)$$

$$= \phi \quad \leftarrow \phi \cup \phi = \phi$$

$$\delta_D(\{q_3, q_5\}, b) = \phi \quad \leftarrow \text{ECLOSE}(\phi) = \phi$$

$\delta_D(\{q_3, q_5\}, c) :-$

$$\delta_E(\{q_3, q_5\}, c) = \delta_E(\{q_3\}, c) \cup \delta_E(\{q_5\}, c)$$

$$= \phi \quad \leftarrow \phi \cup \phi = \phi$$

$$\delta_D(\{q_3, q_5\}, c) = \phi \quad \leftarrow \text{ECLOSE}(\phi) = \phi$$

$$\delta_D(\{q_3, q_5\}, d) : -V$$

$$\delta_E(\{q_3, q_5\}, d) = \delta_E(\{q_3\}, d) \cup \delta_E(\{q_5\}, d)$$

$$= \phi \quad \leftarrow \phi \cup \phi = \phi$$

$$\delta_D(\{q_3, q_5\}, d) = \phi \quad \leftarrow \text{ECLOSE}(\phi) = \phi$$

$$\delta_D(\{q_3, q_5\}, e) : -$$

$$\delta_E(\{q_3, q_5\}, e) = \delta_E(\{q_3\}, e) \cup \delta_E(\{q_5\}, e)$$

$$= \{q_3\} \cup \phi$$

$$= \{q_3\}$$

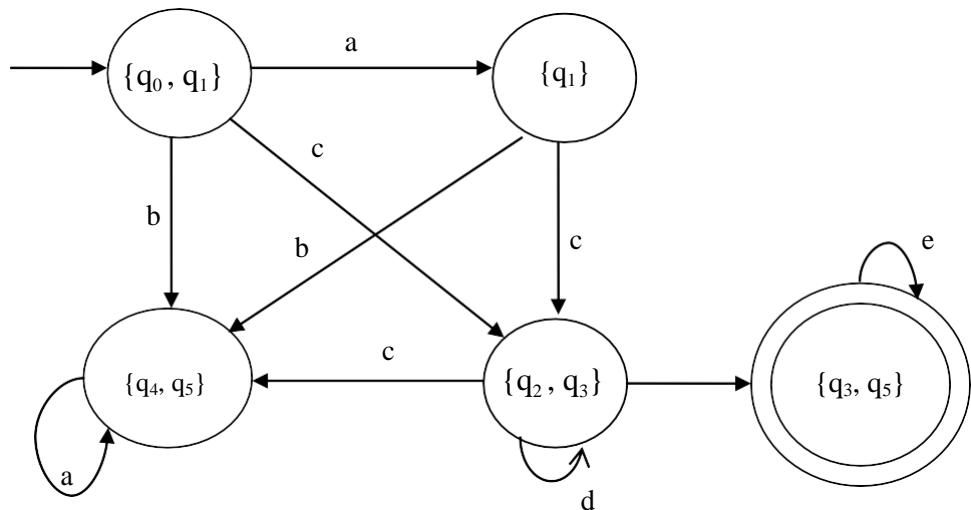
$$\delta_D(\{q_3, q_5\}, e) = \text{ECLOSE}(q_3)$$

$$\delta_D(\{q_3, q_5\}, e) = \{q_3, q_5\}$$

Transition Table:-

Σ Q_D	a	b	c	d	e
$\rightarrow \{q_0, q_1\}$	$\{q_1\}$	$\{q_4, q_5\}$	$\{q_2\}$	ϕ	ϕ
$\{q_1\}$	$\{q_1\}$	$\{q_4, q_5\}$	$\{q_2\}$	ϕ	ϕ
$\{q_4, q_5\}$	$\{q_4, q_5\}$	ϕ	$\{q_2\}$	ϕ	ϕ
$\{q_2\}$	ϕ	ϕ	ϕ	$\{q_2\}$	$\{q_3, q_5\}$
$*\{q_3, q_5\}$	ϕ	ϕ	ϕ	ϕ	$\{q_3, q_5\}$

DFA – Transition Diagram :-



3) Consider the following ϵ – NFA

\sum_{Q_E}	ϵ	a	b	c
$\rightarrow p$	{q,r}	ϕ	{q}	{r}
q	ϕ	{p}	{r}	{p,q}
$*_r$	ϕ	ϕ	ϕ	ϕ

- a) Complete the ϵ – closures of each state
- b) Convert the automation to DFA

Solution:-

- a) ϵ – closure of states:-

State	ϵ - Closure
p	{p,q,r}
q	{q}
r	{r}

- b) ϵ NFA \rightarrow DFA:-

Start state of DFA, q_D

$$\text{ECLOSE}(P) = \{p, q, r\}$$

Successors of {p, q, r} over $\sum = \{a, b, c\}$:-

$$\delta_D(\{p, q, r\}, a) : -$$

$$\begin{aligned} \delta_E(\{p, q, r\}, a) &= \delta_E(\{p\}, a) \cup \delta_E(\{q\}, a) \cup \delta_E(\{r\}, a) \\ &= \phi \cup \{p\} \cup \phi \\ &= \{p\} \end{aligned}$$

$$\delta_D(\{p, q, r\}, a) = \text{ECLOSE}(p)$$

$$\delta_D(\{p, q, r\}, a) = \{p, q, r\}$$

$$\delta_D(\{p, q, r\}, b) : -$$

$$\begin{aligned} \delta_E(\{p, q, r\}, b) &= \delta_E(\{p\}, b) \cup \delta_E(\{q\}, b) \cup \delta_E(\{r\}, b) \\ &= \{q\} \cup \{r\} \cup \phi \\ &= \{q, r\} \end{aligned}$$

← →

$$\delta_D(\{p, q, r\}, b) = ECLOSE(q) \cup ECLOSE(r)$$

$$\delta_D(\{p, q, r\}, b) = \{q, r\}$$

$\delta_D(\{p, q, r\}, c) :=$

$$\delta_E(\{p, q, r\}, c) = \delta_E(\{p\}, c) \cup \delta_E(\{q\}, c) \cup \delta_E(\{r\}, c)$$

$$= \{r\} \cup \{p, q\} \cup \emptyset$$

$$= \{p, q, r\}$$

$$\delta_D(\{p, q, r\}, c) = ECLOSE(p) \cup ECLOSE(q) \cup ECLOSE(r)$$

$$= \{p, q, r\} \cup \{q\} \cup \{r\}$$

$$\delta_D(\{p, q, r\}, c) = \{p, q, r\}$$

Successors of {q, r} over $\Sigma = \{a, b, c\}$:

$\delta_D(\{q, r\}, a) :=$

$$\delta_E(\{q, r\}, a) = \delta_E(\{q\}, a) \cup \delta_E(\{r\}, a)$$

$$= \{p\} \cup \emptyset$$

$$= \{p\}$$

$$\delta_D(\{q, r\}, a) = ECLOSE(p)$$

$$\delta_D(\{q, r\}, c) = \{p, q, r\}$$

$\delta_D(\{q, r\}, b) :=$

$$\delta_E(\{q, r\}, b) = \delta_E(\{q\}, b) \cup \delta_E(\{r\}, b)$$

$$= \{r\} \cup \emptyset$$

$$= \{r\}$$

$$\delta_D(\{q, r\}, b) = ECLOSE(r)$$

$$\delta_D(\{q, r\}, b) = \{r\}$$

$\delta_D(\{q, r\}, c) :=$

$$\delta_E(\{q, r\}, c) = \delta_E(\{q\}, c) \cup \delta_E(\{r\}, c)$$

$$= \{p, q\} \cup \emptyset$$

$$= \{p, q\}$$

$$\delta_D(\{q, r\}, c) = ECLOSE(p) \cup ECLOSE(q)$$

$$\delta_D(\{q, r\}, c) = \{p, q, r\} \leftarrow \{p, q, r\} \cup \{q\} = \{p, q, r\}$$

Successors of {r} over $\Sigma = (a, b, c)$:

$$\delta_D(\{r\}, a) :=$$

$$\delta_E(\{r\}, a) = \phi$$

$$\delta_D(\{r\}, a) = \phi \quad \leftarrow ECLOSE(\phi) = \phi$$

$$\delta_D(\{r\}, b) :=$$

$$\delta_E(\{r\}, b) = \phi$$

$$\delta_D(\{r\}, b) = \phi \quad \leftarrow ECLOSE(\phi) = \phi$$

$$\delta_D(\{r\}, c) :=$$

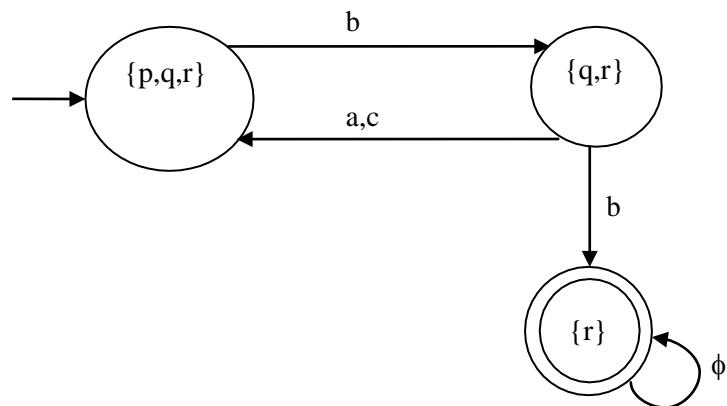
$$\delta_E(\{r\}, c) = \phi$$

$$\delta_D(\{r\}, c) = \phi \quad \leftarrow ECLOSE(\phi) = \phi$$

Transition Table:-

Σ Q_D	a	b	c
$\rightarrow \{p, q, r\}$	$\{p, q, r\}$	$\{q, r\}$	$\{p, q, r\}$
$\{q, r\}$	$\{p, q, r\}$	$\{r\}$	$\{p, q, r\}$
$* \{r\}$	ϕ	ϕ	ϕ

Transition diagram:-



UNIT -2

REGULAR EXPRESSIONS AND LANGUAGES

REGULAR LANGUAGES AND REGULAR EXPRESSIONS

The languages accepted by a finite automation can easily be described by a simple expression called Regular Expressions.

Regular Expressions serve as the input language for many systems that process strings. Regular expression uses a set of operators (like +, ., *,) to represent the language.

Search systems convert RE into DFA/NFA that stimulates the automation on the file being searched.

Example: Lexical Analyzer accepts tokens as inputs and produces a DFA that recognizes which token appears next on the input. The set of Strings accepted by finite automata is known as regular language.

Representation

DFA	Language	Regular Expression
	$\Rightarrow \{\epsilon\}$	$\Rightarrow R.E = \epsilon$
	$\Rightarrow \{a\}$	$\Rightarrow R.E = a$
	$\Rightarrow \{a, b\}$	$\Rightarrow R.E = a + b$
	$\Rightarrow \{ab\}$	$\Rightarrow R.E = a.b = ab$

Formal Definition of a RE

Let Σ be a given alphabet. Then,

- (i) ϕ , ϵ , and a belongs to Σ , are all regular expressions. [primitive Regular Expressions]
- (ii) If r_1 and r_2 are regular expressions, then r_1+r_2 , r_1r_2 , r_1^* , (r_1) are also regular expression if and only if it can be derived from the primitive RE by the finite number of applications of the rules in (ii).

Operators of Regular Expressions

The union of two languages, L and M is denoted by LUM which are a set of strings that are either in L or M or both.

Example:

$$L = \{001, 10, 111\}$$

$$M = \{\epsilon, 001\}$$

$$LUM = \{\epsilon, 00, 10, 111\}$$

The concatenation of languages L and M, denoted LM which are a set of strings that can be formed by taking any string in L and concatenating with any string in M.

Example:

$$L = \{001, 10, 111\}$$

$$M = \{\epsilon, 001\}$$

$$LM = \{001, 001001, 10, 1000, 111, 111001\}$$

The closure of a language, L is denoted as L^* represent the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions and concatenating all of them.

Example:

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

$$L^0 = \{\epsilon\}$$

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

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

PROBLEMS

1) What is $\{10, 11\}^*$? Write atleast first seven terms.

AU DEC 2009

$$(10, 11)^* = (10, 11)) U (10, 11)'' U (10, 11) 2 U (10, 11) 3 U.....$$

$$= \{\epsilon\} U \{10, 11\} U \{1011, 1110\} U \{101110, 101011, ...\}$$

$$= \{\epsilon, 10, 11, 1011, 1110, 101110, 101011, ...\}$$

2) Write a RE for the language accepting all combinations of a's over the set $\Sigma = \{a\}$

$$L = \{\text{all combination of } a^*\text{'s over } \{a\}\}$$

$$= \{C, a, aa, aaa, \dots\}$$

$$R = a^*$$

3) Design the RE for the language accepting all combinations of a's except the null string over

$$\{\epsilon\}$$

$$L = \{\text{all combination of } a \text{ over } \{a\} \text{ except } \epsilon\}$$

$$= \{a, aa, aaa, aaaa, \dots\}$$

$$R = a^+$$

1) Design a RE for the languages containing all thes strings containing any number of a's and b's .

$$L = \{\epsilon, a, b, aa, bb, ab, aab, \dots\}$$

$$R = (a+b)^*$$

2) Construct the RE for the language contai9ning all strings having any number of a's and b's except the null string.

$$L = \{a, ab, b, aa, abb, aabb, \dots\}$$

$$R = (a+b)^+$$

3) Construct a RE for the language accepting all strings which are ending with 00 over the set, $\Sigma = \{0,1\}$

$$L = \{00, 000, 100, 0100, 1000, 01000, 01000, 11100, \dots\}$$

$$R = (0+1)^* 00$$

4) Write a RE for the language accepting the strings which are starting with 1 and ending with 0, over the set $\Sigma = \{0,1\}$

$$L = \{10, 100, 110, 1000, 10101, 111000, \dots\}$$

$$R = 1(0+1)^* 0$$

5) Write a RE to denote a language, L over Σ^* where $\Sigma = \{a, b\}$ that the third character from right end of the string is always a.

$$L = aab, aba, aaa, abb, babb, aaaa, \dots$$

$$R = (a+b)^* a(a+b)(a+b)$$

6) Construct a RE for the language, L which accepts all the strings with at least 2 b's over the alphabets $\Sigma = \{a,b\}$

$$L = \{bb, abb, bba, bab, bbb, bbaa, ababa, \dots\}$$

$$R = (a+b)^*b(a+b)^*b(a+b)^*$$

7) Construct a RE for the language which consist of exactly 2 b's over the set, $\Sigma = \{a, b\}$

$$L = \{bb, abb, baba, bba, aaaabab, \dots\}$$

$$R = a^*ba^*ba^*$$

8) Construct a RE which denotes a language, L over the set, $\Sigma = \{0\}$, having even length of string.

$$L = \{00, \epsilon, 0000, 000000, \dots\}$$

$$R = (00)^*$$

9) Write a RE which denotes a language, L over the set, $\Sigma = \{1\}$, having odd length of strings

$$L = \{1, 111, 11111, 1111111, \dots\}$$

$$R = (11)^*1$$

10) Write the RE to denote the language, L over $\Sigma = \{a, b\}$, such that all the strings do not contain the substring ab.

$$L = \{bbaa, ba, bbbaaa, bb, aa, \dots\}$$

$$R = b^* a^*$$

11) Write a RE which contains L having the strings which have at least one 0 and 1.

$$L = \{01, 001, 101, 010, 011, 10101, 10, \dots\}$$

$$\rightarrow R = (0+1)^* 0(0+1)^* 1(0+1)^* + (0+1)^* 1(0+1)^* 0(0+1)^*$$

12) Describe the following by RE

a) L_1 = the set of all strings of 0's and 1's ending in 00.

b) L_2 = the set of all strings of 0's and 1's beginning with 0 and ending with 1

AU MAY 2004

a) $R_1 = (0+1)^* 00$

b) $R_2 = 0(0+1)^* 1$

13) Find the RE for

- a) The language of all strings containing exactly two 0's
- b) The language of all strings containing at least two 0's
- c) The language of all string that do not end with 01.

Solution

- a) $R_1 = 1^* 0 1^* 0 1^*$
- b) $R_2 = 1^* 0 1^* 0 (1+0)^*$
- c) $R_3 = (1+0)^* (00 + 11 + 10)$

CONVERTING DFA TO RE**THEOREM**

If L is a language defined by some DFA, then there is a regular expression defining L .

(or)

If $L = L(A)$ for some DFA, then there is a RE, R such that $L = L[R]$

PROOF

Let A be a DFA which defines a language L . Assume that A has „n“ states.

Let us construct the RE of the form, $R_{ij}^{(k)}$, whose language is the set of strings, w that takes the DFA from state i to j without going to any state, numbered greater than k .

We construct $R_{ij}^{(k)}$ by induction, k as follows

Basis

Let $k = 0$ [since all the states are numbered 1]. The Restriction on paths is that the path must have no intermediate states at all.

There are only two kinds of paths that need such a condition,

- An arc from state i to state j .
- A path of length zero, that consists of only some state i .

Case 1:- if $i \neq j$

Only (i) is possible. We must examine the DFA, A and find those input symbol, a such that there is a transition from state, i to j on symbol, a

- a) If there is no such symbol, a then $R_{ij}^{(0)} = \emptyset$

- b) If there is exactly one such symbol, a then $R_{ij}^{(0)} = a$
- c) If there are symbols a_1, a_2, \dots, a_k that label arcs from state i to j then $R_{ij}^{(0)} = a_1 + a_2 + a_3 + \dots + a_k$

Case 2:- if $i = j$

The legal paths are the path of length zero and all loops from I to itself.

The path of length zero is represented by the regular expression, ϵ since the path has no symbols along it. Thus we add, ϵ to the various expressions derived in (a) through (c) above.

i.e., Case (a) → No symbol „a“ → The expression becomes ϵ .

Case (b) → One symbol „a“ → The expression becomes $\epsilon + a$

Case (c) → Multiple symbols „a“ → The expression becomes $\epsilon + a_1 + a_2 + a_3 + \dots + a_k$

Induction

Suppose there is a path from state i to state j that goes through no state higher than K. There are two possible cases to consider.

Case 1

The path does not go through state k at all. In this case, the label of the path is in the language of R_{ij}^{k-1}

Case 2

The path goes through state k at least once. Then we can break the path into several pieces. The first goes from state I to state k without passing through k.

The Last piece goes from k to j without passing through k and all the pieces in the middle go from k to itself without passing through k. If a path goes through state k only once, then there are not middle pieces just a path from I to k and a path from k to I.

The set of labels for all paths of this type is represented by the regular expression $R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1}$, i.e., the first expression represents the part of the path that gets to state k, the first time, the second represents the quotient that goes from k to itself „0,, times, once or more than once. And the third expression represents the part of the path that leaves k for the last time and goes to state, j.

When we combine the expressions for the paths of the two types, we have the expression

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

Therefore, by induction hypothesis, we can conclude that the RE R_{ij}^n can be constructed for all values of i, j and n . The RE for the language defined by the DFA is the sum of all expressions of all expressions R_{ij}^n such that state j is an accepting state.

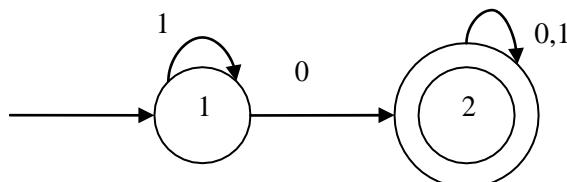
Some Simple Results of RE

1. $(\epsilon + 1)^* = 1^*$	9. $\phi \cdot 0^* = \phi$	17. $\epsilon + 1 = 1 + \epsilon$
2. $1^*(\epsilon + 1)^* = 1^*$	10. $\phi \cdot 1^* = \phi$	18. $\epsilon + \phi = \phi + \epsilon$
3. $(\epsilon + 1)^* = 1^*$	11. $\phi + \phi = \phi$	19. $\epsilon \cdot \epsilon^* = \text{Nothing}$
4. $(\epsilon + 1) + 1^* = 1^*$	12. $\phi + 0 = 0$	20. $\epsilon + \epsilon = \epsilon$
5. $0 + 0^* 1 = 1^* 0$	13. $\phi + 1 = 1$	21. $0 + 0 = 0$
6. $0 + 01^* = 01^*$	14. $\phi + 0^* = 0^*$	22. $1 + 1 = 1$
7. $\phi \cdot 0 = \phi$	15. $\phi + 1^* = 1^*$	23. $0 \cdot 0 = 00$
8. $\phi \cdot 1 = \phi$	16. $\epsilon + 0 = 0 + \epsilon$	24. $1 \cdot 1 = 11$
25. $0 \cdot \epsilon^* = 0$	26. $1 \cdot \epsilon^* = 1$	27. $\epsilon^* = \text{None}$

PROBLEMS

- 1) Obtain the RE that denotes the language as accepted by the following DFA.

AU DEC 2004



Solution: To find R_{ij}^n

$i \rightarrow$ start state = 1

$j \rightarrow$ Final state = 2

$k \rightarrow$ 0 to n [$n \rightarrow$ no of states] = 0,1,2

So, we have to find $\boxed{R_{12}^{(2)}}$

Basis Step

Step 1:

$$R_{ij}^{(0)} = a_1 + a_2 + \dots + a_k ; i \neq j \\ \in + a_1 + a_2 + \dots + a_k ; i = j$$

i takes the values 1,2

j takes the values 1,2

$$R_{11}^{(0)} = \underline{\in + 1} \quad [i = j]$$

$$R_{12}^{(0)} = \underline{0} \quad [i \neq j]$$

$$R_{12}^{(0)} = \underline{\phi} \quad [i \neq j]$$

$$R_{22} = \underline{\in + 0 + 1} \quad [i = j]$$

Inductive Step

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

k takes the values 1 and 2. For k=0, basis step has already been calculated.

Step 2:

k = 1

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{il}^{(0)} (R_{11}^{(0)}) * R_{lj}^{(0)}$$

$$i=1; j=1 \Rightarrow R_{11}^{(1)} = R_{11}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)}) * R_{11}^{(0)}$$

$$= (\underline{\in + 1}) + [(\underline{\in + 1}) (\underline{\in + 1}) * (\underline{\in + 1})]$$

$$= (\underline{\in + 1}) + [\underline{(\in + 1) * (\in + 1)}] \leftarrow (\in + 1) * = 1 *$$

$$= (\underline{\in + 1}) + 1 * (\underline{\in + 1}) \quad \leftarrow (\in + 1) * = 1 *$$

$$= (\underline{\in + 1}) + 1 * \quad \leftarrow 1 * (\in + 1) * = 1 *$$

$$= 1 * \quad (\in + 1) * + 1 * = 1 *$$

$$\begin{aligned}
 i=1; j=2 \Rightarrow R_{21}^{(1)} &= R_{12}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)} \\
 &= 0 + [(\epsilon+1)(\underline{\epsilon+1})^*(0)] \\
 &= 0 + [(\underline{\epsilon+1}) 1^*(0)] \quad \leftarrow (\epsilon+1)^* = 1^* \\
 &= 0 + [1^* 0] \quad \leftarrow (\epsilon+1)^* = 1^* \\
 &= \boxed{1^* 0} \quad \leftarrow 0 + 1^* = 1^* 0
 \end{aligned}$$

$$\begin{aligned}
 i=2; j=1 \Rightarrow R_{21}^{(\emptyset)} &= R_{21}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)} \\
 &= \phi + [\phi + (\underline{\epsilon+1})^*(\epsilon+1)] \\
 &= \phi + [\phi \cdot 1^* (\underline{\epsilon+1})] \quad \leftarrow (\epsilon+1)^* = 1^* \\
 &= \phi + [\phi \cdot 1^*] \quad \leftarrow 1^* (\epsilon+1)^* = 1^* \\
 &= \phi + \phi \quad \leftarrow \phi \cdot 1^* = \phi \\
 &= \phi \quad \leftarrow \phi + \phi = \phi
 \end{aligned}$$

$$\begin{aligned}
 i=2; j=2:- \Rightarrow R_{22}^{(1)} &= R_{22}^{(0)} + [R_{21}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)}] \\
 &\quad \underline{(\epsilon+0+1)} + [\phi \cdot (\underline{\epsilon+1})^* \cdot 0] \\
 &\quad \underline{(\epsilon+0+1)} + \phi \quad \leftarrow \phi \cdot \text{anything} = \phi \\
 &\quad \epsilon+0+1 \quad \leftarrow \phi + \text{Something} = \text{something}
 \end{aligned}$$

Step 3:

k = 2

To find $\boxed{R_{22}^{(2)}}$

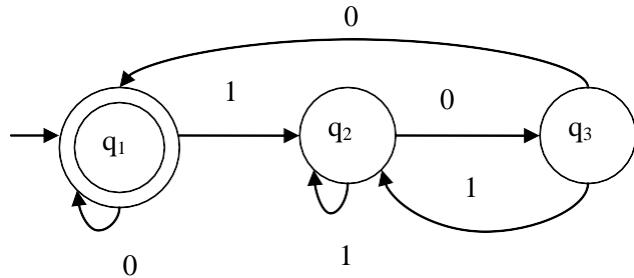
$$R_{ij}^{(2)} = R_{ij}^{(1)} + R_{i2}^{(1)} R_{22}^{(1)} R_{2j}^{(1)}$$

$$\begin{aligned}
 i=1; j=2 \Rightarrow R_{12}^{(2)} &= R_{12}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{22}^{(1)} \\
 &= 1^* 0 + [(1^* 0) (\underline{\epsilon+0+1})^* (\epsilon+0+1)] \\
 &= 1^* 0 + [(1^* 0) (0+1)^* (\underline{\epsilon+0+1})] \quad \leftarrow (\epsilon+1)^* = 1^* \\
 &= 1^* 0 + [(1^* 0) (0+1)^*] \quad \leftarrow 1^* (\epsilon+1) = 1^* \\
 &= \boxed{(1^* 0) (0+1)^*} \quad \leftarrow 0+01^* = 01^*
 \end{aligned}$$

∴ The Required regular expression is $R_{12}^{(2)} = 1^* 0 (0+1)^*$

2) Construct a RE corresponding to the state diagram given in the following figure.

AU MAY 2005



Solution:

Here;

$$i = 1$$

$$j = 1$$

$$k = 3$$

To find: $R_{ij}^{(k)} = R_{11}^{(3)}$

Basis

Step 1:

$$\begin{aligned} R_{ij}^{(0)} &= a_1 + a_2 + a_3 + \dots + a_k ; i \neq j \\ &\in + a_1 + a_2 + \dots + a_k ; i = j \end{aligned}$$

$$R_{11}^{(0)} = \in + 0$$

$$R_{22}^{(0)} = \emptyset$$

$$R_{31}^{(0)} = 0$$

$$R_{12}^{(0)} = 1$$

$$R_{22}^{(0)} = \in + 1$$

$$R_{32}^{(0)} = 1$$

$$R_{13}^{(0)} = \emptyset$$

$$R_{23}^{(0)} = 0$$

$$R_{33}^{(0)} = \in$$

Inductive step

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

$k = 0 \Rightarrow$ computed in step 1

$k = 1, 2, 3 \Rightarrow$ computed in step 2, 3 and 4

i takes the values 1, 2, 3

j takes the values 1, 2, 3.

Step 2:

k = 1

$$\begin{aligned}
 R_{ij}^{(1)} &= R_{ij}^{(0)} + R_{i1}^{(0)} (R_{11}^{(0)})^* R_{1j}^{(0)} \\
 R_{11}^{(1)} &= R_{11}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)} \\
 &= (\in + 0) + [(\in + 0) \underline{(\in + 0)^*} (\in + 0)] \\
 &= (\in + 0) + [(\in + 0) \underline{0^*} (\in + 0)] \quad \leftarrow \in + 0^* \\
 &= (\in + 0) + [(\in + 0) \underline{0^*}] \quad \leftarrow 0^* (\in + 0) = 0^* \\
 &= (\in + 0) \underline{[0^*]} \quad \leftarrow (\in + 0) 0^* = 0^* \\
 &= 0^* \quad \leftarrow (\in + 0) + 0^* = 0^* \\
 R_{12}^{(1)} &= R_{12}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)} \\
 &= 1 + [(\in + 0) \underline{(\in + 0)^*} (1)] \\
 &= 1 + [(\in + 0) + (0)^* 1] \quad \leftarrow (\in + 0)^* = 0 \\
 &= \underline{1 + [0^* 1]} \quad \leftarrow (\in + 0) 0^* = 0^* \\
 &= 0^* 1 \quad \leftarrow 1 + 0^* 1 = 0^* 1 \\
 R_{13}^{(1)} &= R_{13}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)})^* R_{13}^{(0)} \\
 &= \underline{\phi + [(\in + 0) (\in + 0)^* \cdot \phi]} \quad \leftarrow \phi \cdot a = \phi \\
 &= \phi \quad \leftarrow \phi + \phi = \phi \\
 R_{22}^{(1)} &= R_{22}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)} \\
 &= \phi + [\phi \cdot (\in + 0)^* \cdot (\in + 0) \leftarrow \phi \cdot a = \phi] \\
 &= \phi \quad \leftarrow \phi + \phi = \phi \\
 R_{22}^{(1)} &= R_{22}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)} \\
 &= (\in + 1) + [\underline{\phi \cdot (\in + 0)^* \cdot 1}] \\
 &= \underline{(\in + 0) + \phi} \quad \leftarrow \phi \cdot a = \phi \\
 &= (\in + 1) \quad \leftarrow \phi + a = a
 \end{aligned}$$

$$R_{23}^{(1)} = R_{23}^{(0)} + R_{21}^{(0)}(R_{11}^{(0)}) * R_{13}^{(0)}$$

$$= 0 + [\phi.(\in + 0)^*. \phi]$$

$$= \underline{0 + \phi} \quad \leftarrow \phi. a = \phi$$

$$= 0 \quad \leftarrow \phi + a = a$$

$$R_{31}^{(1)} = R_{31}^{(0)} + R_{31}^{(0)}(R_{11}^{(0)}) * R_{11}^{(0)}$$

$$= 0 + [\underline{0.0 * (\in + 0)^*(\in + 0)}]$$

$$= 0 + [0.0^*(\in + 0)] \quad \leftarrow (\in + 0)^* = 0^*$$

$$= 0 + [0.0^*.] \quad \leftarrow 0 + 00^* = 00^*$$

$$R_{32}^{(1)} = R_{32}^{(0)} + R_{31}^{(0)}(R_{11}^{(0)}) * R_{12}^{(0)}$$

$$= 1 + [\underline{(\in + 0)^* 1}]$$

$$= 1 + [0.0^* 1] \quad \leftarrow (\in + 0)^* = 0^*$$

$$= 1 + 00^* 1$$

$$R_{33}^{(1)} = R_{33}^{(0)} + R_{31}^{(0)}(R_{11}^{(0)}) * R_{13}^{(0)}$$

$$= \in + [0.(\in + 0)^* \phi]$$

$$= \in + \phi$$

$$= \in$$

Step 3:

$k = 2$

To find: $R_{11}^{(3)} = R_{11}^{(2)} + R_{13}^{(2)}(R_{33}^{(2)}) * R_{31}^{(2)}$

So at $k = 2$; it is enough to find $R_{11}^{(2)}$, $R_{13}^{(2)}$, $R_{33}^{(2)}$, $R_{31}^{(2)}$

$$R_{ij}^{(2)} = R_{ij}^{(1)} + R_{i2}^{(1)}(R_{22}^{(21)}) * R_{2j}^{(2)} \quad R_{11}^{(2)} = R_{11}^{(1)} + R_{12}^{(1)}(R_{22}^{(1)}) * R_{21}^{(1)}$$

$$= 0^* + [(0^* 1)(\in + 0)^*(\phi)]$$

$$\begin{aligned}
 R_{11}^{(2)} &= 0^* + \phi & \leftarrow \phi \cdot a = \phi \\
 &= 0^* & \leftarrow \phi + a = a \\
 R_{13}^{(2)} &= R_{13}^{(1)} + R_{12}^{(1)} R_{22}^{(1)} R_{23}^{(1)} \\
 &= \phi + [0^* 1] (\underline{\epsilon + 1})^* . 0 \\
 &= \phi + [(0^* 1) (1^*)] & \leftarrow (\epsilon + 1)^* = 1^* \\
 &= 0^* 1 1^* 0 & \leftarrow \phi + a = a \\
 R_{33}^{(2)} &= R_{33}^{(1)} + R_{32}^{(1)} R_{22}^{(1)} R_{23}^{(1)} \\
 &= \epsilon + [(1 + 00^* 1) (\underline{\epsilon + 1})^* (0)] \\
 &= \epsilon + [(1 + 00^* 1) 1^* 0] & \leftarrow (\epsilon + 1)^* = 1^* \\
 &= \epsilon + [(1 + 00^* 1) 1^* 0] & \leftarrow \epsilon + a = \epsilon + a \\
 R_{31}^{(2)} &= R_{31}^{(1)} + R_{32}^{(1)} R_{22}^{(1)} R_{21}^{(1)} \\
 &= (00^*) + [(\underline{100^* 1}) (\epsilon + 1)^* (\phi)] \\
 &= 00^* + \phi & \leftarrow \phi \cdot a = \phi \\
 &= 00^* & \leftarrow \phi + a = \phi
 \end{aligned}$$

Step 4:

$k = 3$

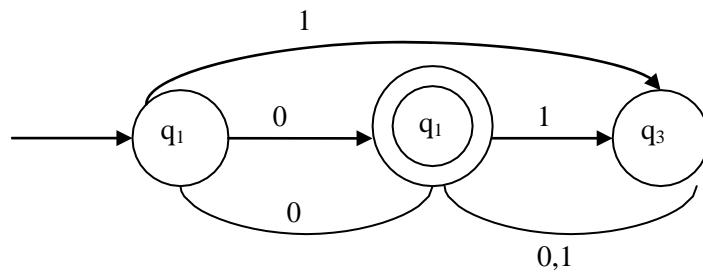
$$\begin{aligned}
 \text{To find: } R_{11}^{(3)} &= R_{11}^{(2)} + R_{13}^{(2)} (R_{33}^{(2)})^* R_{31}^{(2)} \\
 &= 0^* + [(0^* 1 1^* 0) (\epsilon + (1 + 00^* 1) 1^* 0)^* (00^*)]
 \end{aligned}$$

The Required Regular Expression is given as,

$$R_{11}^{(3)} = 0^* + (0^* 1 1^* 0) (\epsilon + (1 + 00^* 1) 1^* 0)^* 00^*$$

3) Obtain a Regular Expression that denotes the language accepted by

AU MAY 2005, DEC 2006, DEC 2009



Solution:

To find: R_{ij}^k

$i = 1 \leftarrow$ Initial State.

$j = 2 \leftarrow$ Final state

$k = 3 \leftarrow$ No of states.

$$\therefore R_{1z}^{(3)} = R_{12}^{(2)} + R_{13}^{(2)} (R_{33}^{(2)})^* R_{32}^{(2)}$$

Basis

Step 1:

$$R_{ij}^{(0)} = a_1 + a_2 + \dots + a_k ; i \neq j \\ \in + a_1 + a_2 + \dots + a_k ; i = j$$

$$R_{11}^{(0)} = \in \quad R_{21}^{(0)} = 0 \quad R_{31}^{(0)} = \phi$$

$$R_{12}^{(0)} = 0 \quad R_{22}^{(0)} = \in \quad R_{32}^{(0)} = 0 + 1$$

$$R_{13}^{(0)} = 1 \quad R_{23}^{(0)} = 1 \quad R_{33}^{(0)} = \in$$

Inductive step

i takes the values 1,2,3

j takes the values 1,2,3

k takes the values 0,1,2,3

Step 2:

$k = 1$

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{il}^{(0)} (R_{11}^{(0)})^* R_{1j}^{(0)}$$

$$R_{11}^{(1)} = R_{11}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)}$$

$$= \in + [\underline{\in (\in)^* \in}] \quad \leftarrow \in . \in = \text{None}$$

$$= \underline{\in + \in} \quad \leftarrow \in + \in = \in$$

$$= \in$$

← →

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)}(R_{11}^{(0)})^*R_{12}^{(0)}$$

$$= 0 + [\underline{\in (\in)^* 0}] \quad \leftarrow \in . \in^* = \text{None}$$

$$= \underline{0 + 0} \quad \leftarrow 0 + 0 = 0$$

$$= 0$$

$$R_{13}^{(1)} = R_{13}^{(0)} + R_{11}^{(0)}(R_{11}^{(0)})^*R_{13}^{(0)}$$

$$= 1 + [\underline{\in (\in)^* 1}] \quad \leftarrow \in . \in^* = \text{None}$$

$$= \underline{1 + 1} \quad \leftarrow 1 + 1 = 1$$

$$= 1$$

$$R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)}(R_{22}^{(0)})^*R_{11}^{(0)}$$

$$= 0 + [\underline{0(\in)^* \in}]$$

$$= 0 + 0 \quad \leftarrow \in . \in^* = \text{None}$$

$$= 0 \quad \leftarrow 0 + 0 = 0$$

$$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)}(R_{11}^{(0)})^*R_{12}^{(0)}$$

$$= \in + [\underline{0(\in)^*.0}]$$

$$= \in + 00 \quad \leftarrow \in^*.0 = 0$$

$$R_{23}^{(1)} = R_{23}^{(0)} + R_{21}^{(0)}(R_{11}^{(0)})^*R_{13}^{(0)}$$

$$= 1 + [\underline{0(\in)^* 1}]$$

$$= 1 + 01 \quad \leftarrow \in^*.1 = 1$$

$$R_{31}^{(1)} = R_{31}^{(0)} + R_{31}^{(0)}(R_{11}^{(0)})^*R_{11}^{(0)}$$

$$= \phi + [\underline{\phi(\in)^* . \in}]$$

$$= \underline{\phi + \phi} \quad \leftarrow \phi.a = \phi$$

$$= \phi \quad \leftarrow \phi + \phi = \phi$$

$$\begin{aligned}
 R_{32}^{(1)} &= R_{32}^{(0)} + R_{31}^{(0)}(R_{11}^{(0)})^* R_{12}^{(0)} \\
 &= (0+1) + [\underline{\phi(\in)} * .0] \\
 &= \underline{(0+1)+\phi} \quad \leftarrow \phi. a = \phi \\
 &= \underline{(0+1)+\phi} \quad \leftarrow \phi. a = \phi \\
 R_{33}^{(1)} &= R_{33}^{(0)} + R_{31}^{(0)}(R_{11}^{(0)})^* R_{13}^{(0)} \\
 &= \underline{\epsilon + [\phi(\in)^* 1]} \\
 &= \underline{\epsilon + 0} \quad \leftarrow \phi. a = \phi \\
 &= \epsilon \quad \leftarrow \epsilon + \phi = \epsilon
 \end{aligned}$$

Step 3:

$k = 2$

$$\text{To find : } R_{12}^{(3)} = R_{12}^{(2)} + R_{13}^{(2)}(R_{33}^{(2)})^* R_{32}^{(2)}$$

∴ At $k = 2$; It is enough to find $R_{12}^{(2)}$, $R_{13}^{(2)}$, $R_{33}^{(2)}$ and $R_{32}^{(2)}$

$$\begin{aligned}
 R_{ij}^{(2)} &= R_{ij}^{(1)} + R_{i2}^{(1)} R_{22}^{(1)*} R_{2j}^{(1)} \\
 R_{12}^{(2)} &= R_{12}^{(1)} + R_{12}^{(1)} R_{22}^{(1)*} R_{22}^{(1)} \\
 &= 0 + [0(\underline{\epsilon + 00})^* (\underline{\epsilon + 00})] \\
 &= 0 + [0(00)^* (\underline{\epsilon + 00})] \quad \leftarrow (\epsilon + 1)^* = 1^* \\
 &= 0 + [0(00)^*] \quad \leftarrow 1^*(\epsilon + 1) = 1^* \\
 &= 0(00)^* \quad \leftarrow 0 + 01^* = 01^* \\
 &= 0(00)^*
 \end{aligned}$$

$$\begin{aligned}
 R_{13}^{(2)} &= R_{13}^{(1)} + R_{12}^{(1)}(R_{22}^{(1)})^* R_{23}^{(1)} \\
 &= 1 + [0(\underline{\epsilon + 00})^* (1 + 01)] \\
 &= 1 + [0(00)^* (1 + 01)] \quad \leftarrow (\epsilon + 00)^* = (00)^*
 \end{aligned}$$

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

$$\mathbf{R}_{33}^{(2)} = \mathbf{R}_{33}^{(1)} + \mathbf{R}_{32}^{(1)} (\mathbf{R}_{22}^{(1)})^* \mathbf{R}_{23}^{(1)}$$

$$= \in + [(0\ 1) \underline{(\in + 00)^*} (1 + 01)]$$

$$=1+[0(00)^*(1+01)] \quad \leftarrow (\in + 00)^* = (00)^*$$

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

$$\mathbf{R}_{32}^{(2)} = \mathbf{R}_{32}^{(1)} + \mathbf{R}_{32}^{(1)} (\mathbf{R}_{22}^{(1)})^* \mathbf{R}_{22}^{(1)}$$

$$= (0+1) + [(0+1)(\underline{\epsilon+00})^*(\epsilon+00)]$$

$$= (0 + 1) + [(0 + 1)(00)^*(\in + 00)] \quad \leftarrow (\in + 00)^* = (00)^*$$

$$= (0+1) + [(0+1)(00)^*]$$

$$\leftarrow(00)^*(\in + 00) = (00)^*$$

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

$\leftarrow 0 + 01^* = 01^*$

Step 4:

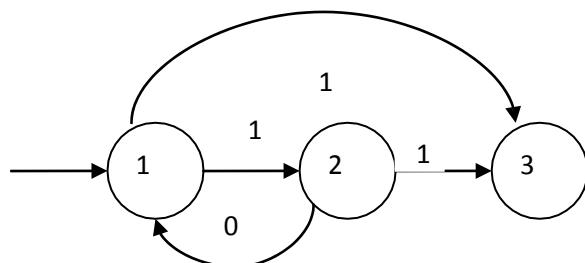
$$k = 3$$

$$R_{12}^{(3)} = R_{12}^{(2)} + R_{13}^{(1)} (R_{33}^{(2)})^* R_{32}^{(2)}$$

$$= 0(00)^* + [(1 + 0(00)^*(1 + 0) (\in + (0 + 1)(00)^*(101)^*(0 + 1)(00)^*)]$$

4. Find the RE, $R_{23}^{(2)}$ from the following DFA.

AU DEC 2007



Solution:

To find: $R_{23}^{(2)} = R_{23}^{(1)} + R_{22}^{(1)}(R_{22}^{(1)})^* R_{23}^{(1)}$

Basis

Step 1:

$$R_{ij}^{(0)} = \frac{a_1 + a_2 + \dots a_k}{k}; i \neq j$$

$$\in + a_1 + a_2 + \dots a_k; i = j$$

$$\begin{array}{lll}
 R_{11}^{(0)} = \phi + \epsilon = \epsilon & R_{21}^{(0)} = 0 & R_{31}^{(0)} = \phi \\
 R_{j2}^{(0)} = 1 & R_{22}^{(0)} = \epsilon & R_{32}^{(0)} = \phi \\
 R_{13}^{(0)} = 1 & R_{23}^{(0)} = 1 & R_{33}^{(0)} = \epsilon
 \end{array}$$

Inductive Step

We Require $R_{23}^{(1)}, R_{22}^{(1)}$

Step 2:

$k = 1$

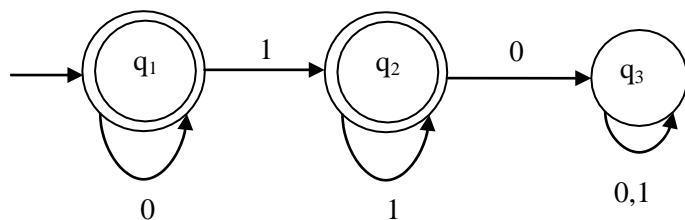
$$\begin{aligned}
 R_{ij}^{(1)} &= R_{ij}^{(0)} + R_{il}^{(0)} (R_{11}^{(0)})^* R_{lj}^{(0)} \\
 R_{23}^{(1)} &= R_{23}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{13}^{(0)} \\
 &= 1 + [\underline{0 (\epsilon)^* 1}] \quad \leftarrow \epsilon^* = \text{None} \\
 &= 1 + 01 \\
 R_{22}^{(1)} &= R_{22}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)} \\
 &= \epsilon + [\underline{0 (\epsilon)^* 1}] \\
 &= \epsilon + 01 \quad \leftarrow \epsilon^* = \text{None}
 \end{aligned}$$

Step 3:

$k = 2$

$$\begin{aligned}
 R_{23}^{(2)} &= R_{23}^{(1)} + R_{22}^{(1)} (R_{22}^{(1)})^* R_{23}^{(1)} \\
 &= (1 + 01) + [(\epsilon + 01)(\epsilon + 01)^*(1 + 01)] \\
 &= (1 + 01) + [(\epsilon + 01)(01)^*(1 + 01)] \\
 \text{R.E.} &= (1 + 01)[\epsilon + (\epsilon + 01)(01)^*]
 \end{aligned}$$

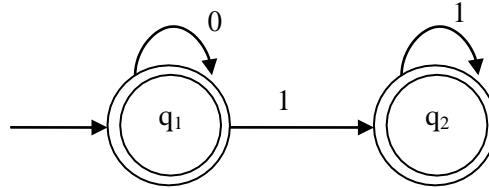
5. Find the language accepted by the following automaton



Solution:

The language accepted by the above DFA is $R_{11}^{(2)} + R_{12}^{(2)}$ [since there are two final states]

The state q_3 is eliminated since it is a trap/dead state.


Basis
Step 1:

$$\begin{aligned} R_{ij}^{(0)} &= a_1 + a_2 + a_3 + \dots + a_k ; i \neq j \\ &\in + a_1 + a_2 + a_3 + \dots + a_k ; i = j \end{aligned}$$

$$R_{11}^{(0)} = \in + 0 \quad R_{21}^{(0)} = \phi$$

$$R_{12}^{(0)} = 1 \quad R_{22}^{(0)} = \in + 1$$

Step 2:

$$k = 1$$

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{21}^{(1)}$$

$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{22}^{(1)}$$

\therefore It is enough to find $R_{11}^{(0)}, R_{12}^{(1)}, R_{21}^{(1)}, R_{22}^{(1)}$

$$\begin{aligned} R_{11}^{(1)} &= R_{11}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)} \\ &= (\in + 0) + [(\in + 0)(\in + 0)^*(\in + 0)] \\ &= (\in + 0) + [(\in + 0)(0)^*(\in + 0)] \quad \leftarrow (\in + 1)^* = 1^* \\ &= (\in + 0) + [(\in + 0)(0)^*] \quad \leftarrow 1^*(\in + 1) = 1^* \\ &= (\in + 1) + (0)^* \quad \leftarrow 1^*(\in + 1) = 1^* \\ &= 0^* \quad \leftarrow (\in + 1) + 1^* = 1^* \end{aligned}$$

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)}$$

$$= 1 + [(\in + 0)(\in + 0)^* 1]$$

$$= 1 + [\underline{(\epsilon + 0)(0)^* 1}] \quad \leftarrow (\epsilon + 0)^* = 0^*$$

$$= \underline{1 + [(0)^* 1]} \quad \leftarrow (\epsilon + 0) 0^* = 0^*$$

$$= 0^* 1 \quad \leftarrow 1 + 0^* 1 = 0^* 1$$

$$R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)}$$

$$= \underline{\phi + \phi} \quad \leftarrow \phi \cdot a = \phi$$

$$= \phi \quad \leftarrow \phi + \phi = \phi$$

$$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)}$$

$$= (\epsilon + 1) + [\underline{\phi (\epsilon + 0)^* 1}]$$

$$= (\epsilon + 1) + \phi \quad \leftarrow \phi \cdot a = \phi$$

$$= (\epsilon + 1) \quad \leftarrow \phi + a = a$$

Step 3:

k = 2

$$R_{11}^{(2)} = R_{11}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{21}^{(1)}$$

$$= 0^* + [\underline{0^* 1 (\epsilon + 1)^* \phi}]$$

$$= \underline{0^* + \phi} \quad \leftarrow \phi \cdot a = \phi$$

$$= 0^* \quad \leftarrow \phi + a = a$$

$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{22}^{(1)}$$

$$= 0^* 1 + [(0^* 1) (\epsilon + 1)^* (\epsilon + 1)]$$

$$= 0^* 1 + [\underline{(0^* 1) (1)^* (\epsilon + 1)}] \quad \leftarrow (\epsilon + 1)^* = 1^*$$

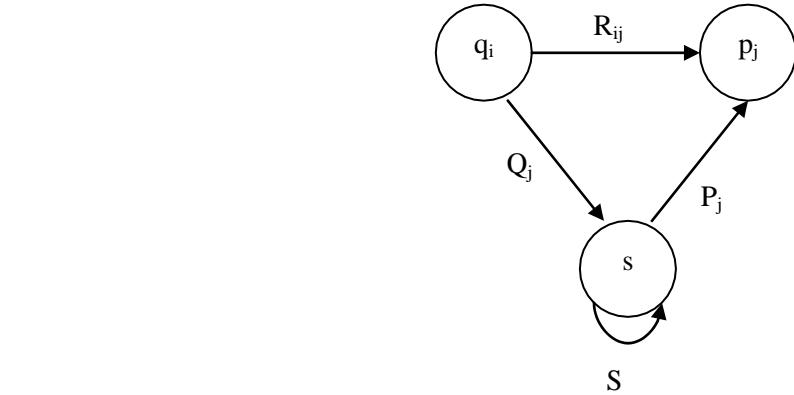
$$= \underline{0^* 1 + [(0^* 1) (1^*)]} \quad \leftarrow 1^* (\epsilon + 1) = 1^*$$

$$= 0^* 11^* \quad \leftarrow 0 + (1^* = 01^*)$$

$$\text{The R.E} \Rightarrow R_{11}^{(2)} + R_{12}^{(2)} + 0^* + 0^* 11^*$$

CONSTRUCTING DFA BY ELIMINATING STATES

Consider the following setup in which the state „S“ is to be eliminated.

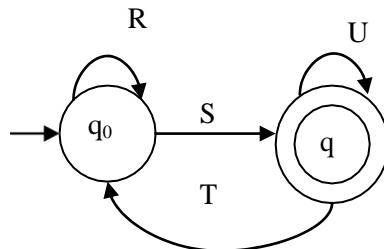


Assume that the state, q_i is predecessors of the state S . p_j is the successor of s . When we eliminate the state s , the regular expression, is labeled to R from Q_i to P_j . Note that Q_i and P_j may also be same.

$$R_{ij} + Q_i S^* P_j$$

The construction of a R. E. from F.A. is as follows:

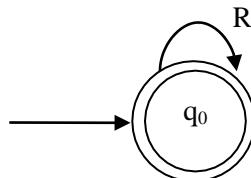
- (i) For each accepting state Q , eliminate all states except Q , on the start state q_0 . By applying the above reduction process to produce an equivalent automaton with regular expressions labels on the arcs.
- (ii) Suppose, if $Q \neq q_0$ i.e., accepting state and start state are distinct, then finally we have two state automaton.



We can represent the accepting string of this two-state automaton in many ways. In general, we take the string accepted by this automaton, of the form;

$$(R + SU^*T)^* SU^*$$

- (iii) If the start state and the accepting state are the same, then we are left with one-state automaton.



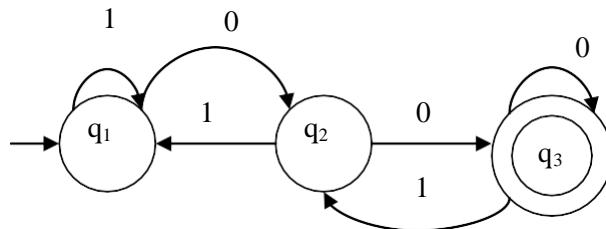
The regular expression denoting the language that it accepts is, R^*

- (iv) The derived regular expression is the sum of all expressions from the reduced automaton for each accepting states by rules (ii) and (iii)

PROBLEMS

- 1) Construct the transition diagram for the given DFA and give a R.E by using state elimination technique.

$\Sigma \backslash Q$	0	1
Q		
$\rightarrow q_1$	{q ₂ }	{q ₁ }
q ₂	{q ₃ }	{q ₁ }
* q ₃	{q ₃ }	{q ₂ }



Solution:-

Here, q₁ → Start State

q₃ → Final State

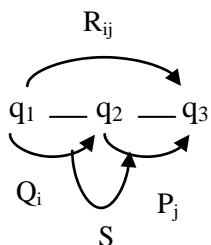
∴ The state to be eliminated is q₂.

We use the Regular Expression: R_{ij} + Q_iS*P_j is labeled for the arcs from q₁ to q₃.

To be found

- q₁-q₂-q₃
- q₃-q₂-q₁
- q₁-q₂-q₁
- q₃-q₂-q₁

Step -1:- Eliminating q₂ from q₁-q₂-q₃



← →

Here,

$$R_{ij} = \emptyset$$

$$Q_i = 0$$

$$S = \emptyset$$

$$P_j = 0$$

$$\therefore r_1 = \emptyset + 0 \cdot \underline{\emptyset^* 0}$$

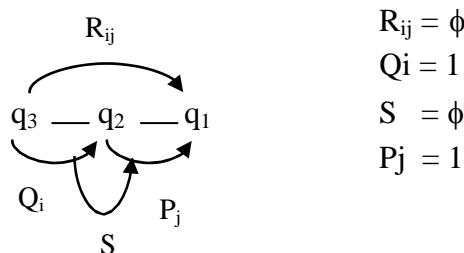
$$= \underline{\emptyset + 00}$$

← $\emptyset^* = \text{None}$

$$r_1 = 00$$

← $\emptyset + a = a$

Step 2: Eliminating q_2 from $q_3 - q_2 - q_1$

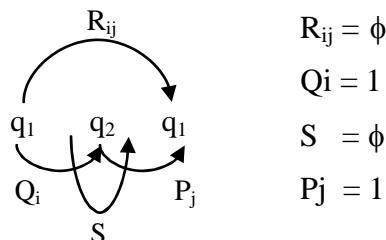


$$r_2 = \emptyset + 1 \cdot \emptyset^* 1$$

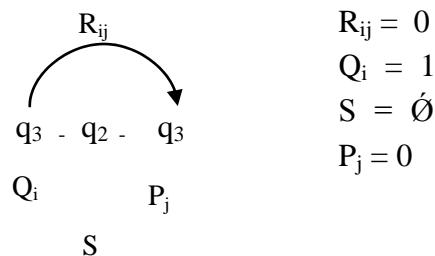
$$= \underline{\emptyset + 11} \quad \leftarrow \emptyset^* = \text{None}$$

$$r_2 = 11 \quad \leftarrow \emptyset + a = a$$

Step 3: Eliminating q_2 from $q_1 - q_2 - q_1$



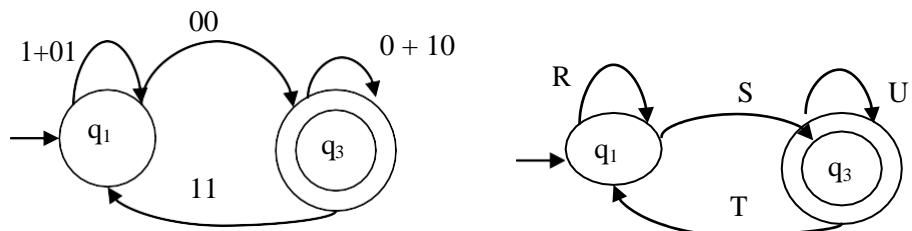
Step 4: Eliminating q_2 from $q_3 - q_2 - q_3$



$$r_4 = 0 + 1 \cdot \phi^* \cdot 0$$

$$r_4 = 0 + 10 \quad \leftarrow \phi^* = \text{None}$$

∴ The \$ Two – State Automaton:-

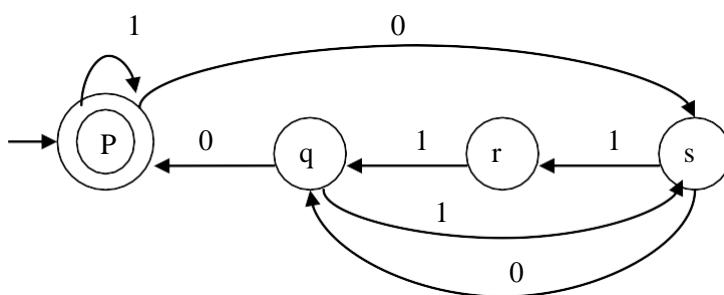


$$\text{The R.E} = (R + SU^* T)^* SU^*$$

$$\text{R.E} = ((1 + 01) + (00)(0 + 10)^*(11))^* 00(0 + 10)$$

2) Convert the following DFA to RE using state elimination technique.

Σ	0	1
Q	0	1
$* \rightarrow p$	s	p
q	p	s
r	r	q
s	q	r



Solution:-

[First eliminate the state that is far away from start state].

States to be eliminated: s, r, q

Initial / start state = Final start = P \Rightarrow cannot be eliminated

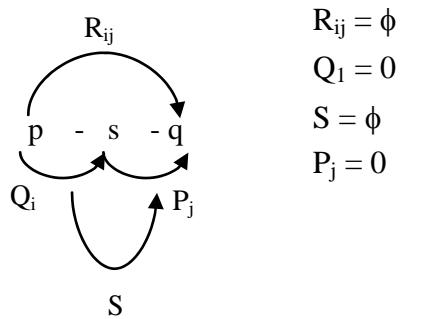
$$R.E = R_{ij} + Q_i S^* P_j$$

To find:

- $r_1 = p - s - q$
- $r_2 = q - s - p$
- $r_3 = p - s - r$
- $r_4 = r - s - p$
- $r_5 = q - s - r$
- $r_6 = r - s - q$
- $r_7 = p - s - p$
- $r_8 = q - s - r$
- $r_9 = r - s - r$

Step 1:

Elimination of s from p - s - q (r_1)



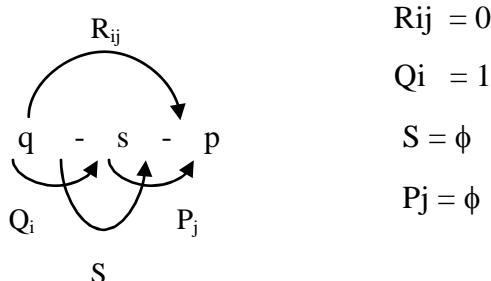
$$\begin{aligned} R_{ij} &= \emptyset \\ Q_i &= 0 \\ S &= \emptyset \\ P_j &= 0 \end{aligned}$$

$$r_1 = \emptyset + 0\emptyset^*0$$

$$s = \emptyset + 00 \quad \leftarrow \emptyset^* = \text{None}$$

$$r_1 = 00 \quad \leftarrow \emptyset + a = a$$

Elimination of s from q - s - p (r_2)



$$\begin{aligned} R_{ij} &= 0 \\ Q_i &= 1 \\ S &= \emptyset \\ P_j &= \emptyset \end{aligned}$$

$$r_2 = 0 + \underline{1\phi * \phi}$$

$$\begin{aligned} &= \underline{0 + \phi} & \leftarrow \phi \times a = \phi \\ &= 0 & \leftarrow \phi + a = a \end{aligned}$$

Eliminating s from p - s - r (r3)

	$R_{ij} = \phi$ $Q_i = 0$ $S = \phi$ $P_j = 1$	$r_3 = R_{ij} + Q_i S * P_j$ $= \phi + 0\phi * 1$ $= \phi + 01 \quad \leftarrow \phi^* = \text{None}$ $r_3 = 01 \quad \leftarrow \phi + a = a$
--	---	---

Elimination of s from r - s - p (r4)

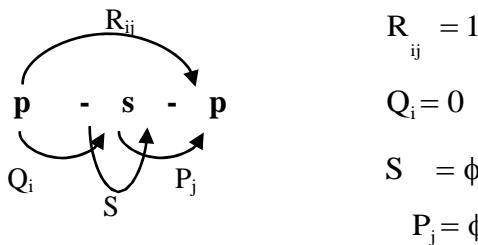
	$R_{ij} = \phi$ $Q_i = \phi$ $S = \phi$ $P_j = \phi$	$r_3 = R_{ij} + Q_i S * P_j$ $= \phi + \underline{\phi \phi * \phi}$ $= \underline{\phi + \phi} \quad \leftarrow \phi \cdot a = \phi$ $r_3 = \phi \quad \leftarrow \phi + \phi = \phi$
--	---	---

Elimination of s from q - s - r (r5)

	$R_{ij} = \phi$ $Q_i = 1$ $S = \phi$ $P_j = 1$	$r_5 = R_{ij} + Q_i S * P_j$ $= \phi + \underline{1\phi * 1}$ $= \underline{\phi + 11} \quad \leftarrow \phi^* = \text{None}$ $r_5 = 11 \quad \leftarrow \phi + a = a$
--	---	---

Elimination of s from r - s - q (r6)

	$R_{ij} = 1$ $Q_i = \phi$ $S = \phi$ $P_j = 0$	$r_6 = R_{ij} + Q_i S * P_j$ $= 1 + \underline{\phi \cdot \phi * 0}$ $= \underline{1 + \phi} 1 \quad \leftarrow \phi \cdot a = \phi$ $r_6 = 1 \quad \leftarrow \phi + a = a$
--	---	---

Elimination of s from p - s - p (r7)


$$R_{ij} = 1$$

$$Q_i = 0$$

$$S = \phi$$

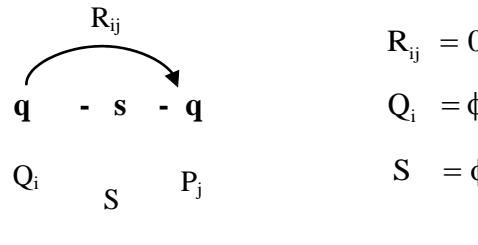
$$P_j = \phi$$

$$r_7 = R_{ij} + Q_i S^* P_j$$

$$= 1 + \underline{\phi \cdot \phi^* \phi}$$

$$= 1 + \underline{\phi} \quad \leftarrow \phi \cdot a = \phi$$

$$r_7 = 1 \quad \leftarrow \phi + a = a$$

Elimination of s from q - s - q (r8)


$$R_{ij} = 0$$

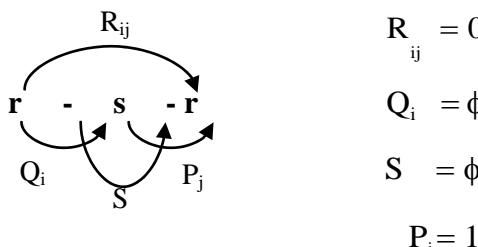
$$Q_i = \phi$$

$$S = \phi$$

$$r_8 = R_{ij} + Q_i S^* P_j$$

$$= 1 + \underline{1 \phi^* 0}$$

$$= \underline{\phi + 1 0} \quad \leftarrow \phi = \text{None}$$

Elimination of s from r - s - r (r9)


$$R_{ij} = 0$$

$$Q_i = \phi$$

$$S = \phi$$

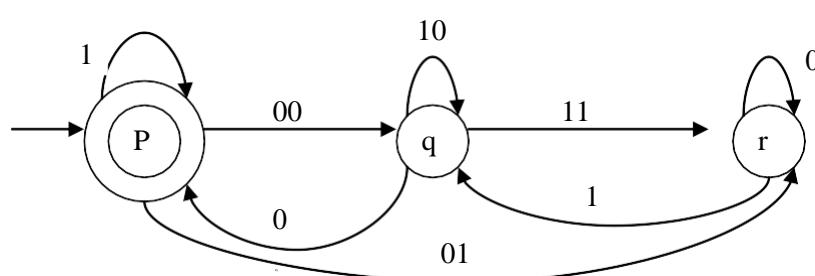
$$P_j = 1$$

$$r_9 = R_{ij} + Q_i S^* P_j$$

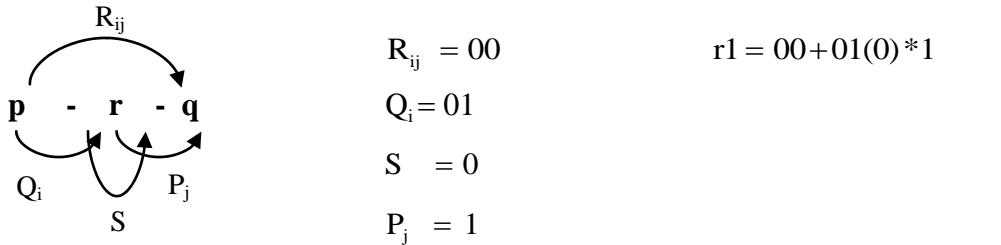
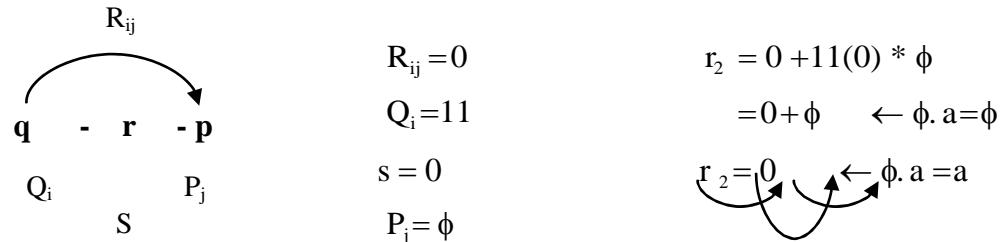
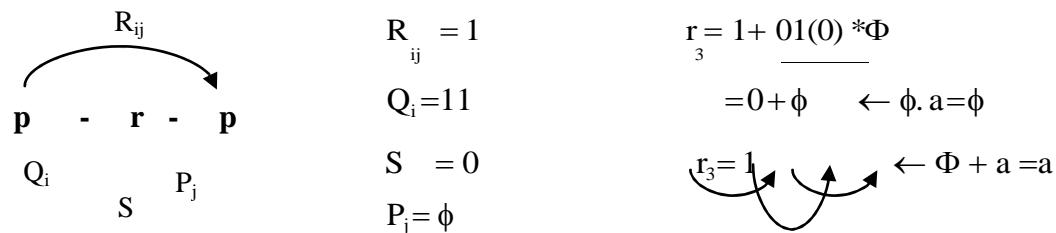
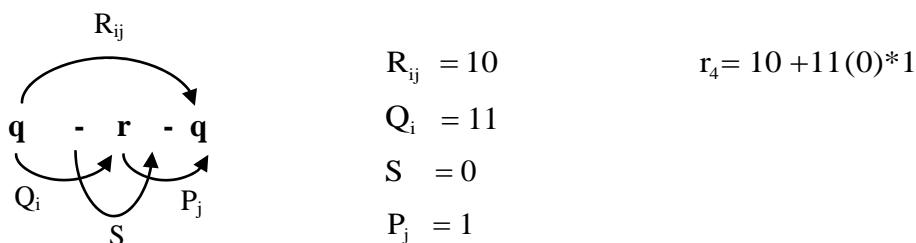
$$= 0 + \underline{\phi \phi^* 1}$$

$$= \underline{0 + \phi} \quad \leftarrow \phi \cdot a = \phi$$

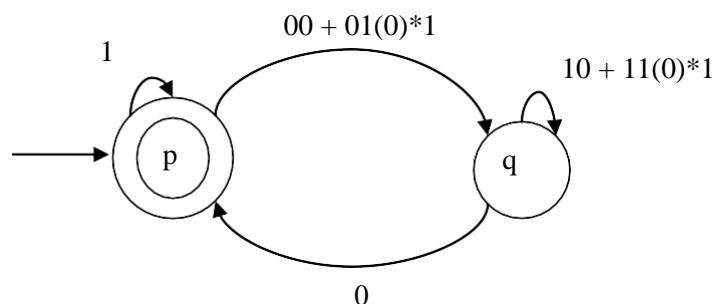
$$r_9 = 0 \quad \leftarrow \phi + a = a$$

Reduced Automaton

Step 2: Elimination of 'r'.

- $r_1 : p - r - q$
- $r_2 : q - r - p$
- $r_3 : p - r - p$
- $r_4 : q - r - q$

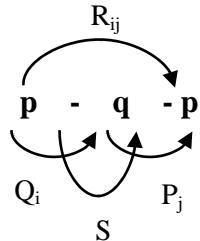
Elimination of r from p - r - q (r1)**Elimination of r from q - r - p (r2)****Elimination of r from p - r - p (r3)****Elimination of r from q - r - q (r4)**

The Reduced Automaton is given as



Step 3: Elimination of 'q'

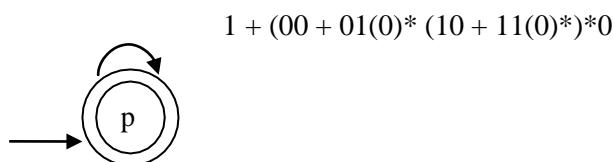
Elimination of q from $p - q - p$ (r_1)



$$\begin{aligned} R_{ij} &= 1 \\ Q_i &= 00 + 01(0)^*1 \\ S &= 10 + 11(0)^*1 \\ P_j &= 0 \end{aligned}$$

$$\begin{aligned} r_1 &= R_{ij} + Q_i S^* P_j \\ &= 1 + [00 + 01(0)^*1 (10 + 11(0)^*1)^*0] \end{aligned}$$

The Reduced Automaton is given as

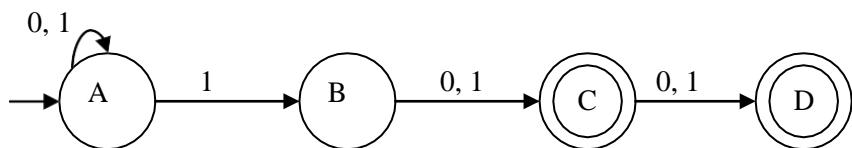


\therefore The RE = r^*

$$R = [1 + (00 + 010^*1) (10 + 110^*1)^*0]^*$$

- 3) Convert the following NFA that accepts all strings of 0's and 1's such that either the second or third position from right end has 1 to a R.E using state elimination technique.

AU NOV/DEC 2013



Solution:-

Eliminate B. A \rightarrow Start state \Rightarrow cannot be eliminated

C, D \rightarrow Final states \Rightarrow cannot be eliminated first

To find:

Elimination of state B

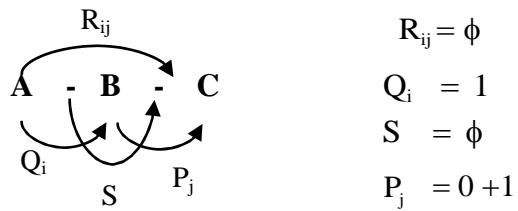
- $r_1 : A - B - C$
- $r_2 : C - B - A$
- $r_3 : A - B - D$
- $r_4 : D - B - A$
- $r_5 : A - B - A$
- $r_6 : C - B - C$

- $r_7 : D - B - D$
- $r_8 : C - B - D$
- $r_9 : D - B - C$

Step 1:

Elimination of state B can be done using $R_{ij} + Q_i S^* P_j$

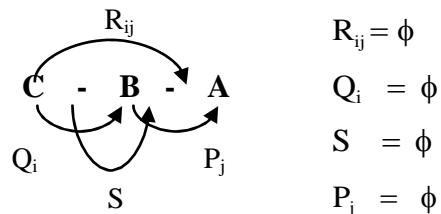
Eliminating B from $A - B - C$ (r1)



$$\begin{array}{lll} R_{ij} = \phi & Q_i = 1 & r_2 = \phi + \phi\phi^* \phi \\ Q_i = \phi & S = \phi & = \phi + \phi \\ P_j = 0 + 1 & P_j = \phi & r_2 = \phi \end{array}$$

$$\begin{aligned} r_2 &= \phi + \phi\phi^* \phi \\ &= \phi + \phi \\ r_2 &= \phi \end{aligned}$$

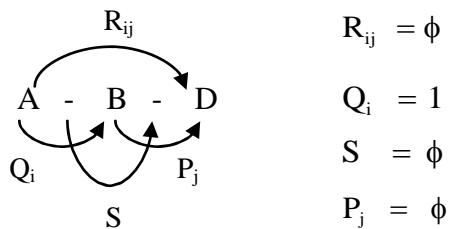
Eliminating B from $C - B - A$ (r2)



$$\begin{array}{lll} R_{ij} = \phi & Q_i = \phi & r_2 = \phi + \phi\phi^* \phi \\ Q_i = \phi & S = \phi & = \phi + \phi \\ P_j = \phi & P_j = \phi & r_2 = \phi \end{array}$$

$$\begin{aligned} r_2 &= \phi + \phi\phi^* \phi \\ &= \phi + \phi \\ r_2 &= \phi \end{aligned}$$

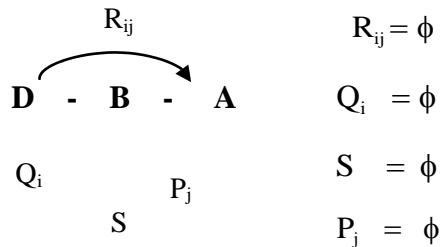
Eliminating B from $A - B - D$ (r3)



$$\begin{array}{lll} R_{ij} = \phi & Q_i = 1 & r_3 = \phi + 1\phi^* \phi \\ Q_i = 1 & S = \phi & = \phi + \phi \\ P_j = \phi & P_j = \phi & r_3 = \phi \end{array}$$

$$\begin{aligned} r_3 &= \phi + 1\phi^* \phi \\ &= \phi + \phi \\ r_3 &= \phi \end{aligned}$$

Eliminating B from $D - B - A$ (r4)



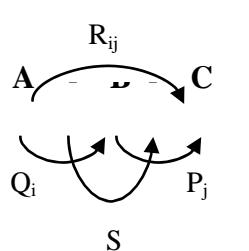
$$\begin{array}{lll} R_{ij} = \phi & Q_i = \phi & r_4 = \phi + \phi\phi^* \phi \\ Q_i = \phi & S = \phi & = \phi + \phi \\ P_j = \phi & P_j = \phi & r_4 = \phi \end{array}$$

$$\begin{aligned} r_4 &= \phi + \phi\phi^* \phi \\ &= \phi + \phi \\ r_4 &= \phi \end{aligned}$$





Eliminating B from A - B - A (r5)



$$R_{ij} = 0+1$$

$$Q_i = 1$$

$$S = \phi$$

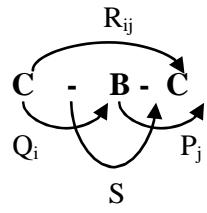
$$P_j = \phi$$

$$r_5 = (0+1) + 1\phi^*\phi$$

$$= (0+1) + \phi$$

$$r_5 = 0+1$$

Eliminating B from C - B - C (r6)



$$R_{ij} = \phi$$

$$Q_i = \phi$$

$$S = \phi$$

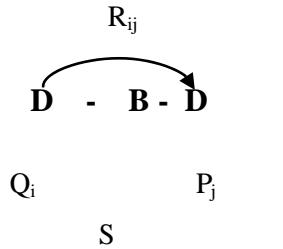
$$P_j = 0+1$$

$$r_6 = \phi + \phi \phi^*(0+1)$$

$$= \phi + \phi$$

$$r_6 = \phi$$

Eliminating B from D - B - D (r7)



$$R_{ij} = \phi$$

$$Q_i = \phi$$

$$S = \phi$$

$$P_j = \phi$$

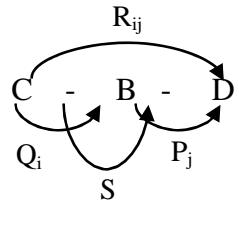
$$r_7 = \phi + \phi \phi^* \phi$$

$$= \phi + \phi$$

$$r_7 = \phi$$



Eliminating B from C - B - D (r8)



$$R_{ij} = 0+1$$

$$Q_i = \phi$$

$$S = \phi$$

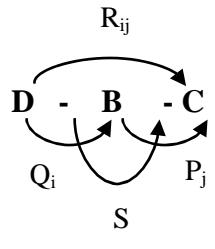
$$P_j = \phi$$

$$r_8 = (0+1) + \phi \phi^* \phi$$

$$= (0+1) + \phi$$

$$r_8 = 0+1$$

Eliminating B from D - B - C (r4)



$$R_{ij} = \phi$$

$$Q_i = \phi$$

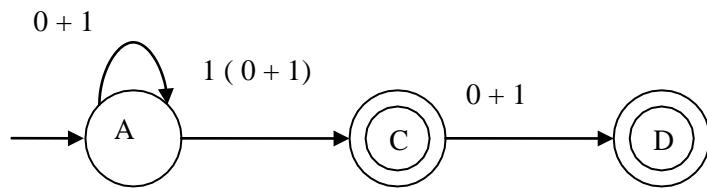
$$S = \phi$$

$$P_j = 0+1$$

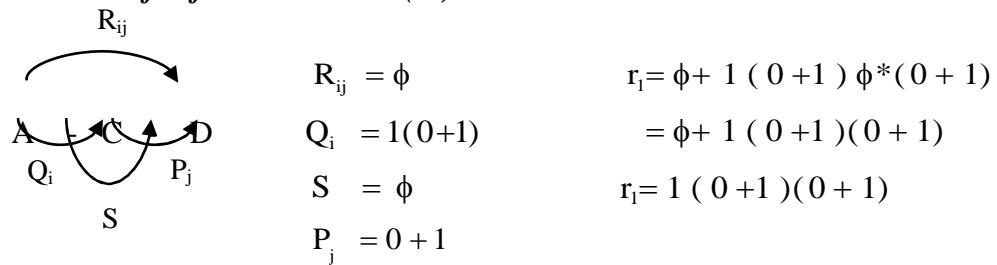
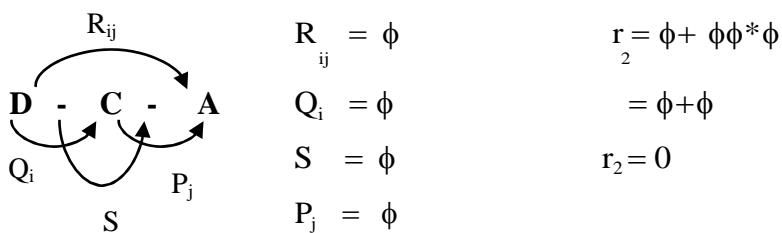
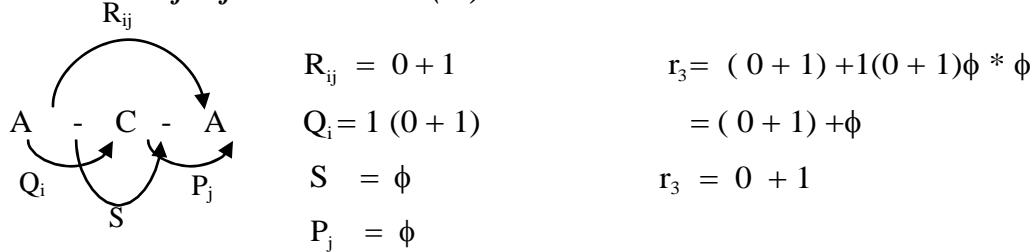
$$r_9 = \phi + \phi \phi^* \phi (0+1)$$

$$= \phi + \phi$$

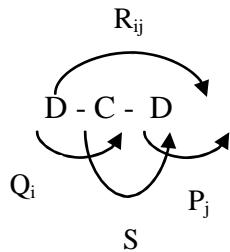
$$r_9 = \phi$$

Reduced Automaton**Step 2:****Case – 1: Eliminating of ‘C’**

- $r_1 : A - C - D$
- $r_2 : D - C - A$
- $r_3 : A - C - A$
- $r_4 : D - C - D$

Elimination of C from $A - C - D$ ($r1$)**Elimination of C from $D - C - A$ ($r2$)****Elimination of C from $A - C - A$ ($r3$)**

Eliminating C from D - C - D:- (r4)



$$R_{ij} = \phi$$

$$Q_i = \phi$$

$$S = \phi$$

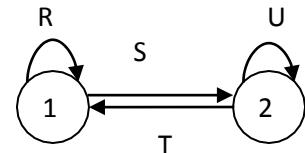
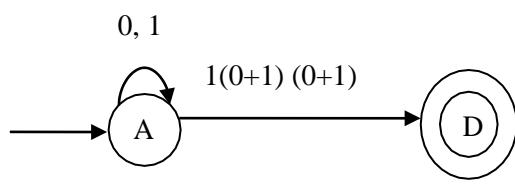
$$P_j = \phi$$

$$r_4 = \phi + \phi^* \phi (0+1)$$

$$= \phi + \phi$$

$$r_4 = \phi$$

Resulting Automaton



Regular Expression (RE1) = $(R + SU^*T)^*$ SU^*

$$\begin{aligned} &= ((0+1) + (1(0+1)(0+1))(\emptyset)^*\emptyset^*)^* \\ &= (1(0+1)(0+1)\emptyset^*)^* \\ &= ((0+1) + \emptyset^*)^* (1(0+1)(0+1)) \\ &= (0+1)^* (1(0+1)(0+1)) \end{aligned}$$

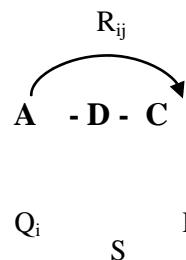
Case – 2: Elimination of D from Automation obtained from step – 1

A - C - D

D - C - A

A - C - A

Eliminating D from A- D - C (r₁)



$$R_{ij} = 1(0+1)$$

$$Q_i = \phi$$

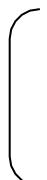
$$S = \phi$$

$$P_j = \phi$$

$$r_1 = 1(0+1) + \phi^* \phi \phi$$

$$= 1(0+1) + \phi$$

$$r_1 = 1(0+1)$$



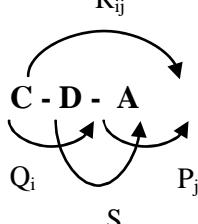
Eliminating D from C - D - A (r2)

$$\begin{array}{ccc}
 \begin{array}{c} R_{ij} \\ \curvearrowright \\ A - D - C \end{array} &
 \begin{array}{l} R_{ij} = \phi \\ Q_i = 0 + 1 \\ S = \phi \\ P_j = \phi \end{array} &
 \begin{array}{l} r_2 = \phi + (0 + 1) \phi^* \phi \\ = \phi + \phi \\ r_2 = \phi \end{array} \\
 Q_i & S & P_j
 \end{array}$$

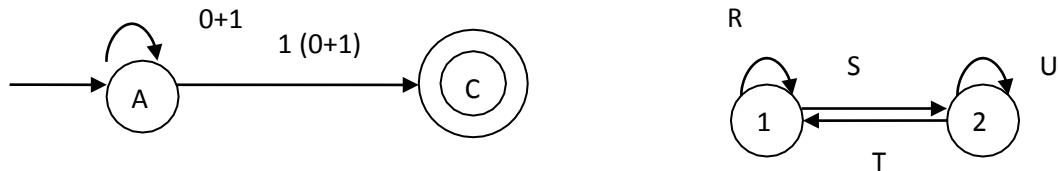

Eliminating D from A - D - A (r3)

$$\begin{array}{ccc}
 \begin{array}{c} R_{ij} \\ \curvearrowright \\ A - D - A \end{array} &
 \begin{array}{l} R_{ij} = 0 + 1 \\ Q_i = \phi \\ S = \phi \\ P_j = \phi \end{array} &
 \begin{array}{l} r_3 = (0 + 1) \phi^* \phi \phi \\ = (0 + 1) + \phi \\ r_3 = 0 + 1 \end{array} \\
 Q_i & S & P_j
 \end{array}$$


Eliminating D from C - D - A (r4)

$$\begin{array}{ccc}
 \begin{array}{c} R_{ij} \\ \curvearrowright \\ C - D - A \end{array} &
 \begin{array}{l} R_{ij} = \phi \\ Q_i = 0 + 1 \\ S = \phi \\ P_j = \phi \end{array} &
 \begin{array}{l} r_4 = \phi + (0 + 1) \phi^* \phi \\ = \phi + \phi \\ r_4 = \phi \end{array} \\
 Q_i & S & P_j
 \end{array}$$


The Reduced Automaton



Regular Expression (RE1) = $(R + Su^*T)^* SU^*$

$$RE1 = (0 + 1) + (1(0 + 1)) \phi^* \phi)^* (1(0 + 1)) \phi^*$$

$$RE2 = (0 + 1)^* 1(0 + 1)$$

Final Automaton

$$\begin{aligned}
 R &= RE_1 + RE_2 \\
 &= (0 + 1)^* 1(0 + 1)(0 + 1) + (0 + 1)^* 1(0 + 1)
 \end{aligned}$$

CONVERTING R.E TO AUTOMATA

THEOREM

Every language defined by a regular expression is also defined by definite automata

(or)

Let „r“ be a regular expression, then there exists a NFA with ϵ – transition that accepts $L(r)$.

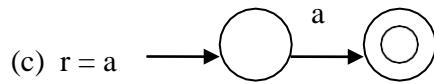
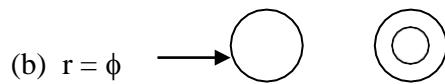
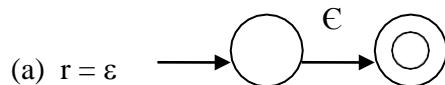
Proof

Suppose $L = L[R]$ for a regular expression, R . we show that $L = L[E]$ for some ϵ – NFA, with

- (i) Exactly one accepting state.
- (ii) No arcs into the initial state.
- (iii) No arcs out of the accepting state.

Structural Induction on R

Basis



In (a) → the RE, ϵ is handled. The language = $\{\epsilon\}$, since the only part from the start state to the accepting state, labeled ϵ .

In (b) → It shows the construction of ϕ clearly there are no paths from start state $L = \{\phi\}$

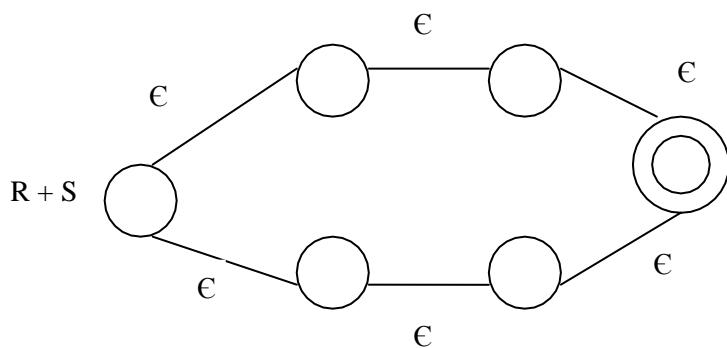
In (c) → It gives the automaton for a RE , „a“. The language of their automaton evidently consists of the one string, „a“ which is also $L[a]$. These automaton satisfy conditions (i), (ii), (iii) of the induction hypothesis.

Inductive Step

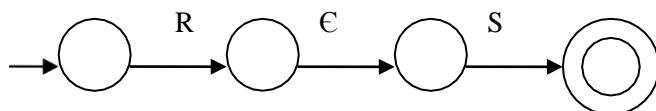
Assume that the theorem is true for the immediate sub-expressions of a given regular expression. The languages of ϵ -NFA's with a single accepting state.

- The expression is $R + S$ for some smaller expression R and S .
- It starts at a new state, can go to the start state of either R or S , and then reach the accepting state of one of these automata following a path labeled by some string $L[R]$ or $L[S]$ respectively.
- Once we reach the accepting state of the automaton for R or S .

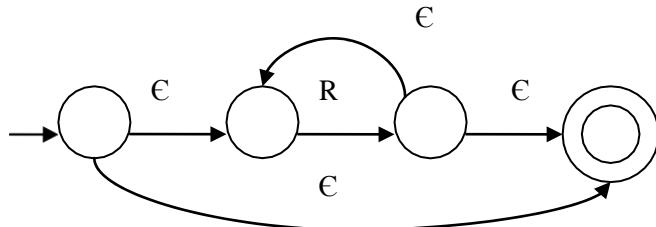
(a)



- We can follow one of the ϵ – arcs to the accepting state of the new automaton.
- Thus the language of this automaton is $L [R] \cup L [S]$.

(b) $R.S$ 

- The expression is $R.S$ for some smaller expression R and S .
- The start state of the first automaton becomes the start state of the whole and the accepting state is the accepting state of the whole.
- The idea is that the only paths from start state to accepting state, go first through the automaton for R where it must follow a path labeled by a string in $L[R]$, and then prove the automaton for S where it follows a path labeled by a string in $L [S]$.
- Thus the language is the concatenation of two regular expressions, $L[R] L[S]$.

(c) R^* 

- The expression is R^* for some smaller expression, R .
- This automaton allows us to go either:
 - Directly from the start state to the accepting state along a path labeled ϵ . That path ϵ which is in $L[R^*]$ doesn't matter what expression R is.
 - Goes to the start state of the automaton for R and processes through that automaton one or more times and then to the accepting state. This set of paths allows us to accept strings in ϵ , $L[R]$, $L[R]L[R]$, $L[R]L[R]L[R]$ and so on.
- Thus covering all strings in $L[R^*]$ except perhaps ϵ which was covered by the direct arc to the accepting state mentioned in (c)

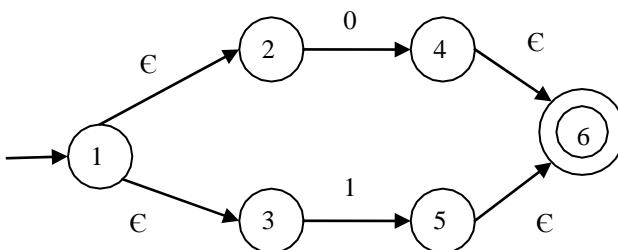
(d) *The expression is (r) for some smaller expression, R .*

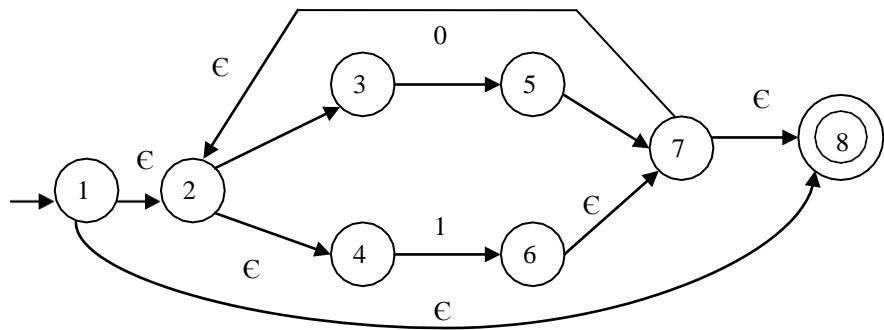
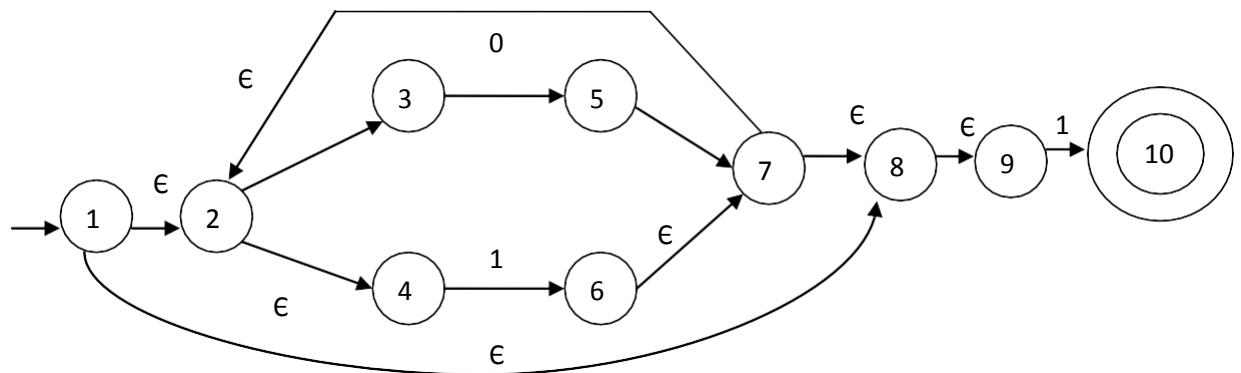
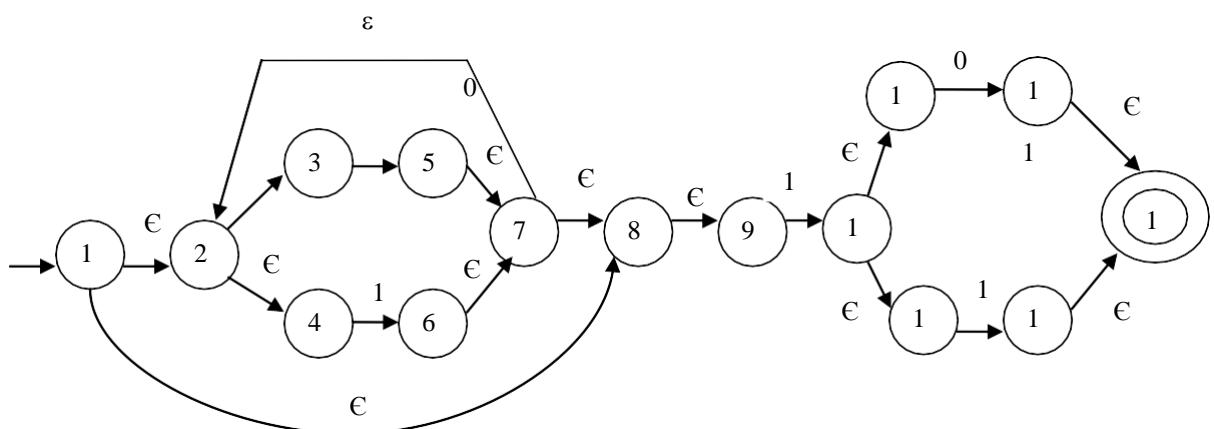
- The automaton for R also serves as the automaton for (r) .
- Since the parenthesis does not change the language defined by the expression.
- It is a simple observation that the constructed automaton satisfies the three conditions given the inductive hypothesis, one accepting state with no arcs into the initial state or out of the accepting state.

PROBLEMS

1) Convert the RE: $(0 + 1)^* 1(0+1)$ to ϵ -NFA

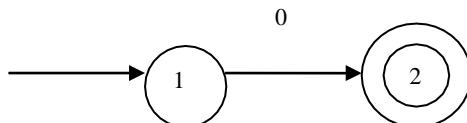
$(0+1)$



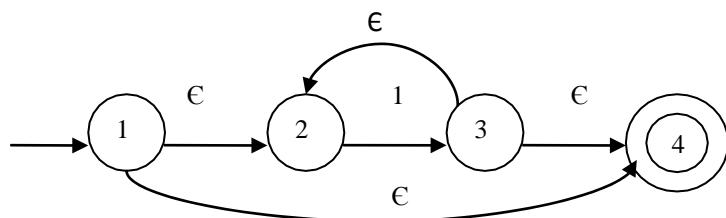
$(0+1)^*$  $(0+1)^*I$  $(0+1)^* I (0+1)$ 

2) Convert the RE: 01^* to ϵ -NFA

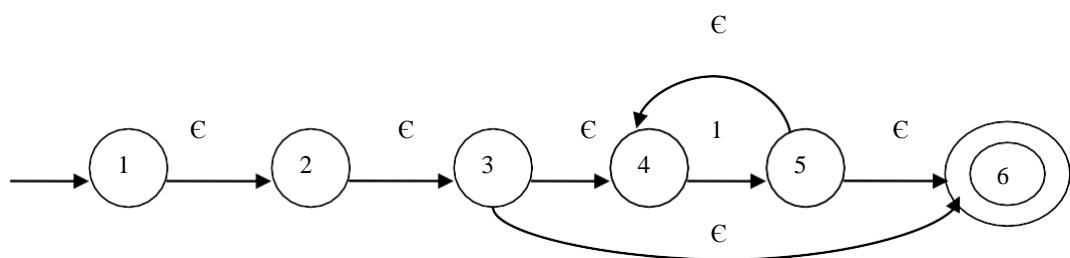
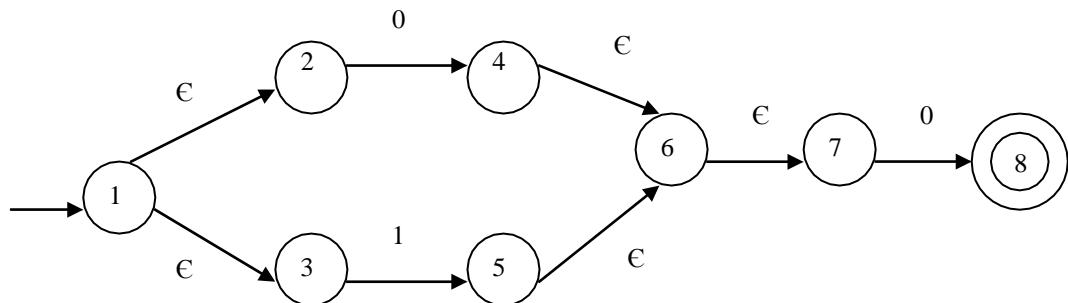
0



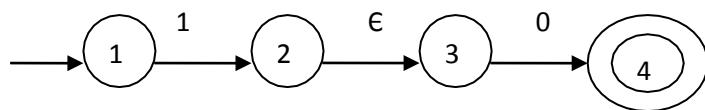
I^*

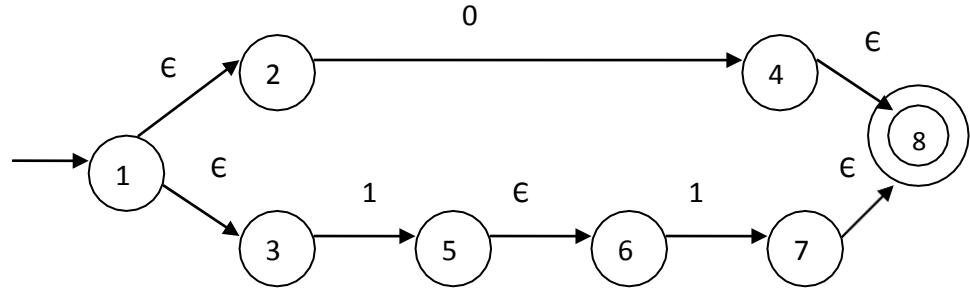


$0I^*$

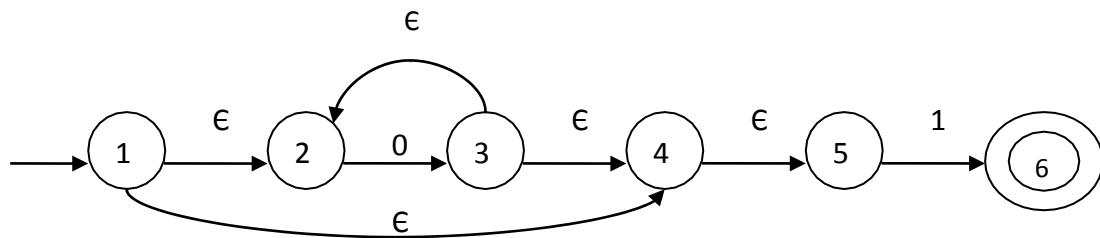

3) Convert the RE:- $(0+1)0$ to ϵ -NFA

4) Convert the RE: $10 + (0 + 11) 0^*1$

10

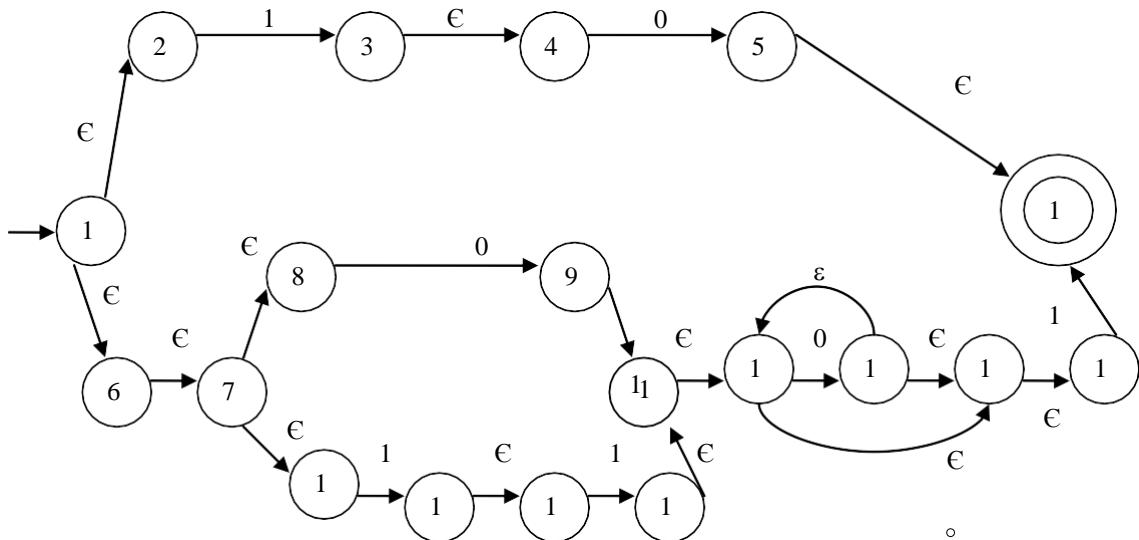




0^*1

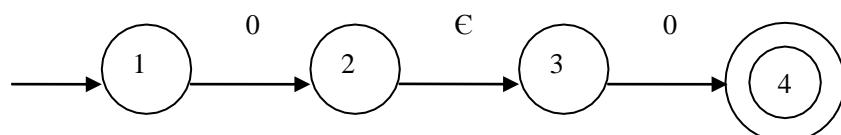


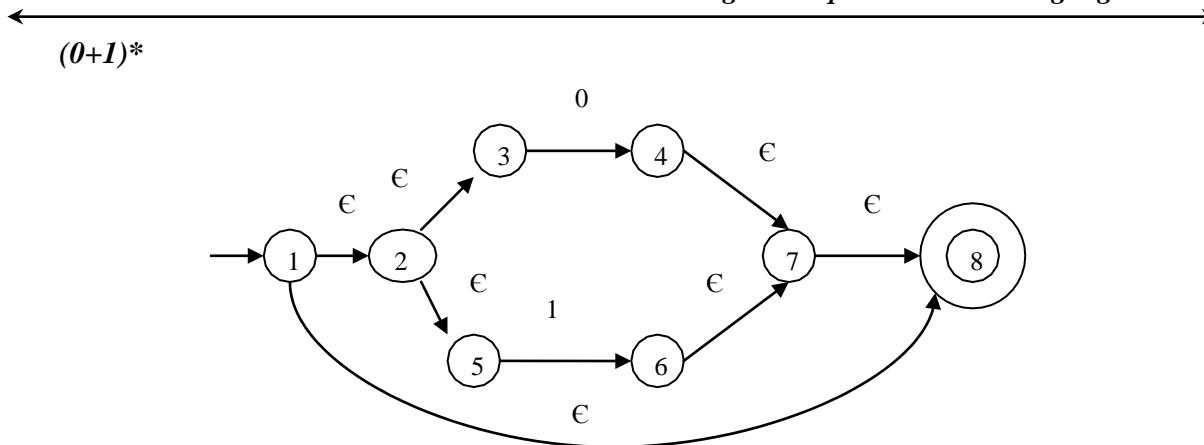
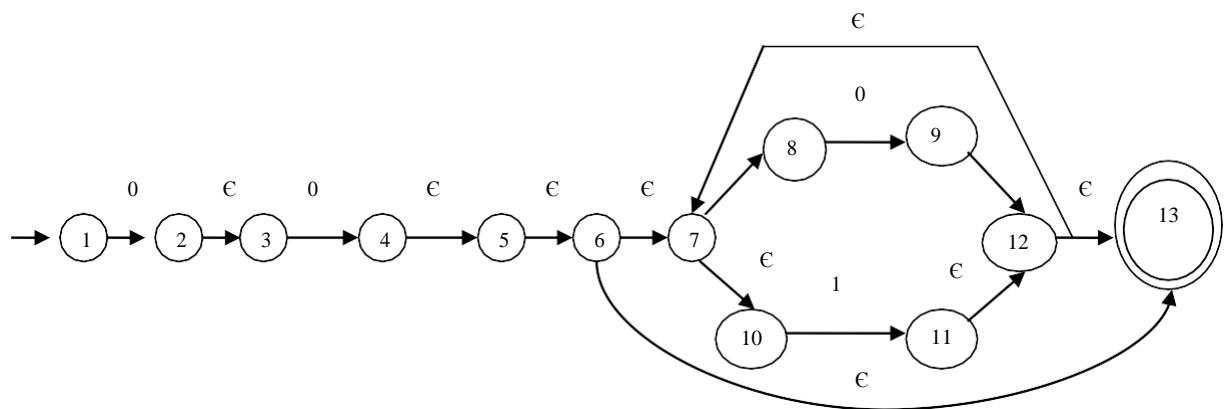
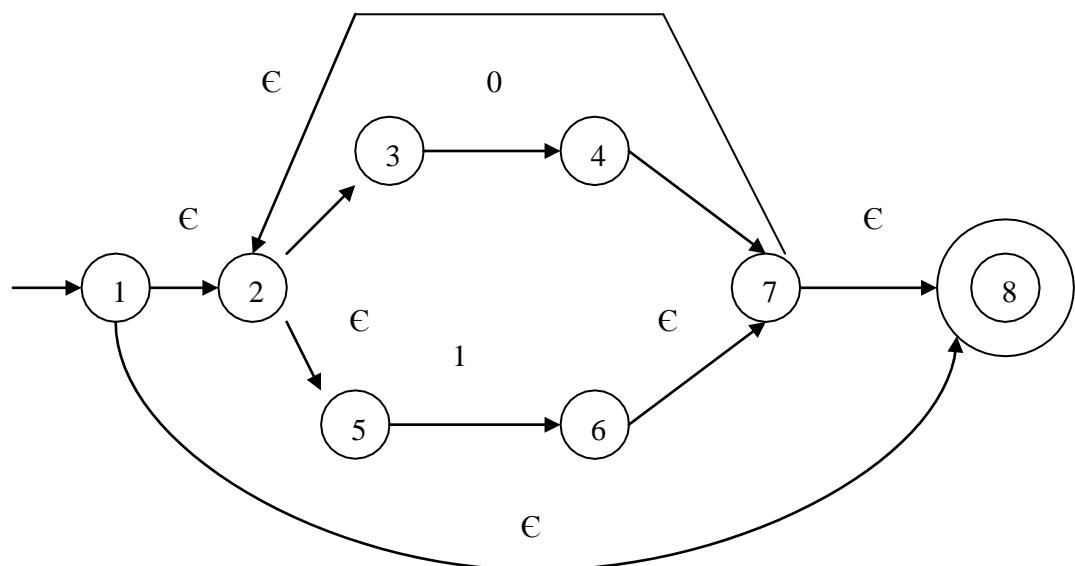
$10 + (0 + 11)0^*1$



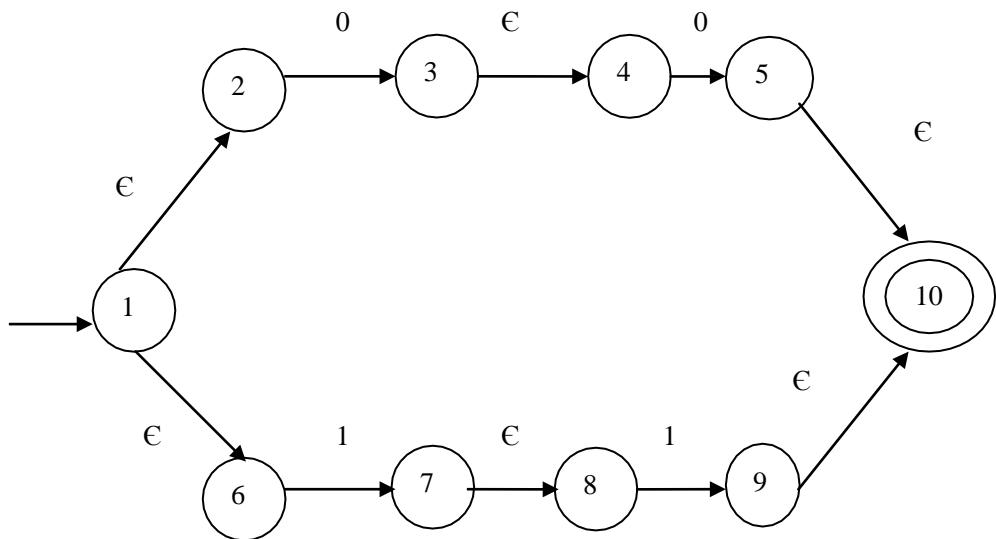
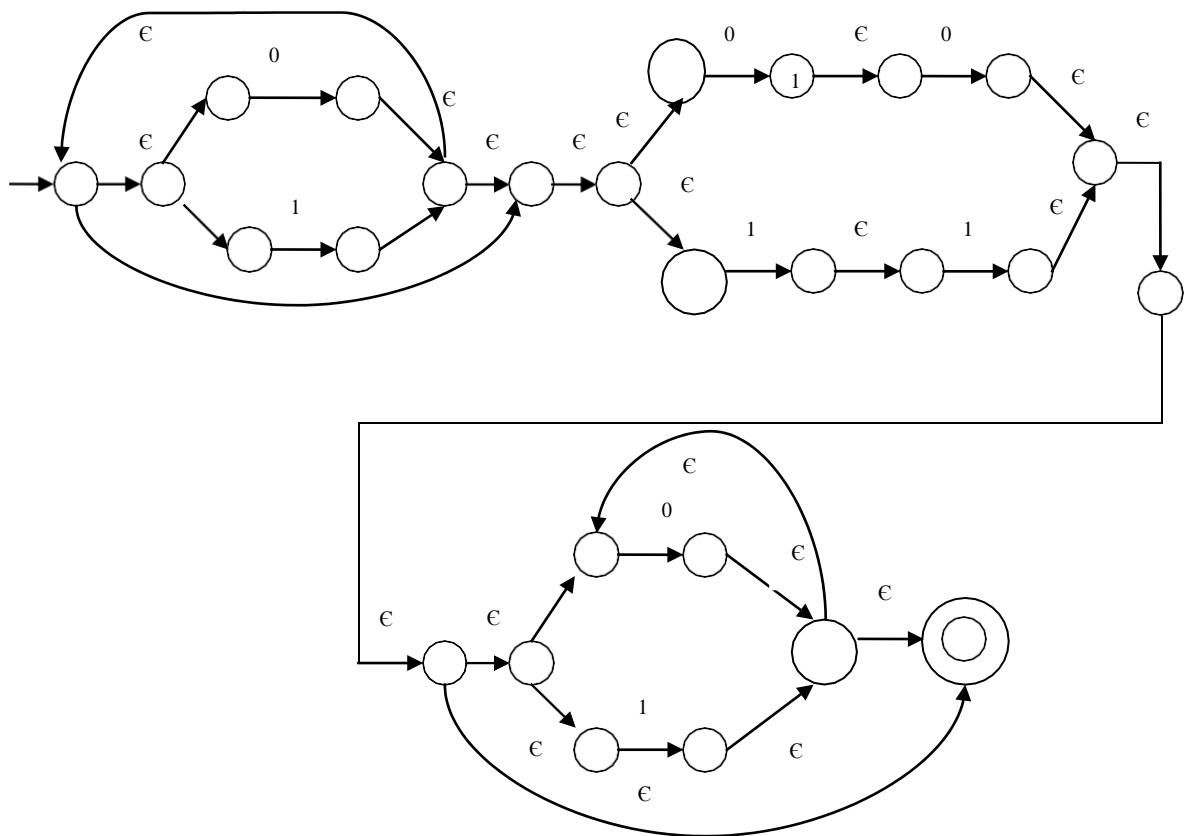
5) Convert the RE: $00(0+1)^*$ to ϵ -NFA

00



 $00(0+1)^*$ **6) Convert the RE:- $(0+1)^* (00+11) (0+1)^*$ to ϵ -NFA** $(0+1)^*$ 

00+11:-

 $(0+1)^*(00+11)(0+1)^*$ 

PROVING LANGUAGES NOT TO BE REGULAR

The following theorem, known as “**Pumping Lemma for Regular Languages**” is used as a tool to prove that certain languages are not regular. The principle behind this lemma is “**Pigeon Hole principle**.”

PIGEON – HOLE PRINCIPLE

If „n“ objects are put into „m“ containers, where $n > m$, then at least one container must hold more than one object.

Theorem: // Pumping Lemma

AU NOV/DEC 2013

Let L be a regular language, then there exist a constant n (which depends on L) such that for every string w in L such that $|w| \geq n$, we can break „ w “ into 3 strings:

$$w = xyz$$

such that

$$(i) \quad y \neq \epsilon$$

$$(ii) \quad |xy| \leq n$$

$$(iii) \quad \text{For all } k \geq 0, \text{ the string } xy^kz \text{ is also in } L.$$

i.e., we can always find a non – empty string y not too far from the beginning of w that can be pumped (repeating y any number of times or deleting it keeps the resulting string in the language, L .

PROOF

Suppose L is regular, then $L = L[A]$ for some DFA, A . Suppose A has n states.

Consider any string, w of length „ n “ or more, say $w = a_1, a_2, \dots, a_m$ where $m \geq n$ and a_i is an input symbol. For $i = 0, 1, 2, \dots, n$ define state $P_i = \hat{\delta}(q_0, (a_1, a_2, \dots, a_n))$ where δ is the transition function of A and q_0 is the start state of A . i.e, P_i is the state A after reading the first i symbols of w .

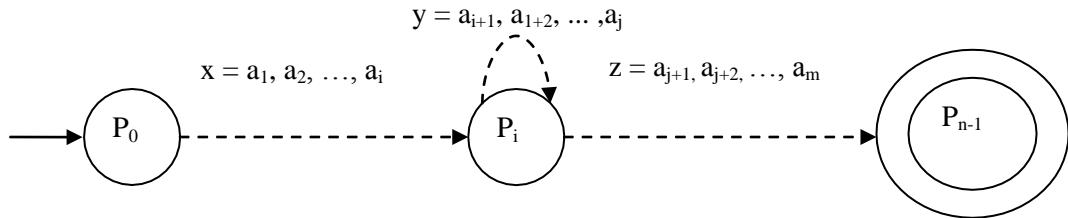
Assume that, $q_0 = P_0 = \delta(q_0, \epsilon)$. Since we have only n states, it is easy to see that every string consisting „ n “ or more symbols must cause at least a state to repeat. (This principle is called as Pigeon hole principle).

Therefore, we can find two different integers i and j such that $0 \leq i < j \leq n$ with $P_i = P_j$. By assuming the loop occurs are P_i , we can break the string w as,

$$w = xyz \text{ such that}$$

$$(i) \quad x = a_1, a_2, \dots, a_i$$

- (ii) $y = a_{i+1}, a_{i+2}, \dots, a_j$
 (iii) $z = a_{j+1}, a_{j+2}, \dots, a_m$



That is, x takes us from P_0 to P_i once, y takes us from P_i back to P_i (since P_i is also P_j) and z takes us from P_i to P_{n-1} . Note that x may be empty in the case that $i = 0$, also z may be empty if $j = n = m$. However y cannot be empty since i is strictly less than j . Now consider what happens if the automaton A receives the input xy^kz for any $k \geq 0$.

Case 1:

If $k = 0$, then the automaton from the start state q_0 (which is also P_0) to P_i on input x . Since P_i is also P_j it must be that A goes from P_i to the accepting state on input z . Thus accepts xz .

Case 2:

If $k > 0$, then A goes from q_0 to P_i on input x circles from P_i to P_i , $k - 1$ times on input y^k and then goes to the accepting state on input, z . Thus for any $k \geq 0$, xy^kz is also accepted by A that xy^kz is in L . Therefore we can say that L is a regular language.

PROBLEMS

- 1) By using pumping lemma, P.T the language, $L = \{o^n \mid o^n / n \geq 1\}$ is not a regular language.

Solution:

Assume that L is a regular language. Let n be a constant and $w = o^n \phi^n$ such that $|w| \geq n$. By pumping lemma, w can be splitted into 3 parts such that $w = xy$ with

$$(i) y \neq \epsilon [\text{i.e., } |y| \geq 1]$$

$$(ii) |xy| \leq n$$

$$(iii) xy^k z \in L \quad \forall k \geq 0$$

Since $y \neq \epsilon$

$$x = o^{n-1} \quad // \text{since } |y| \leq n$$

$$y = 0 \quad // \text{ Since } |y| \geq 1$$

$$z = 10^n$$

Thus

$$(i) \quad y \neq \epsilon$$

$$(ii) \quad |xy| = (n - 1) + 1 \Rightarrow |xy| \leq n$$

$$(iii) \text{ When } k = 0; xy^k z = xy^0 z = xz = 0^{n-1} 10^n \in L.$$

The number of zeroes should be equal. Here it is a contradiction.

$\therefore L$ is not a regular language

2) Show that the language, $L = \{a^i b^j c^k / 0 \leq i < j < k\}$ is not regular.

Assume that L is regular and let n be any integer. Consider the string, $w = a^n b^{n+1} c^{n+2}$. Then $w \not\models n$.

By pumping lemma, w can be splitted as $w = xyz$ such that

$$(1) \quad y \neq \emptyset, y \neq \epsilon$$

$$(2) \quad |xy| \leq n$$

$$(3) \quad xy^k z \in L \quad \text{for } k \geq 0$$

Since $|xy| \leq n$

$$x = a^{n-1} \Rightarrow xy = a^{n-1} \cdot a^1 = a^n \Rightarrow |y| \leq n$$

$$y = a \quad \Rightarrow |y| = 1$$

$$z = b^{n+1} c^{n+2}$$

At $k = 0$:

$$\begin{aligned} xy^0 z &= xz \\ &= a^{n-1} \cdot b^{n+1} \cdot c^{n+2} = a^{n-1} \cdot b^{n+1} \cdot c^{n+2} \end{aligned}$$

At $k = 1$:

$$\begin{aligned} xy^1 z &= a^{n-1} \cdot a \cdot b^{n+1} \cdot c^{n+2} \\ &= a^n \cdot b^{n+1} \cdot c^{n+2} \end{aligned}$$

At $k = 2$:

$$xy^2 z = a^{n-1} \cdot a \cdot a \cdot b^{n+1} \cdot c^{n+2}$$

$$= a^{n+1} \cdot b^{n+1} \cdot c^{n+2}$$

Comparing the powers of a, b and c, the language L is not regular.

3) Show that $L = \{a^p \mid p \text{ is a prime number}\}$ is not regular. AU – DEC 2007, NOV 2011

Assume that L is regular and n be any integer. Let $P \geq n$ is a prime number and $w = a^p \in L$. By pumping lemma, w can be written as $w = xyz$ such that

$$(i) |y| \geq 1 \Rightarrow y \neq \epsilon$$

$$(ii) |xy| \leq n$$

$$(iii) xy^k z \in L \quad \forall k \geq 0$$

Let,

$$Y = am \text{ where } m \geq 1 \Rightarrow |y|=m=n$$

$$\text{Consider } xy^k z = xyz \cdot y^{k-1}$$

Let $w = a^k ; k \rightarrow \text{first prime}$

$$xy^k z = xyz \cdot y^{k-1}$$

$k+1 \rightarrow \text{Next prime} > P$

$$|xy^k z| = |xyz| + |y^{k-1}|$$

$$|xy^{k+1} z| = |xyz| + |y^k|$$

$$= P + (k-1)|y|$$

$$= k + k|y|$$

$$= P + (k-1)m$$

$= k(1 + |y|) \rightarrow \text{not prime}$

$$= P + Pm \leftarrow k = p+1 \Rightarrow P = k - 1$$

$$= P(1+m)$$

Both p and $(1+m)$ are greater than 1. The product of P and $(1 + m)$ cannot be prime

$$\Rightarrow xy^k z \notin L \text{ for } k = p+1$$

$\Rightarrow L$ is not regular

4) Show that $L = \{0^n 1^n \mid n \geq 1\}$ is not regular. AU MAY 2004, DEC 2004, DEC 2005

Assume that L is a regular language. Let n be a constant and $w = 0^n 1^n$ such that $|w| \geq n$. By pumping lemma, w can be splitted into 3 parts such that $w = xyz$ with

$$(i) y \neq \epsilon \quad (\text{i.e. } y \neq 1)$$

$$(ii) |xy| \leq n$$

$$(iii) xy^k z \in L \quad \forall k \geq 0$$

Since $|y| \leq n$ and $y \neq \epsilon$,

$$x = 0^{n-1}$$

$$y = 0$$

$$z = 1^n$$

By pumping lemma, $xy^kz \in L$ for $k \geq 0$

When $k = 0$;

$$xy^0z = xz$$

$$0^{n-1} \cdot 1^n$$

$\notin L$ [Comparing the power of 0 and 1 with w]

$\therefore L$ is not a regular language.

5) Show that $L = \{a^{i^2} / i \geq 1\}$ is not a regular

AU MAY 2006, DEC 2006, DEC 2007, DEC 2008

When, $i = 1; L = \{a^{1^2}\} = a^1$

$i = 2; L = \{a^{2^2}\} = aaaa = a^4$

$i = 3; L = \{a^{3^2}\} = a^9$

\therefore The no of a's is always a perfect square. Consider $L = a^{n^2}$ where length = n^2

$$\therefore \psi \neq n^2$$

By pumping lemma, $w = xyz$, where $|y| \geq 1$, $|xy| \leq n$, and $xy^kz \in L \quad \forall k \geq 0$

$$\text{At } k = 1 \Rightarrow xy^1z = xyz = a^{n^2} \rightarrow 1$$

$$|xyz| = |x| + |y| + |z| = n^2 \rightarrow 2$$

$$\text{At } k = 2 \Rightarrow |xy^2z| = |x| + |y|^2 + |z| = n^2 \rightarrow 3$$

From 2 and 3,

$$|x| + |y| + |z| > |x| + |y|^2 + |z|$$

$$\Rightarrow |xy^2z| > n^2 \rightarrow 4$$

Since $|xy| \leq n$;

$$4 \Rightarrow |xy^2z| \geq n^2$$

When $|y|=n$ and $x = \epsilon$

$$\begin{aligned} |xy^2z| &\leq n^2 + n \\ \Rightarrow |xy^2z| &< n^2 + n + n + 1 \quad [\text{For inequality removal}] \\ \Rightarrow |xy^2z| &< (n + 1)^2 \end{aligned}$$

From 4 and 5,

$$n^2 < |xy^2z| < (n + 1)^2$$

Example: $n = 2; n^2 = 2^2 = 4$

$$(n+1) = 2+1 \Rightarrow (n+1)^2 = 3^2 = 9$$

There is no perfect square between 4 and 9. Henceforth, $|xy^2z|$ is not a perfect square since there is no perfect square between n and $(n+1)$

$$\Rightarrow xyz \notin L \text{ since } xy^kz \notin L$$

$\therefore L$ is not regular.

6) Using Pumping lemma, prove that $L = \{o^{m,n}o^{m+n} \mid m \geq 1 \text{ and } n \geq 1\}$ is not regular.

AU DEC 2006, DEC 2013, MAY 2013

Let us assume that L is regular and L is accepted by a FA with „ n “ states.

Let $w = o^{m,n}o^{m+n}$

$$|w| = m + n + (m + n) = 2(m + n) \geq n$$

Let $w = xyz$ such that,

$$|y| > 0 \text{ and}$$

$$|xy| \leq n$$

$$\therefore x = o^{m-1}$$

$$y = 0$$

$$z = 1^n o^{m+n}$$

$$\text{At } k = 0; xy^kz \Rightarrow xyoz = o^{m-1}.1^n.o^{m+n} \rightarrow 1$$

From (1), the powers of O is not equal with that of L

Hence $xyz \notin L$

$\therefore L$ is not regular language.

7) Using pumping lemma, P.T:- $L = \{ww^R \mid w \in (0,1)^*\}$ is not regular. AU DEC 2006

Let us assume that L is regular and is accepted by a FA with n states.

Let $w = a^n b$

$$w^R = ba^n$$

$$L = ww^R = \{a^n bba^n\}$$

Consider $w = xyz$ such that

$$(i) |y| > 0$$

$$(ii) |xy| \leq n$$

$$(iii) xy^k z \in L \quad \forall k \geq 0$$

Let w be splitted as,

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

$$y = a$$

$$z = b.b.a^n$$

At $k = 0$:

$$xy^k z \Rightarrow xy^0 z \Rightarrow xz = a^{n-1}.b.b.a^n$$

Since the powers of a does not match, L is not regular.

8) Using Pumping lemma, P.T:- $\{w.w \mid w \in (0,1)^*\}$ is not regular.

Let us assume that L is a regular and is accepted by a FA with „ n “ states.

Let $w = a^n b$

$$\therefore L = \{ww\} = \{a^n b a^n b\}$$

Let $w = xyz$ such that

$$(i) |y| > 0$$

$$(ii) |xy| \leq n$$

$$(iii) xy^k z \in L \quad \forall k \geq 0$$

$\therefore w$ can be splitted as

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

$$y = a$$

$$z = b.a^n b.$$

At $k = 0$:

$$xy^k z \Rightarrow xy^0 z \Rightarrow xz = a^{n-1} \cdot b \cdot a^n b$$

Since the powers of a does not match with w, L is not regular.

CLOSURE PROPERTIES OF REGULAR LANGUAGES

AU MAY 2012, MAY 2013, NOV 2010, NOV 2011, NOV 2013, NOV 2012

If an operation on regular languages generates a regular language, then we say that the class of regular languages is closed under the above operation.

Some of the important closure properties of RL are as follows.

1. Union
2. Difference
3. Concatenation
4. Intersection
5. Complementation
6. Transpose / Reversal
7. Kleene star
8. Homomorphism
9. Inverse homomorphism

Closure under Union

Theorem

If L and M are regular languages, then LUM is also regular.

Proof

Let, $L = L[R]$

$$M = L[S]$$

$$LUM = L[R] \cup L[S]$$

$$= L[R+S]$$

Example

Let $m_1 = (S, \Sigma, \delta_1, S_o, F)$ and

\longleftrightarrow

$m_2 = (Q, \Sigma, \delta_2, q_0, G)$ be two given automata

Now $m_3 = L[m_1] \cup L[m_2]$

$$= (R, \Sigma, \delta_3, r_0, H),$$

Where $r_0 \rightarrow$ New start state having two ϵ – moves from $r_0 \rightarrow s_0$ and $r_0 \rightarrow q_0$ being added,

$$R = SUQ \cup \{r_0\}$$

$$H = FUG$$

$$\delta_3 = \delta_1 \cup \delta_2 \cup \{(r_0, \epsilon, s_0), (r_0, \epsilon, q_0)\}$$

Hence, $L[m_3] = L[m_1] \cup L[m_2]$

Thus, Regular Language is closed under Union.

Closure under Complementation

Theorem

If L is a regular language over the alphabet, Σ then $\bar{L} = \sum^* - L$ is also a regular language.

Proof

Let $L = L[A]$ for some DFA, where

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

Then, $\bar{L} = L[B]$, where

$$B = (Q, \Sigma, \delta, q_0, Q - F)$$

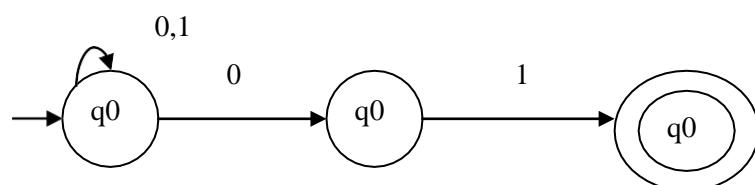
i.e., B is exactly like A , but the accepting states of A have not become the accepting states of B and vice-versa.

Then w is in $L[B]$, if and only if,

$\hat{\delta}(q_0, w)$ is in $\{Q - F\}$, which occur if and only if w is not in $L[A]$.

Example

Find the complement of $(0+1)^*01$



Here,

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

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

$$q_0 = \{q_0\}$$

$$F_N = \{q_2\}$$

$$\delta_N \rightarrow$$

Σ	0	1
Q_N		
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
$*q_2$	ϕ	ϕ

By subset construction technique,

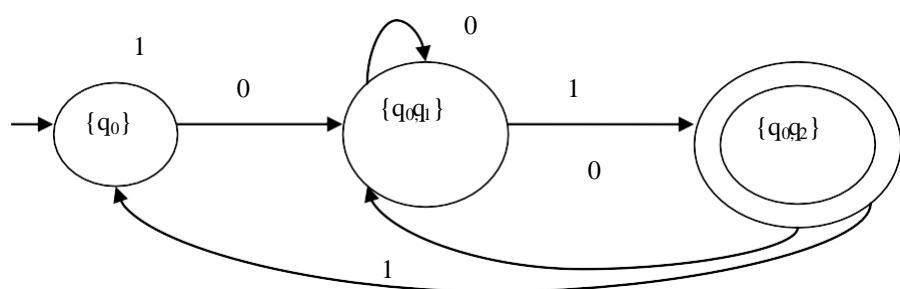
$$Q_D = \{\phi, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_1, q_2, q_3\}\}$$

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

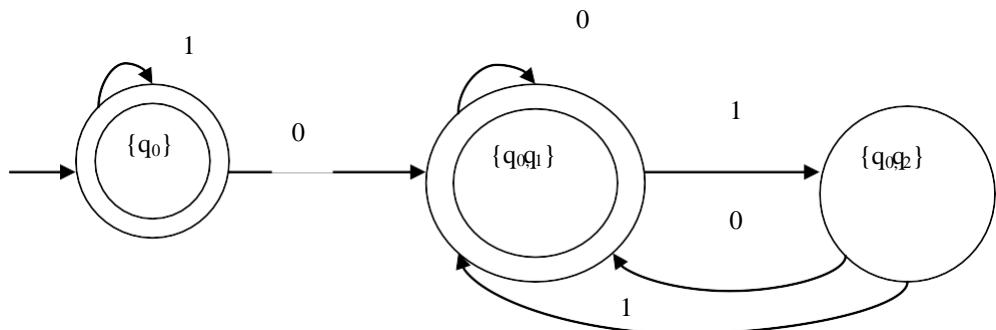
$$\delta_D$$

Σ	0	1
Q_N		
ϕ	ϕ	ϕ
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	ϕ	$\{q_2\}$
$* \{q_2\}$	ϕ	ϕ
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$* \{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$* \{q_1, q_2\}$	ϕ	$\{q_2\}$
$* \{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Reachable states from $\{q_0\}$ / Minimised δ_D is



Complement of DFA



Closure under Intersection

Theorem

If L and M are two regular languages, then $L \cap M$ is also regular language.

Proof

We define $A = (Q_L \times Q_M, \Sigma, \delta(q_L, q_M), q_L \times q_M, F_L \times F_M)$

where,

$$\delta((p, q), a) = \delta_L(p, a), \delta_M(q, a)$$

$$L[A] = L[A_L] \cap L[A_M]$$

By induction on $w \downarrow$

$$\hat{\delta}((q_L, q_M), w) = \hat{\delta}M(q_M, w))$$

But A accepts w if and only if $\hat{\delta}((q_L, q_M), w)$ is a pair of accepting states

(i.e.,) $\hat{\delta}(q_L, w)$ must be in F_L and

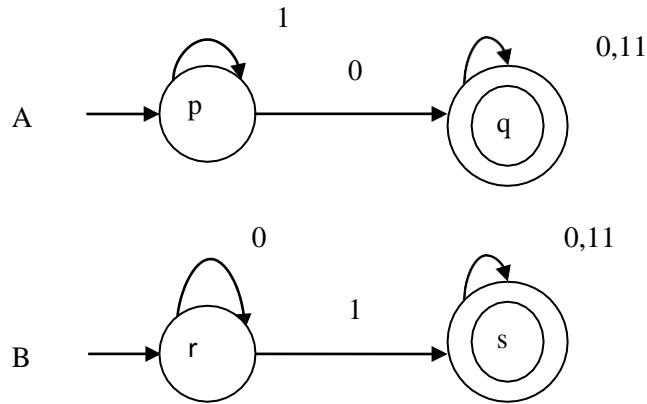
$\hat{\delta}(q_M, w)$ must be in F_M

Putting in another way, "w is accepted by A if and only if both AL and AM accepts w."

Thus A accepts the intersection of L and M.

Example

Construct $A \cap B$ where A and B is given by



$$A_A = (Q_A, \Sigma, \delta_A, q_a, F_A)$$

$$A_B = (Q_B, \Sigma, \delta_B, q_B, F_B)$$

$$A \cap B = (Q_A \times Q_B, \Sigma, \delta(q_A \times q_B), F_A \times F_B)$$

Where

$$\delta((p, q), a) = \delta_L(p, a), \delta_M(q, a))$$

Here,

$$Q_A \times Q_B = \{p, q\} \times \{r, s\}$$

$$= \{(p, r), (p, s), (q, r), (q, s)\}$$

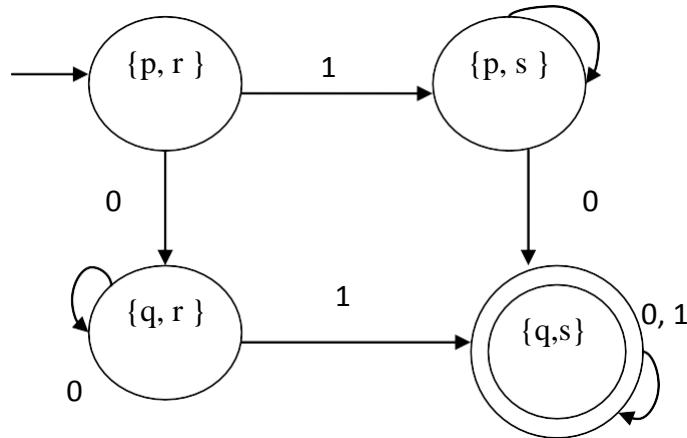
$$\Sigma = \{0, 1\}$$

$$q_A \times q_B = \{p, r\}$$

$$F_A \times F_B = \{q, s\}$$

$\delta(q_A, q_B)$	Σ	0	1
$Q \downarrow$			
$\rightarrow \{p, r\}$	$\{q, r\}$	$\{p, s\}$	
$\{p, s\}$	$\{q, s\}$	$\{p, s\}$	
$\{q, r\}$	$\{q, r\}$	$\{q, s\}$	
$* \{q, s\}$	$\{q, s\}$	$\{q, s\}$	

Finite Automata



Closure under Reversal

Theorem

If L is a regular language, then L^R is also regular.

Proof

Assume L is defined by regular expression. The proof is a structural induction on the size of E . We show that there is another regular expression.

ER such that $L(ER) = (L(E))^R$.

That is the language ER is the reversal of the language of E .

Basis

If $\epsilon = \epsilon / \phi / a$, then

$$ER = \epsilon / \phi / a$$

$$\therefore E = ER$$

Induction

There are 3 cases depending on the form of E .

- 1) $E = E_1 + E_2$ then $ER = E_1^R + E_2^R$

The justification is that the reversal of the union of two languages is obtained by computing the reversals of the two languages and taking the union of those languages.

- 2) $E = E_1 \cdot E_2$ then $ER = E_2^R \cdot E_1^R$

Example: $L(E_1) = \{01, 111\}$; $L(E_2) = \{00, 10\}$

Then, $L(E_1 \cdot E_2) = \{0100, 0110, 11100, 11110\}$

$$L(E_1 \cdot E)R = \{0010, 0110, 00111, 01111\} \quad \rightarrow 1$$

$$L(E_1)R = \{10, 111\}$$

$$L(E_2)R = \{00, 01\}$$

$$L(E_2)R \cdot L(E_1)R = \{0010, 0110, 00111, 01111\} \quad \rightarrow 2$$

From (1) and (2) $L(E_1E_2)R = L(E_2)R \cdot L(E_1)R$

3) E, E_1^* then $ER, (E_1^*)^*$

The justification is that any string w in $L(E)$ can be written as w_1, w_2, \dots, w_n where each w_i is in $L(E)$. But,

$$w^R = w_n^R \ w_{n-1}^R \ \dots \ w_1^R$$

Each w_i^R is in $L(E_i^R)$, so w^R is in $(E_1^*)^*$. Conversely any string in $L(E_1^*)^*$ is of the form $w_1^R, w_2^R, \dots, w_n^R$, where each w_i^R is the reversal of a string in $L(E_i)$. The reversal of the string, $w_n^R = w_n^R \ \dots \ w_1^R$ is therefore a string in $L(E^*)$ which is $L(E)$.

Thus a string is in $L(E)$ if and only if its reversal is in $L(E^*)^*$

Example: If $L = (0 + 1)^0*$

Then $LR = 0^* (0 + 1)$ [By applying closure under concatenation, and union]

Closure under Difference

Theorem

If L and M are regular languages then $L - M$ is also regular

$$L - M = L \cap \overline{M}$$

Proof

Since M is a regular language, \overline{M} is also a regular language by closure under complementation. Since L and \overline{M} are regular, the $L \cap \overline{M}$ is also regular by closure under intersection.

$\therefore L - M$ is a regular Language.

Closure under Concatenation

Let $M_1 = (S, \Sigma, \delta_1, s_o, F)$

and $M_2 = (Q, \Sigma, \delta_2, q_o, G)$ be two given automata.

Theorem

If two languages, M_1 and M_2 are regular, then $L(M_1) \cdot L(M_2)$ is also regular.

$$L(M_3) = L(M_1) \cdot L(M_2)$$

M_3 is given by, $M_3 = (R, \Sigma, \delta_3, s_o, G)$

$$\delta_3 = \delta_1 \cup \delta_2 \cup \{\epsilon - \text{move from every final state of } M_1 \text{ to start state of } M_2\}$$

Closure under kleene star

Theorem

Let $M_1 = (Q, \Sigma, \delta, q_0, F)$ be the given automata. Then M_2 can be constructed such that, $L(M_2) = L(M_1)^*$.

M_2 is constructed as:

(i) A new start state s_0 is added with an ϵ – move is from s_0 to q_0 .

(ii) A new final state f_0 is added with ϵ – move is added from s_0 to f_0 as ϵ is a member of $L(M_1)^*$.

$$M_2 = (q \cup \{s_0, f_0\}, \Sigma, \delta, s_0, \{f_0\})$$

Homomorphism

A string homomorphism is a function on strings that works by substituting a particular string for each symbol.

Example: Let $n(0) = ab$

$$h(1) = \epsilon$$

$$h(0011) = h(0)h(0)h(1)h(1)$$

$$= abab$$

Theorem

If L is a regular language over an alphabet, Σ then h is a homomorphism function on Σ then $h(L)$ is also regular.

Proof

To prove: $L[h(E)] = h[L(E)]$

Basis

If $E = \epsilon$ or ϕ , then

$$H(E) = \epsilon \text{ or } \phi$$

$$\therefore L[h(E)] = L(E)$$

If E is ϵ or ϕ , then $L(E)$ contains either no strings or a string with no symbols.

Thus,

$$H[l(E)] = l(E) = L[h(E)]$$

If $E = a$, then

$$L(E) = \{a\}$$

$$h[(E)] = \{\{h(a)\}\} = L[h(E)] \text{ for } E = \{a\}$$

Induction

There are 3 cases in induction part → Union, Concatenation and closure.

$$\text{Let } E = F + G$$

$$\begin{aligned} L[h(E)] &= L[h(F + G)] \\ &= L[h(F)] \cup h(G) \\ &= L[h(F)] \cup L[h(G)] \end{aligned} \quad \rightarrow 1$$

$$\begin{aligned} H[L(E)] &= h[L(F + G)] \\ &= h[L(F)] \cup L(G) \\ &= h[L(F)] \cup h[L(G)] \end{aligned} \quad \rightarrow 2$$

Therefore by induction, from (1) and (2)

$$L[h(f)] = h[L(f)] \text{ and}$$

$$L[h(G)] = h[L(G)]$$

Similarly concatenation and closure is also true,

$$\therefore L[h@] = h[L(R)] \Rightarrow \text{Hence proved}$$

Inverse homomorphism

$h^{-1}(L)$ is the set of strings, w in Σ^* such that $h(w)$ is in L .

Theorem

If h is a homomorphism from alphabet, Σ to alphabet T and L is regular language over T , then $h^{-1}(L)$ is also a regular language.

Proof

We construct a DFA, A for L , then a DFA for $h^{-1}(L)$ by using A and h . This DFA uses the states of A but translates the input symbol according to h before deciding on the next state. Let L be $L(A)$ where DFA, $A \mid (Q, \Sigma, \delta, q_0, F)$ define a DFA, $B = (Q, \Sigma, \gamma, q_0, F)$ where the transition function γ is constructed by,

$\gamma(q, a) = \hat{\delta}(q, h(a))$. It is a easy induction on $|w|$ to show that

$$\hat{\delta}(q, h(w)) = \hat{\delta}(q, w)$$

Since the accepting states of A and B are same, B accepts w if and only if A accepts $h(w)$ i.e., B accepts exactly those strings w that are in $h^{-1}(L)$

Example: Suppose h is a homomorphism from the alphabet $\{0, 1, 2\}$ to the alphabet $\{a, b\}$ defined by $h(0) = a$, $h(1) = ab$, $h(2) = ba$.

$$(i) h(0120) = h(0)h(1)h(2)h(0)$$

$$= aabbba$$

$$(ii) h(21120) = h(2)h(1)h(1)h(2)h(0)$$

$$= baababbaa$$

$$(iii) L[01^*2]$$

$$H(L) = h(01^*2)$$

$$= L[h(01^*2)]$$

$$= L[h(0)h(1^*)h(2)]$$

$$= L[a(ab)^*ba]$$

$$(v) h^{-1}(L) \Rightarrow h^{-1}(ababa) \text{ where } L = ababa$$

$$= h^{-1}(a)h^{-1}(ba)h^{-1}(ba)$$

$$= 022.$$

2.7 EQUIVALENCE AND MINIMISATION OF AUTOMATA

Two finite automata m_1 and m_2 are said to be equivalent if they accept the same language.

$$\text{i.e., } L(m_1) = L(m_2)$$

Two FA m_1 and m_2 are not equivalent over Σ if there exists a string w such that,

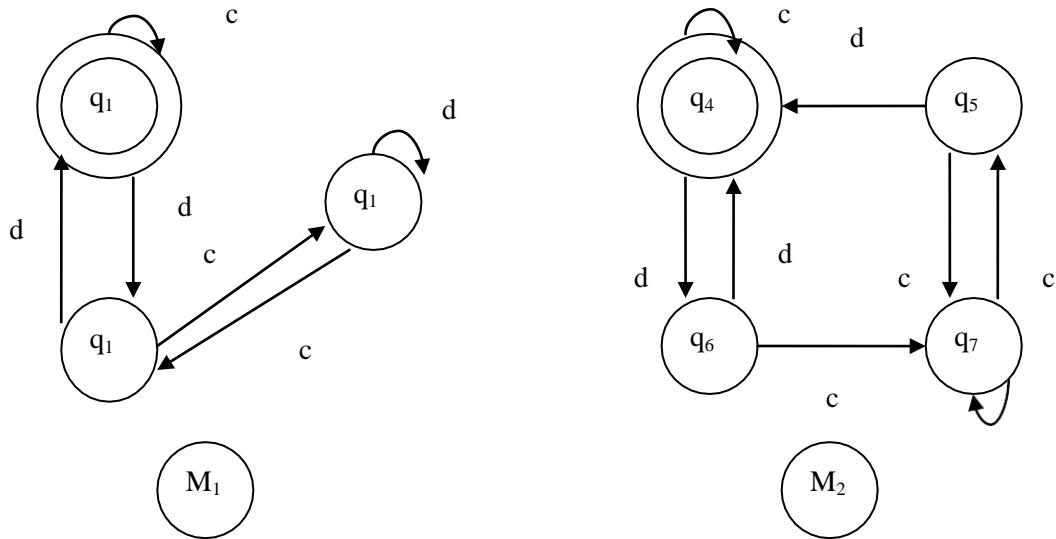
$$w \in L(m_1) \text{ and } w \notin L(m_2)$$

(or)

$$w \notin L(m_1) \text{ and } w \in L(m_2)$$

EXAMPLE:

- 1) Show that the automata given below are equivalent or not.**

**Step 1:**

Here the start states are $\langle q_1 \rangle$ and $\langle q_4 \rangle$

\therefore We shall start with $\langle q_1, q_4 \rangle$ over $\Sigma = \{c, d\}$

$$\delta_3(\langle q_1, q_4 \rangle, c) \Rightarrow \delta_1(q_1, c), \delta_2(q_4, c)$$

$$\Rightarrow \langle q_1, q_4 \rangle$$

$$\delta_3(\langle q_1, q_4 \rangle, d) \Rightarrow \delta_1(q_1, d), \delta_2(q_4, d)$$

$$\Rightarrow \langle q_2, q_6 \rangle$$

Step 2:

State $\langle q_2, q_6 \rangle$ over $\Sigma = \{c, d\}$

$$\delta_3(\langle q_2, q_6 \rangle, c) \Rightarrow \delta_1(q_2, c), \delta_2(q_6, c)$$

$$\Rightarrow \langle q_3, q_7 \rangle$$

$$\delta_3(\langle q_2, q_6 \rangle, d) \Rightarrow \delta_1(q_2, d), \delta_2(q_6, d)$$

$$\Rightarrow \langle q_1, q_4 \rangle$$

Step 3:

State $\langle q_3, q_7 \rangle$ over $\Sigma = \{c, d\}$

$$\delta_3(\langle q_3, q_7 \rangle, c) \Rightarrow \delta_1(q_3, c), \delta_2(q_7, c)$$

$$\Rightarrow \langle q_2, q_5 \rangle$$

$$\delta_3(\langle q_3, q_7 \rangle, d) \Rightarrow \delta_1(q_3, d), \delta_2(q_7, d)$$

$$\Rightarrow \langle q_3, q_7 \rangle$$

Step 4:

State $\langle q_2, q_5 \rangle$ over $\Sigma = \{c, d\}$

$$\delta_3(\langle q_2, q_5 \rangle, c) \Rightarrow \delta_1(q_2, c), \delta_2(q_5, c)$$

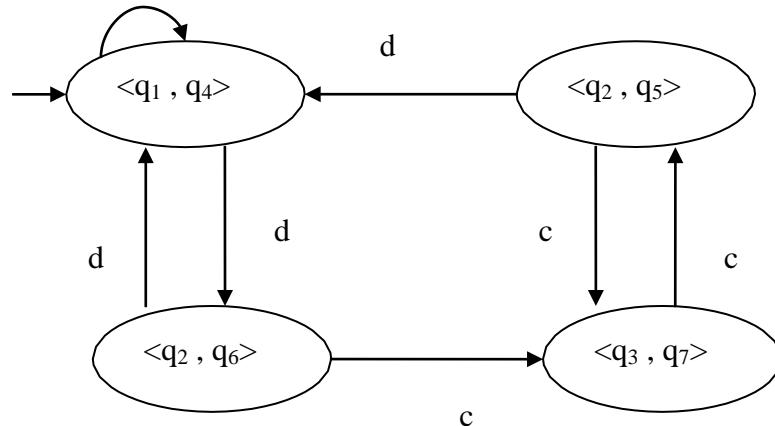
$$\Rightarrow \langle q_3, q_7 \rangle$$

$$\delta_3(\langle q_2, q_5 \rangle, d) \Rightarrow \delta_1(q_2, d), \delta_2(q_5, d)$$

$$\Rightarrow \langle q_1, q_4 \rangle$$

Thus every state in $m3$ has been expanded.

The construction of combine DFA is as follows.



$\langle q_1, q_4 \rangle \rightarrow$ both are final states

$\langle q_2, q_5 \rangle \rightarrow$ Non final states

$\langle q_2, q_6 \rangle \rightarrow$ Non final states

$\langle q_3, q_7 \rangle \rightarrow$ Non final states

\therefore From the state obtained from the above states, m_1 and m_2 are equivalent.

Example: c, cccc, dd, ddcc, dcddcd, dcddcdcc, are accepted by both m_1 and m_2 .

MINIMIZATION OF DFA

Minimization of DFA is a process of constructing an equivalent DFA with minimum number of states.

Equivalent states/ k – equivalence

Let q_i and q_j are **k – equivalent states** if and only if no string of length $\leq k$ can distinguish them.

If there exists a string of length k , which can distinguish q_i and q_j , the states q_i and q_j are said to be **k – distinguishable**.

Algorithm

Algorithm for minimization divides states of a DFA into blocks such that:

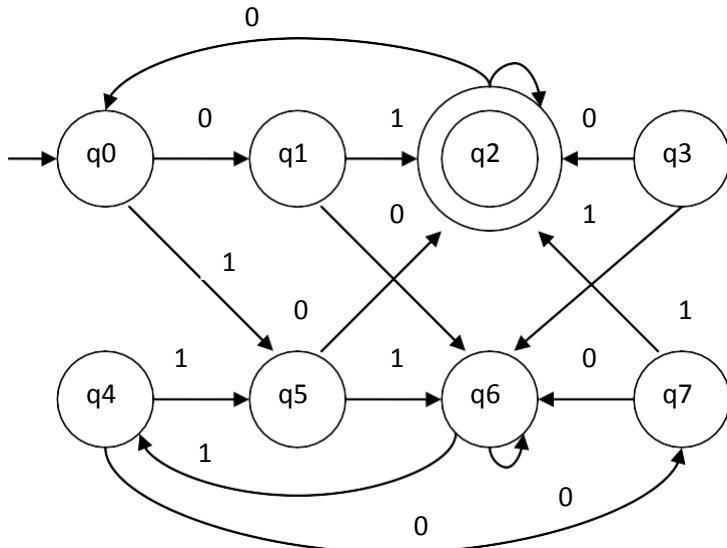
1. States in a block are equivalent.
2. No two states taken from two different blocks are equivalent.

The first step is to partition the states of M into blocks such that all states in a block are o -equivalent. This is done by placing,

- a. Final states into one block
- b. Non-final states into another block
- c. The next step is to obtain the partition, P_1 whose blocks consists of the set of states which are 1- equivalent [length = 1]

PROBLEMS

1) Minimise the following DFA



Solution:

0 - Equivalence partitioning:

The given set of states in the DFA are classified into two,

- 1) States that are not final states \Rightarrow **block 1**
- 2) Final states \Rightarrow **block 2**

$P_0 \Rightarrow$

$$\text{block1} = \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}$$

$$\text{block2} = \{q_2\}$$

1 – Equivalence partitioning:

This divides the existing non final states based on the inputs that the states accept.

Processing input ‘0’:

$$\delta(q_0, 0) = q_1 \quad [\text{block1}]$$

$$\delta(q_1, 0) = q_6 \quad [\text{block1}]$$

$$\delta(q_3, 0) = q_2 \quad [\text{block 2}]$$

$$\delta(q_4, 0) = q_7 \quad [\text{block1}]$$

\rightarrow Equivalent states since the transition result in same output.

$$\delta(q_5, 0) = q_2 \quad [\text{block 2}]$$

$$\delta(q_6, 0) = q_6 \quad [\text{block 1}]$$

$$\delta(q_7, 0) = q_6 \quad [\text{block 1}]$$

Processing Output -‘1’:-

$$\delta(q_0, 1) = q_5 \quad [\text{block 1}]$$

$$\delta(q_1, 1) = q_2 \quad [\text{block 2}]$$

$$\delta(q_3, 1) = q_6 \quad [\text{block 1}]$$

$$\delta(q_4, 1) = q_5 \quad [\text{block 1}]$$

$$\delta(q_5, 1) = q_6 \quad [\text{block 1}]$$

$$\delta(q_6, 1) = q_4 \quad [\text{block 1}]$$

$$\delta(q_7, 1) = q_2 \quad [\text{block 2}]$$

→ Equivalent states

From the above process of input 0 and 1, $\{q_3, q_5\}$, $\{q_1, q_7\}$ are a set of equivalent states.

∴ 1 – equivalence partitioning ⇒

$$\begin{aligned} P1 = & (q_3, q_5), (q_1, q_7), (q_0, q_4, q_6), (q_2) \\ \downarrow & \quad \downarrow \quad \downarrow \quad \downarrow \\ \text{Block 11} & \quad \text{Block 12} \quad \text{Block 13} \quad \text{Block 2} \end{aligned}$$

2 – Equivalence partitioning:

Block – 11

$$\delta(q_3, 0) = q_2 \quad \text{Mapped to the same block, Block 2}$$

$$\delta(q_5, 0) = q_2$$

$$\delta(q_3, 0) = q_6 \quad \text{Mapped to the same block, Block 12}$$

$$\delta(q_5, 0) = q_6$$

Block – 12

$$\delta(q_1, 0) = q_6 \quad \text{Mapped to the same block, Block 13}$$

$$\delta(q_7, 0) = q_6$$

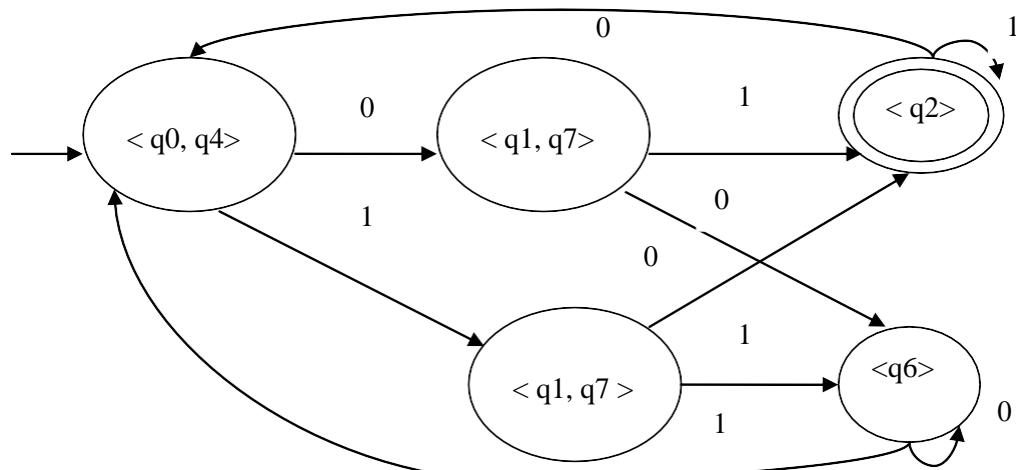
$\delta(q_1, 1) = q_2$	Mapped to the same block, Block 2
$\delta(q_7, 1) = q_2$	
Block - 13	Mapped to the same block, Block 12
$\delta(q_0, 0) = q_1$	
$\delta(q_4, 0) = q_7$	→ belong to the same block Block 12
$\delta(q_6, 0) = q_6$	
$\delta(q_0, 1) = q_5$	→ belong to the same block Block 11
$\delta(q_4, 1) = q_5$	
$\delta(q_6, 1) = q_4$	

Here q_6 doesn't map with any other states hence it is distinguished from $\{q_0, q_4\}$

2 – Equivalence partitioning ⇒

$$P_2 = (q_3, q_5), (q_1, q_7) (q_0, q_4) (q_6) (q_2)$$

Since further partitioning cannot be done, the final minimized DFA is given by



2) Minimise the DFA given below

AU MAY 2007

Input States \	a	b
$\rightarrow q_0$	q_0	q_3
q_1	q_2	q_5
q_2	q_3	q_4
q_3	q_0	q_5
q_4	q_0	q_6
q_5	q_1	q_4
* q_6	q_1	q_3

0- equivalence partitioning

$$P_0 = (q_0, q_1, q_2, q_3, q_4, q_5) (q_6)$$

\downarrow
 block 1

\downarrow
 block 2

1 – equivalence partitioning

$\delta(q_0, a) = q_0$	$\delta(q_0, b) = q_3$
$\delta(q_1, a) = q_2$	$\delta(q_1, b) = q_5$
$\delta(q_2, a) = q_3$	$\delta(q_2, b) = q_4$
$\delta(q_3, a) = q_0$	$\delta(q_3, b) = q_5$
$\delta(q_4, a) = q_0$	$\delta(q_4, b) = q_6$
$\delta(q_5, a) = q_1$	$\delta(q_5, b) = q_4$

There are no equivalent states. But still, q4 is the distinguishable state from the rest of the states

$$P_1 \Rightarrow (q_0, q_1, q_2, q_3, q_5) (q_4) (q_6)$$

\downarrow
 block 11

\downarrow
 block 12

\searrow
 block 2

2 – equivalence partitioning:

From the above transitions, there are no equivalent states but there are distinguishable states, {q2, q5} since

$$\delta(q_2, a) = q_3 \rightarrow \text{block 11}$$

$$\delta(q_5, a) = q_1$$

$$\delta(q_2, b) = q_3 \rightarrow \text{block 12}$$

$$\delta(q_3, b) = q_0$$

$$P_2 = (q_2, q_5) (q_0, q_1, q_3) (q_4) (q_6)$$

\swarrow
 Block 111

\downarrow
 Block 112

\downarrow
 Block 12

\searrow
 Block 2

3- Equivalence partitioning:

$$\delta(q_0, a) = q_0 \rightarrow \text{block 112}$$

$$\delta(q_1, a) = q_2 \rightarrow \text{block 111}$$

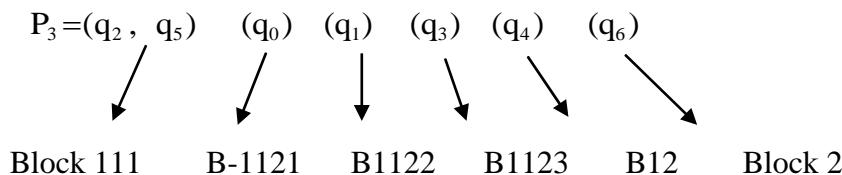
$$\delta(q_3, a) = q_0 \rightarrow \text{block 112}$$

$$\delta(q_0, b) = q_0 \rightarrow \text{block 112}$$

$$\delta(q_1, b) = q_0 \rightarrow \text{block 111}$$

$$\delta(q_3, b) = q_0 \rightarrow \text{block 111}$$

Since no states are equivalent, q_0, q_1, q_3 are distinguishable from each other.


4 - Equivalence partitioning:

But again in 4 – equivalence partitioning, (q_2, q_5) is considered.

$$(q_2, a) = q_3 \rightarrow \text{block 1123}$$

$$(q_5, a) = q_1 \rightarrow \text{block 1122}$$

$$(q_2, b) = q_4 \rightarrow \text{block 12}$$

$$(q_3, b) = q_4 \rightarrow \text{block 12}$$

$$P_4 = \{q_2\} \{q_5\} \{q_0\} \{q_1\} \{q_3\} \{q_4\} \{q_6\}$$

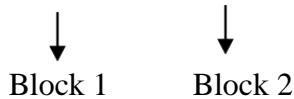
Thus the given DFA cannot be minimized since no equivalent states are obtained.

3) Construct a min –DFA for

Inputs → States ↓	0	1
→ q_0	q_1	q_4
q_1	q_2	q_5
* q_2	q_3	q_7
q_3	q_4	q_7
q_4	q_5	q_8
* q_5	q_6	q_1
q_6	q_7	q_1
q_7	q_8	q_2
* q_8	q_0	q_4

0- equivalence Partitioning:-

$$P_0 = (q_0, q_1, q_3, q_4, q_6, q_7) \quad (q_2, q_5, q_8)$$

**1- equivalence partitioning:-**

$$\begin{aligned}
 \delta(q_0, 0) &= q_1 \rightarrow \text{Block 1} \\
 \delta(q_1, 0) &= q_2 \rightarrow \text{Block 2} \\
 \delta(q_3, 0) &= q_4 \rightarrow \text{Block 1} \\
 \delta(q_4, 0) &= q_5 \rightarrow \text{Block 2} \\
 \delta(q_6, 0) &= q_7 \rightarrow \text{Block 1} \\
 \delta(q_7, 0) &= q_8 \rightarrow \text{Block 2} \\
 \delta(q_0, 1) &= q_4 \rightarrow \text{block 1} \\
 \delta(q_1, 1) &= q_5 \rightarrow \text{block 2} \\
 \delta(q_3, 1) &= q_7 \rightarrow \text{block 1} \\
 \delta(q_4, 1) &= q_8 \rightarrow \text{block 2} \\
 \delta(q_6, 1) &= q_1 \rightarrow \text{block 1} \\
 \delta(q_7, 1) &= q_2 \rightarrow \text{block 2}
 \end{aligned}$$

From the above transitions,

$$P_1 = (q_0, q_3, q_6) \quad (q_1, q_4, q_7) \quad (q_2, q_5, q_8)$$

\downarrow \downarrow \downarrow
 block 11 block 12 block 2

2- Equivalence partitioniong:

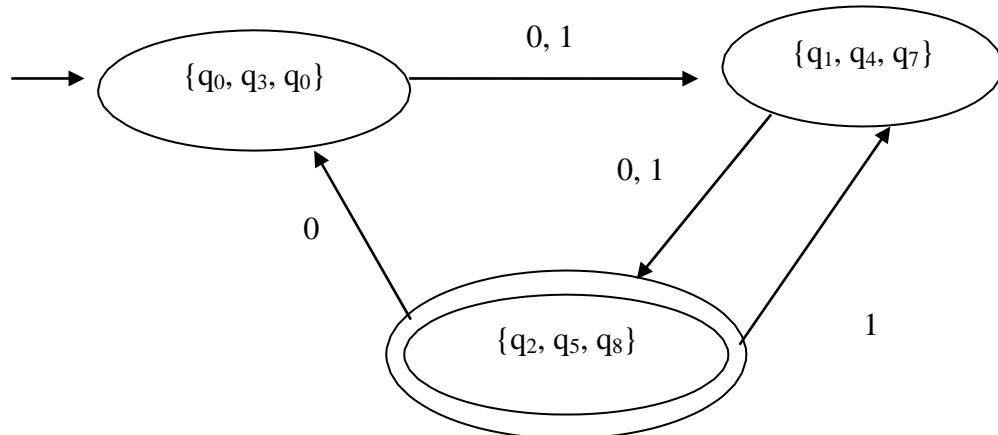
$\delta(q_0, 0) = q_1$	$\delta(q_0, 1) = q_4$
$\delta(q_3, 0) = q_4$	$\delta(q_3, 1) = q_7$
$\delta(q_6, 0) = q_7$	$\delta(q_6, 1) = q_1$
$\delta(q_1, 0) = q_2$	$\delta(q_1, 1) = q_5$
$\delta(q_4, 0) = q_5$	$\delta(q_4, 1) = q_8$
$\delta(q_7, 0) = q_8$	$\delta(q_7, 1) = q_2$

$$\begin{array}{ll}
 \delta(q_2, 0) = q_3 & \delta(q_2, 1) = q_7 \\
 \delta(q_5, 0) = q_6 & \text{Block 11} \qquad \delta(q_5, 1) = q_1 \qquad \text{Block 11} \\
 \delta(q_8, 0) = q_0 & \delta(q_8, 1) = q_4
 \end{array}$$

$$P_2 = (q_0, q_3, q_6) (q_1, q_4, q_7) (q_2, q_5, q_8)$$

The States cannot be further subdivided.

\therefore The minimized DFA is given by



UNIT – 3

CONTEXT FREE GRAMMAR AND LANGUAGES

GRAMMAR

- Every language is generated by some grammar.
- It produces strings of a language by means of replacing symbols.
- In general, a grammar is a set of production rules for generating strings in a language.
- A grammar can also be called as formal grammar, which generates strings in a formal language.
- A grammar generates a language with the aid of **four elements, G=(V,T,P,S)** such as,
 - Terminal symbol
 - Non terminal symbol
 - Start symbol
 - Production rules
- **Terminal** refers to the symbols of the alphabet being used.
 - There are several types of terminals namely –
 - Binary alphabet → 0 and 1
 - Uppercase alphabet → A, B, … , Z
 - Lowercase alphabet → a, b, … , z
 - Decimal alphabets → 0, 1, 2, … , 9 etc.
 - Most commonly used terminals are binary and lowercase types.
- The **non-terminal symbols** are those variables that denote/produce some terminals in turn.
 - Non-terminals are equivalent to the states of automation.
 - They are usually represented by uppercase alphabets like. A, B, … , Z.
- The **start symbol** is a special non-terminal that works similar to the start state of a finite automaton.
 - It is represented by an element in variable set.
- The **production rules** are the translation functions/replacement rules of the variables.

Example: (Informal)

- Consider the string: „sentence“.

A sentence is formed by „subject“, „verb“, „object“.

So,

Start symbol: $\langle \text{sentence} \rangle$

and,

$\langle \text{sentence} \rangle \Rightarrow \langle \text{subject} \rangle \langle \text{verb} \rangle \langle \text{object} \rangle$

- Now the above statement forms a production rule, which requires some strings to be substituted/replaced for subject, verb and object.
- Let

$\langle \text{subject} \rangle \Rightarrow \text{fruits}$

$\langle \text{verb} \rangle \Rightarrow \text{are}$

$\langle \text{object} \rangle \Rightarrow \text{fresh}$

- Therefore,

$\langle \text{sentence} \rangle \Rightarrow \text{fruits are fresh} \Rightarrow \text{one string in the language}$

where

- Terminals: fruits, are, fresh
- Nonterminals: $\langle \text{sentence} \rangle$, $\langle \text{subject} \rangle$, $\langle \text{verb} \rangle$, $\langle \text{object} \rangle$
- Start symbol: $\langle \text{sentence} \rangle$
- Production rules:

$\langle \text{sentence} \rangle \Rightarrow \langle \text{subject} \rangle \langle \text{verb} \rangle \langle \text{object} \rangle$

$\langle \text{subject} \rangle \Rightarrow \text{fruits}$

$\langle \text{verb} \rangle \Rightarrow \text{are}$

$\langle \text{object} \rangle \Rightarrow \text{fresh}$

- A grammar thus generates a formal language, which is a set of strings formed by applying production rules that is derived from the start symbol.

Example: (formal)

(AU – Dec 2004, May 2005, June 2006)

- Consider, the terminals a and b, the start symbol, S and the production rules:

$S \rightarrow aSb$

$S \rightarrow ab$

- This may generate,

$$\begin{aligned} S \rightarrow aSb &\rightarrow \text{by applying rule 1.} \\ \rightarrow aaSbb &\rightarrow \text{by applying rule 1 on } S. \\ \rightarrow aaabbb &\rightarrow \text{by applying rule 2 on } S. \end{aligned}$$
- Thus the above production rules may generate,

$$S \rightarrow \{ab, aabb, aaabbb, a^4b^4, \dots\}$$
- Therefore the production rules generate a language containing equal number of a's and b's and is defined as

$$L = \{a^n b^n \mid n \geq 1\}$$

PROBLEMS ON GRAMMARS

1. Construct the grammar representing the set of palindromes over $(0 + 1)^*$.

Solution:

The first possibility is having a string of single / zero length.

$$S \rightarrow 0 \mid 1 \mid \epsilon$$

For other strings,

$$\begin{aligned} S \rightarrow 0S0 &\rightarrow \text{first and last data are equal} \\ S \rightarrow 1S1 &\rightarrow \text{first and last data are equal} \end{aligned}$$

Thus the grammar is given by

$$S \rightarrow 0 \mid 1 \mid \epsilon$$

$$S \rightarrow 0S0$$

$$S \rightarrow 1S1$$

2. Let $G = \{S\}, \{a,b\}, P, S\}$ with the productions,

$$P: S \rightarrow aSa$$

$$S \rightarrow bSb$$

$$S \rightarrow \epsilon$$

Find the language generated by the grammar.

Solution:

- a) $S \rightarrow a\underline{S}a$ [since $S \rightarrow aSa$]

$$\begin{aligned} \rightarrow aa &[since S \rightarrow \epsilon] \end{aligned}$$
- b) $S \rightarrow b\underline{S}b$ [since $S \rightarrow bSb$]

$$\begin{aligned} \rightarrow bb &[since S \rightarrow \epsilon] \end{aligned}$$

c) S	$\rightarrow a\underline{S}a$	[since $S \rightarrow aSa$]
	$\rightarrow aa\underline{S}aa$	[since $S \rightarrow aSa$]
	$\rightarrow aaaa$	[since $S \rightarrow \epsilon$]
d) S	$\rightarrow a\underline{S}a$	[since $S \rightarrow aSa$]
	$\rightarrow ab\underline{S}ba$	[since $S \rightarrow bSb$]
	$\rightarrow abba$	[since $S \rightarrow \epsilon$]
e) S	$\rightarrow b\underline{S}b$	[since $S \rightarrow bSb$]
	$\rightarrow bb\underline{S}bb$	[since $S \rightarrow bSb$]
	$\rightarrow bbbb$	[since $S \rightarrow \epsilon$]
f) S	$\rightarrow b\underline{S}b$	[since $S \rightarrow bSb$]
	$\rightarrow ba\underline{S}ab$	[since $S \rightarrow aSa$]
	$\rightarrow baab$	[since $S \rightarrow \epsilon$]
g) S	$\rightarrow a\underline{S}a$	[since $S \rightarrow aSa$]
	$\rightarrow ab\underline{S}ba$	[since $S \rightarrow bSb$]
	$\rightarrow aba\underline{S}aba$	[since $S \rightarrow aSa$]
	$\rightarrow aba \left \begin{matrix} aba \\ w \quad \quad w^R \end{matrix} \right.$	[since $S \rightarrow \epsilon$]

From the above examples, the language can be defined as,

$$L = \{ww^R \mid w \in (a, b)^*\}$$

Every string, when broken into two, contains the reverse of the first half as second half. The language represents a set of palindromes over {a, b}.

3. Construct a grammar for $L = \{a^n \mid n \text{ is odd}\}$.

Solution:

$$\text{When } n = 1 \Rightarrow S = a^1 = a \Rightarrow [S \rightarrow a]$$

$$\text{When } n = 3 \Rightarrow S = a^3 = aaa \Rightarrow [S \rightarrow aSa]$$

$$\text{When } n = 5 \Rightarrow S = a^5 = aaaaa \Rightarrow [S \rightarrow aSa]$$

.

.

.

Therefore the production shall be defined as,

$$S \rightarrow a$$

$$S \rightarrow aSa$$

The grammar $G = (V, T, P, S)$ is given by

$$V = \{S\}$$

$$T = \{a\}$$

$$P = \{S \rightarrow aSa \mid a\}$$

$$S = \{S\}$$

TYPES OF GRAMMARS

- Noam Chomsky formalized grammars in 1956.
- He classified them into four types which are collectively called as „the Chomsky hierarchy“.
- Chomsky hierarchy consists of the following four levels, namely
 - Type 0 grammars / unrestricted grammars
 - Type 1 grammars / context sensitive grammars
 - Type 2 grammars / context free grammars
 - Type 3 grammars / regular grammars

Type 0 grammars

- These are free / unrestricted grammars that have no restrictions on the production rules.
- All formal grammars belong to this class.
- The grammars, G is defined by four tuples as $G = (V, T, P, S)$ where

$$V \rightarrow \text{finite set of variable / non terminals}$$

$$T \rightarrow \text{finite set of terminals}$$

$$P \rightarrow \text{set of production rules}$$

$$S \rightarrow \text{start symbol}$$

- The production rule is of the form,

$$\alpha \rightarrow \beta \mid \alpha \neq \epsilon$$

where $\alpha, \beta \rightarrow$ can be strings composed of terminals and non-terminals.

- The most important property is that the left hand side (α) should contain at least one non-terminal and cannot be empty.

- Type 0 grammar are also called as **Recursively enumerable/Turing acceptable** since the machine halts if the input is accepted with the answer, „YES“, else may loop forever.

Properties of type 0 grammars

- These grammars are closed under,
 - Kleene star
 - Concatenation
 - Intersection
 - Union

Example

- All computable functions are examples of type 0 grammars.
- Almost all natural languages fall under this category.

Type 1 grammar

- The languages that follow type 1 grammars are called as **context-sensitive grammars** (CSG).
- A language is said to be context sensitive if the production rules are of the form,

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

$\gamma \rightarrow$ non-empty

$\alpha, \beta \rightarrow$ can be empty

where,

A is a non-terminal symbol

α, β, γ are any combination of terminals and non-terminals.

- Thus CSG possess two properties
 - At least one symbol on the left hand side should be a non terminal.
 - The number of symbols on the left hand side does not exceed the number of symbols on the right hand side.
- The machine/automaton that recognizes a CSG is called **linear-bounded machine/automaton** which is a finite automaton with memory to store symbols.
- Most importantly, every language described by context free grammars can be described by a context sensitive grammar.

Properties

- The type 1 grammar are closed under
 - Union
 - Concatenation
 - Complementation
 - Intersection
 - Substitution
 - Inverse homomorphism
 - Kleene plus

Example

- Most programming languages are context sensitive.
 - Eg: $L = \{a^n b^n c^n | n \geq 1\}$, $L = \{a^{i^2} | i \geq 1\}$ are all context sensitive.

Type 2 grammars

- Type 2 grammars are **context free grammars**.
- A grammars, $G = (V, T, P, S)$ is said to be context free if the production rule is of the form,

$$A \rightarrow \alpha$$

The transition allows

$$A \rightarrow \epsilon \text{ [i.e., } \alpha \rightarrow \epsilon]$$

where,

A is a non terminal symbol

α is any terminal or non-terminal symbol

- Here, the left hand side of the transition rule consists of only one non-terminal.
- Type 2 grammars are the basis of the syntax of most of the programming languages such as XML.

Properties

- The CFG is closed under
 - Union
 - Concatenation
 - Kleene closure

- It is not closed under complementation, substitution, reversal.

Example

- Consider the production, $P \Rightarrow \{S \rightarrow aSa, S \rightarrow bSb, S \rightarrow \epsilon\}$
- Since

$$\begin{aligned} S &\rightarrow a\underline{S}a \\ &\rightarrow aa\underline{S}aa \quad [\text{as } S \rightarrow aSa] \\ &\rightarrow aab\underline{S}baa \quad [\text{as } S \rightarrow bSb] \\ &\rightarrow aabbbaa \quad [\text{as } S \rightarrow \epsilon] \end{aligned}$$

- Thus, S may generate

$$S = \{\epsilon, aa, bb, abba, aabbaa, abaaba, \dots\}$$

- Thus, the language is defined as

$$L(G) = \{w w^R \mid w \in \{a, b\}^*\}$$

- CFG can be handled using the pushdown automaton that uses stack memory to store the symbols.

Type 3 grammars

- Type 3 grammars are **regular grammars** that describe regular / formal languages.
- These grammars contain production rules consisting of
 - only one non-terminal at the left hand side
 - the right hand side having a single terminal and may or may not be followed by non terminals

Example

- $A \rightarrow \epsilon$
- $A \rightarrow a$
- $A \rightarrow b$
- $A \rightarrow aA$ etc

Types

- There are two types of regular grammars namely,
 - Right linear / Right regular grammar
 - Left linear / Left regular grammar

Right linear grammar

- This is a regular grammar with the production rules of the form

$$A \rightarrow \alpha \text{ (or) } A \rightarrow \alpha B$$

where,

$A, B \rightarrow$ non terminal symbols

$\alpha \rightarrow$ terminal symbol

- These grammars are right-branching in nature since the substitution is done for the right most non terminal, if available.

Left linear grammar

- This is a regular grammar with the production rules of the form:

$$A \rightarrow \alpha \text{ (or) } A \rightarrow B \alpha$$

where,

A and B is non terminals and

α is a terminal symbol

- Since the substitution is done for the left most non-terminal, these grammars are left branching in nature.

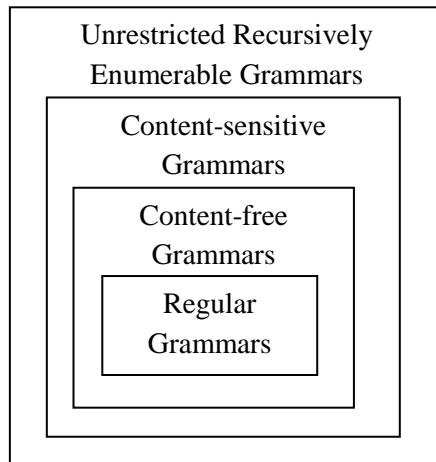
Properties of Regular grammars

- The regular grammars generates regular languages which are closed under,
 - Union
 - Concatenation
 - Intersection
 - Complementation
 - Difference
 - Reversal
 - Closure
 - Homomorphism
 - Inverse homomorphism

Chomsky hierarchy

- The four forms of grammars follows the given hierarchy,
- Type 3 grammar \subseteq type 2 grammar \subseteq type 1 grammar \subseteq type 0 grammar

\Rightarrow Regular grammars \subseteq context free grammars \subseteq context sensitive grammars \subseteq unrestricted grammars



Chomsky hierarchy

CONTEXT-FREE GRAMMARS AND LANGUAGES

- A grammar is a means of representing a language.
- A computational language is represented using a set of recursive equations.
- The equations are called as productions.

Formal Definition

(JUNE – 2013)

A context free grammar (CFG) is described by four tuples (quadruple) as,

$$\mathbf{G} = (\mathbf{V}, \mathbf{T}, \mathbf{P}, \mathbf{S})$$

where,

\mathbf{V} is a finite set of variables / non terminals

\mathbf{T} is a finite set of terminals / terminal symbols

\mathbf{P} is a finite set of transitions / production rules

\mathbf{S} is the start symbol, where $S \in V$.

- The production rules are the recursive definitions of the languages.
- The production is stated as,

Head of the production \rightarrow body of the production

where,

Head of the production is any variable defined in V .

Body of the production is set of zero or more terminals and/or non terminals.

Thus the production is of the form,

$$V_i \rightarrow \alpha_i ,$$

where,

$$V_i \in V$$

$$\alpha_i \in \{VUT\}$$

Notations

- The terminals are denoted by digits (0, …, 9) or lower case letters (a, …, z) etc. It also includes special characters like +, -, *, (,) etc.
- Non terminals are denoted by uppercase letters (A, …, Z)

Example

Generation of palindromes over the terminals {a, b}

(AU – June 2014)

Basis: $\{\epsilon, a, b\}$

Induction: $\{aa, bb, aba, bab, aaa, bbb, aaaa, abba, bbbb, baab, \dots\}$

To generate the basics form of strings, we get

$$S \rightarrow \epsilon$$

$$S \rightarrow a$$

$$S \rightarrow b$$

Other strings, specified by the induction are given by

$S \rightarrow aSa \Rightarrow$ generates aa, aaa, aba, aaaa, abba etc.,

$S \rightarrow bSb \Rightarrow$ generates bb, bbb, bab, bbbb, baab, etc,

Thus the grammar, $G = (V, T, P, S)$ is given by $G = (\{S\}, \{a,b\}, P, \{S\})$ where P is given by

$$S \rightarrow \epsilon | a | b | aSa | bSb$$

The language of a grammar

- The language of a grammar, $G = (V, T, P, S)$ is denoted as $L(G)$.
- Every CFG generates its corresponding language.
- Every string of a language is generated by applying the production rules, a finite number of times.
- Each string is derived from the start symbol and ends with a final string, consisting of a set of terminals.

Formal definition

(AU – Nov/Dec 2009, May/June 2013)

- The language $L(G)$ of a grammar, G is a set of strings derived from the start symbol.
- It is denoted as

$$L(G) = \left\{ w \mid w \in T^* \text{ and } S \xrightarrow[G]{*} w \right\}$$

„*“ denotes that the grammar, G is applied over S several times.

- If the grammar is applied once, it is denoted as $S \xrightarrow[G]{} w$
- The language generated from a context - free grammar is called a **context - free language**.

Example

Design the language for the CFG given. The grammar, $G = (\{S\}, \{0,1\}, P, \{S\})$ where

$$P: S \rightarrow 0S1$$

$$S \rightarrow 01$$

Solution:

For the given grammar, we can generate the strings as

$$\begin{array}{lllll} S \rightarrow 01 & S \rightarrow 0\underline{S}1 & S \rightarrow 0\underline{S}1 & S \rightarrow 0\underline{S}1 & \dots \\ & \rightarrow 0011 & \rightarrow 00\underline{S}11 & \rightarrow 00\underline{S}11 & \\ & & \rightarrow 000111 & \rightarrow 000\underline{S}111 & \\ & & & \rightarrow 00001111 & \end{array}$$

Thus the strings are $\{01, 0011, 000111, 0^4 1^4, 0^5 1^5, \dots\}$

The language, $L = \{0^n 1^n \mid n \geq 1\}$ generates equal number of zeroes and ones.

PROBLEMS ON CFG

1. Construct a CFG for the language, $L = \{w C w^R \mid w \text{ is a string in } (a+b)^*\}$.

(AU – Nov / Dec 2010)

Solution:

The intermediate symbol is „C“. The string written on the left of „C“ is reversed at the right side.

The possible strings generated from L can be $\{c, aca, bcb, aacaa, bbcbb, abcba, bacab, abacaba, babcbab, aaacaaa, \dots\}$

\therefore The production rules are defined to be,

$$S \rightarrow c$$

$$S \rightarrow aSa$$

$$S \rightarrow bSb$$

Thus the grammar, $G = (V, T, P, S)$ where

$$V = \{S\}$$

$$T = \{a, b, c\}$$

$$P = \{S \rightarrow aSa \mid bSb \mid c\}$$

$$S = \{S\}$$

2. Construct a CFG representing the set of palindromes over $(0+1)^*$.

Solution:

The possible productions are $\{\epsilon, 0, 1, 00, 11, 000, 111, \text{etc}\}$

So the productions are

$$P \Rightarrow S \rightarrow \epsilon$$

$$S \rightarrow 0$$

$$S \rightarrow 1$$

$$S \rightarrow 0S0$$

$$S \rightarrow 1S1$$

Thus the CFG is given by $G = (V, T, P, S)$ where

$$V = \{S\}$$

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

$$P = \{S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \epsilon\}$$

$$S = \{S\}$$

3. Construct a CFG for $L = \{a^n b^n \mid n \geq 0\}$

(AU May/June 2009, Dec 2013)

Solution:

The possible productions are $\{\epsilon, ab, aabb, aaabbb, a^4 b^4, \text{etc}\}$

So the productions are,

$$S \rightarrow \epsilon$$

$$S \rightarrow aSb$$

Thus, the CFG, $G = (V, T, P, S)$ is given by

$$V = \{S\}$$

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

$$P = \{S \rightarrow aSb \mid \epsilon\}$$

$$S = \{S\}$$

4. Construct a grammar for $L = \{a^n b^k \mid k > n \geq 0\}$ with $T = \{a, b\}$ and $V = \{S, B\}$

Solution:

The possible strings generated from L can be {b, bb, bbb, b^7 , abb, aabbb, abbbbb, ...}

The number of b's are greater than the number of a's, where the number of a's can be zero more.

The production P is given by

$$S \rightarrow b$$

$$S \rightarrow bS$$

$$B \rightarrow S$$

$$B \rightarrow aBb$$

where, $V = \{S, B\}$, $T = \{a, b\}$, $S = \{S\}$

5. Let $G = \{\{S, C\}, \{a, b\}, P, S\}$ where $S \rightarrow aCa$, $C \rightarrow aCa|b$. Find $L(G)$.

(AU – Nov/Dec 2003, Nov/Dec 2005)

Solution:

1. $S \rightarrow a\underline{C}a \rightarrow aba$
2. $S \rightarrow a\underline{C}a \rightarrow aa\underline{C}aa \rightarrow aabaa$
3. $S \rightarrow a\underline{C}aa \rightarrow aaa\underline{C}aaa \rightarrow aaabaaa$

$$\therefore L(G) = \{a^n b a^n \mid n \geq 1\}$$

6. Find out the CFL for,

(AU – Dec 2009, Dec 2011, June 2009)

$$S \rightarrow aSb \mid aAb$$

$$A \rightarrow bAa$$

$$A \rightarrow ba$$

Solution:

1. $S \rightarrow a\underline{S}b$

$\rightarrow aa\text{A}bb$ [using $S \rightarrow aAb$]
 $\rightarrow aababb$ [using $A \rightarrow ba$]

2. $S \rightarrow a\text{S}b$

$\rightarrow aa\text{A}bb$ [using $S \rightarrow aAb$]
 $\rightarrow aab\text{A}abb$ [using $A \rightarrow bAa$]
 $\rightarrow aabbaabb$ [using $A \rightarrow ba$]

3. $S \rightarrow a\text{A}b$

$\rightarrow abab$ [using $A \rightarrow ba$]

4. $S \rightarrow a\text{A}b$

$\rightarrow ab\text{A}ab$ [using $A \rightarrow bAa$]
 $\rightarrow abbaab$ [using $A \rightarrow ba$]

From the above strings, the CFL is given as,

$$L(G) = \{ a^n b^m a^m b^n \mid m, n \geq 1 \}$$

7. Consider $\Sigma = \{a, b, c, (,), +, *, \dots, \epsilon\}$. Construct a CFG that generate all strings in Σ^* over $\{a, b\}$. (AU – Nov/Dec 2006)

Solution:

The CFG is given as

$$\begin{aligned} S &\rightarrow S + S \\ S &\rightarrow S * S \\ S &\rightarrow (S) \\ S &\rightarrow a \mid b \mid \epsilon \end{aligned}$$

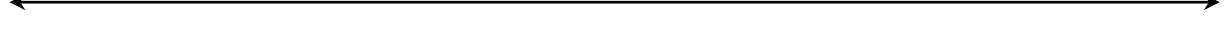
8. Write a CFG that generates string of balanced parenthesis.

Solution:

Example: (), () (), () () (), etc

The productions for the above set of strings are given by

$$\begin{aligned} S &\rightarrow SS \\ S &\rightarrow (S) \\ S &\rightarrow \epsilon \end{aligned}$$



Example: Consider $()()$

$$S \Rightarrow \underline{SS} \quad [\text{using } S \rightarrow SS]$$

$$\Rightarrow (\underline{S}) S \quad [\text{using } S \rightarrow (S)]$$

$$\Rightarrow () \underline{S} \quad [\text{using } S \rightarrow \epsilon]$$

$$\Rightarrow () (\underline{S}) \quad [\text{using } S \rightarrow (S)]$$

$$\Rightarrow () () \quad [\text{using } S \rightarrow \epsilon]$$

9. Design a CFG for the regular expression, $r = (a + b)^* aa (a + b)^*$

Solution:

The grammar consists of at least two a's and may have any number of „a“ or „b“ before and after it.

Example: aa, aaa, aab, baa, aaab, baaaabb, bbbaaa, etc,

The production is given by

$$P \Rightarrow \begin{cases} S \rightarrow AaaA \\ A \rightarrow aA \mid bA \mid \epsilon \end{cases}$$

where,

$$V = \{S, A\}$$

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

$$S = \{S\}$$

10. Design a CFG for $\Sigma = (a, b)$ that generates

- (i) All strings with exactly one a.
- (ii) All strings with at least one a
- (iii) All strings with at least 3 a's

Solution:

- (i) All strings with exactly one a

Example: a, b, bbb, ba, ab, abbbb, babb, bbbabbb, etc

The productions are given as

$$P \Rightarrow \begin{cases} S \rightarrow AaA \\ A \rightarrow bA \mid \epsilon \end{cases}$$

where,

$$V = \{S, A\}$$

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

$$S = \{S\}$$

(ii) All strings with at least one a

Example: a, aaa, aabb, baaa, aabbaa, etc

The productions are given by

$$P \Rightarrow \begin{cases} S \rightarrow AaA \\ A \rightarrow aA \mid bA \mid \epsilon \end{cases}$$

where,

$$V = \{S, A\}$$

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

$$S = \{S\}$$

(iii) All strings with atleast 3 a's

Example: aaa, baaa, ababa, aabbaa, etc

The production rules are given by

$$P \Rightarrow \begin{cases} S \rightarrow AaAaAaA \\ A \rightarrow aA \mid bA \mid \epsilon \end{cases}$$

where,

$$V = \{S, A\}$$

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

$$S = \{S\}$$

11. Construct CFG for $G = \{a^n b a^n \mid n \geq 1\}$

(AU – May/June 2009)

Solution:

Example: aba, aabaa, aaabaaa, etc

The production rule is given by $G = (V, T, P, S)$ where

$$P \Rightarrow \{S \rightarrow aSa \mid b\}$$

where,

$$V = \{S\}$$

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

$$S = \{S\}$$

DERIVATION AND LANGUAGES

Derivations

- Derivations are the set of strings that are derived from the start symbol, after applying the production rules, a finite number of times.

$$S \xrightarrow[G]{*} \omega \mid \omega \epsilon T^*$$

Representation of derivations

- Derivations are represented in either of the two forms.
- They are,
 - Sentential form
 - Parse tree form

Types of derivations

(AU – May/June 2009)

There are two types of derivations, namely

- Leftmost derivations
- Rightmost derivations

Sentential form

- Sentential form is the derivation derived from the start symbol, by applying the rules/productions.
- Let the $G = (V, T, P, S)$ be a CFG then

$$S \xrightarrow{*} \alpha$$

is in sentential form, where $\alpha \in (TUV)^*$

Example:

Let $G = (V, T, P, S)$ be given by $G = (\{S, A, B\}, \{0, 1\}, P, \{S\})$. (AU – June 2009)

$$\begin{aligned} P: \quad & S \rightarrow A1B \\ & A \rightarrow 0A \mid \epsilon \\ & B \rightarrow 0B \mid 1B \mid \epsilon \end{aligned}$$

Solution:

Consider the string 00101, which is to be generated by the given grammar using sentential form.

$S \Rightarrow \underline{A}1B$	[Using $S \rightarrow A1B$]
$\Rightarrow 0\underline{A}1B$	[Using $A \rightarrow 0A$]
$\Rightarrow 00\underline{A}1B$	[Using $A \rightarrow 0A$]
$\Rightarrow 001\underline{B}$	[Using $A \rightarrow \epsilon$]
$\Rightarrow 0010\underline{B}$	[Using $B \rightarrow 0B$]
$\Rightarrow 00101\underline{B}$	[Using $B \rightarrow 1B$]
$\Rightarrow 00101$	[Using $B \rightarrow \epsilon$]

Thus the string $00101 \in L(G)$.

Types of sentential form

There are two types of sentential forms namely,

- Left sentential forms
- Right sentential forms

Left sentential form

- When the derivations are generated by expanding the leftmost symbol is called left sentential form.
- It is denoted as, $S \xrightarrow[\ell_m]{*} \alpha$

Example

Consider the grammar given below

$$\begin{aligned} S &\rightarrow A1B \\ A &\rightarrow 0A \mid \epsilon \\ B &\rightarrow 0B \mid 1B \mid \epsilon \end{aligned}$$

Give the leftmost derivation for the string 1001.

Solution:

$S \xrightarrow[\ell_m]{*} \underline{A}1B$	[Using $S \rightarrow A1B$]
$\xrightarrow[\ell_m]{*} 1\underline{B}$	[Using $A \rightarrow \epsilon$]
$\xrightarrow[\ell_m]{*} 10\underline{B}$	[Using $B \rightarrow 0B$]
$\xrightarrow[\ell_m]{*} 100\underline{B}$	[Using $B \rightarrow 0B$]

$\xrightarrow[\ell_m]{} 1001\mathbf{B}$	[Using $B \rightarrow 1B$]
$\xrightarrow[\ell_m]{} 1001$	[Using $B \rightarrow \epsilon$]

- Thus the string „1001“ is thus derived using left sentential form.
- At each step, the left most variable is expanded.

Right sentential form

- The derivation generated by performing substitution on the right most variable is called right sentential form.
- It is denoted as

$$\underset{\text{rm}}{S \xrightarrow{*} \alpha}$$

Example

Consider

$$\begin{aligned} S &\rightarrow A1B \\ A &\rightarrow 0A|\epsilon \\ B &\rightarrow 0B|1B|\epsilon \end{aligned}$$

Generate the right-most derivation for the string ‘1001’.

Solution:

$\underset{\text{rm}}{S \xrightarrow{} A1\mathbf{B}}$	[Using $S \rightarrow A1B$]
$\xrightarrow[\text{rm}]{} A10\mathbf{B}$	[Using $B \rightarrow 0B$]
$\xrightarrow[\text{rm}]{} A100\mathbf{B}$	[Using $B \rightarrow 0B$]
$\xrightarrow[\text{rm}]{} A1001\mathbf{B}$	[Using $B \rightarrow 1B$]
$\xrightarrow[\text{rm}]{} A1001$	[Using $B \rightarrow \epsilon$]
$\xrightarrow[\text{rm}]{} 1001$	[Using $A \rightarrow \epsilon$]

- Thus the string „1001“ is derived from the start symbol using right most derivation.
- The right most variable is expanded at each step.
- The left most and right most derivations are equivalent to the left and right sentential forms.

PROBLEMS ON DERIVATIONS

1. Construct a CFG for the language having equal number of a's and b's over {a, b}.
Derive LMD and RMD for the string. 'abab'.

Solution:

The possible productions are $\{\epsilon, ab, aabb, abab, baba, bbaa, baab, abba, aaabbb, \text{etc}\}$

The productions are given as

$$\begin{aligned} S &\rightarrow \epsilon \\ S &\rightarrow SS \\ S &\rightarrow aSb \\ S &\rightarrow bSa \end{aligned}$$

where, $V = \{S\}$, $T = \{a, b\}$, $S = \{S\}$

Left most derivations for $S = abab$

$$\begin{aligned} S &\xrightarrow[\ell_m]{} aSb & [\text{Since } S \rightarrow aSb] \\ &\xrightarrow[\ell_m]{} abSb & [\text{Since } S \rightarrow bSa] \\ &\xrightarrow[\ell_m]{} abab & [\text{Since } S \rightarrow \epsilon] \end{aligned}$$

Right most derivations for $S = abab$

$$\begin{aligned} S &\xrightarrow[\rm rm]{} SS & [\text{Since } S \rightarrow SS] \\ &\xrightarrow[\rm rm]{} SaSb & [\text{Since } S \rightarrow aSb] \\ &\xrightarrow[\rm rm]{} aSab & [\text{Since } S \rightarrow \epsilon] \\ &\xrightarrow[\rm rm]{} aSbab & [\text{Since } S \rightarrow aSb] \\ &\xrightarrow[\rm rm]{} abab & [\text{Since } S \rightarrow \epsilon] \end{aligned}$$

2. Consider the CFG, $G = \{S, I\}, T, P, S\}$ where $T = \{+, *, (), a, b, 0, 1\}$ and P is given as

$$\begin{array}{lll} S \rightarrow I & I \rightarrow a & I \rightarrow I1 \\ S \rightarrow S+S & I \rightarrow b & I \rightarrow I0 \\ S \rightarrow S*S & I \rightarrow Ia \\ S \rightarrow (S) & I \rightarrow Ib \end{array}$$

← →

Derive $a^*(a + b00)$.

Solution

Left most derivations

$S \xrightarrow[\ell_m]{} \underline{S}^* S$	[Using $S \rightarrow S^* S$]
$\Rightarrow \underline{I}^* S$	[Using $S \rightarrow I$]
$\Rightarrow a^* \underline{S}$	[Using $I \rightarrow a$]
$\Rightarrow a^* (\underline{S})$	[Using $S \rightarrow (S)$]
$\Rightarrow a^* (\underline{S+S})$	[Using $S \rightarrow (S + S)$]
$\Rightarrow a^* (\underline{I+S})$	[Using $S \rightarrow I$]
$\Rightarrow a^* (a + \underline{S})$	[Using $I \rightarrow a$]
$\Rightarrow a^* (a + \underline{I})$	[Using $S \rightarrow I$]
$\Rightarrow a^* (a + \underline{I0})$	[Using $I \rightarrow I0$]
$\Rightarrow a^* (a + \underline{I00})$	[Using $I \rightarrow I0$]
$\Rightarrow a^* (a + b00)$	[Using $I \rightarrow b$]

Right most derivations

$S \xrightarrow[\rm rm]{} S^* \underline{S}$	[Using $S \rightarrow S^* S$]
$\Rightarrow S^* (\underline{S})$	[Using $S \rightarrow (S)$]
$\Rightarrow S^* (S + \underline{S})$	[Using $S \rightarrow S + S$]
$\Rightarrow S^* (S + \underline{I})$	[Using $S \rightarrow I$]
$\Rightarrow S^* (S + \underline{I0})$	[Using $I \rightarrow I0$]
$\Rightarrow S^* (S + \underline{I00})$	[Using $I \rightarrow I0$]
$\Rightarrow S^* (\underline{S} + b00)$	[Using $I \rightarrow B$]
$\Rightarrow \underline{S}^* (a + b00)$	[Using $S \rightarrow a$]

$$\xrightarrow[\text{rm}]{\Rightarrow} I^* (a + b00) \quad [\text{Using } S \rightarrow I]$$

$$\xrightarrow[\text{rm}]{\Rightarrow} a^* (a + b00) \quad [\text{Using } I \rightarrow a]$$

3. Let G be the grammar. $P = \{S \rightarrow aB \mid bA, A \rightarrow a \mid aS \mid bAA, B \rightarrow b \mid bs \mid aBB\}$. For the string aabbbaaa, find LMD and RMD. (AU – Apr/May 2012, Nov/Dec 2010)

Solution:

Leftmost derivation	Rightmost derivation
$S \xrightarrow[\ell_m]{\Rightarrow} a\underline{B}$	$S \xrightarrow[\text{rm}]{\Rightarrow} aB$
$\xrightarrow[\ell_m]{\Rightarrow} aa\underline{B}B$	$\xrightarrow[\text{rm}]{\Rightarrow} aaB\underline{B}B$
$\xrightarrow[\ell_m]{\Rightarrow} aab\underline{B}B$	$\xrightarrow[\text{rm}]{\Rightarrow} aaBb\underline{S}$
$\xrightarrow[\ell_m]{\Rightarrow} aabb\underline{S}B$	$\xrightarrow[\text{rm}]{\Rightarrow} aaBbb\underline{A}B$
$\xrightarrow[\ell_m]{\Rightarrow} aabb\underline{B}A$	$\xrightarrow[\text{rm}]{\Rightarrow} aaBbbb\underline{A}A$
$\xrightarrow[\ell_m]{\Rightarrow} aabbbaAA$	$\xrightarrow[\text{rm}]{\Rightarrow} aaBbbb\underline{A}a$
$\xrightarrow[\ell_m]{\Rightarrow} aabbbaA\underline{A}$	$\xrightarrow[\text{rm}]{\Rightarrow} aa\underline{B}bbbbaa$
$\xrightarrow[\ell_m]{\Rightarrow} aabbbaaa$	$\xrightarrow[\text{rm}]{\Rightarrow} aabbbaaa$

4. Let the productions of grammar be

(AU – Apr/May 2007)

$$S \rightarrow 0B \mid 1A$$

$$A \rightarrow 0 \mid 0S \mid 1AA$$

$$B \rightarrow 1 \mid 1S \mid 0BB$$

For the string 0110, find RMD.

Solution:

$S \xrightarrow[\text{rm}]{\Rightarrow} 0B$	[Using $S \rightarrow 0B$]
$\xrightarrow[\text{rm}]{\Rightarrow} 01\underline{S}$	[Using $B \rightarrow 1S$]
$\xrightarrow[\text{rm}]{\Rightarrow} 011\underline{A}$	[Using $S \rightarrow 1A$]
$\xrightarrow[\text{rm}]{\Rightarrow} 0110$	[Using $A \rightarrow 0$]

PARSE TREES

(AU – May 2004)

- The representation of a derivation in the form of a tree is called a parse tree/derivation tree.
- The tree representation provides a clear view of the expansions / substitutions done on the variables.
- The tree representation also facilitates easy conversion / translation of the source code in to executable code, when used in compilers.

Formal definition and construction of parse tree

- A set of derivations substituted / applied to generate a string can be represented using a tree, called parse tree.
- Parse tree provides clear understanding of recursive function, grouping of symbols especially parenthesis balancing etc.

The parse trees can be constructed by using the following three conditions.

1. The **root node** of the tree being the start symbol and the **interior nodes** [non – leaf nodes] is represented by the variable, V of grammar G.
2. The **leaf nodes** are labeled by either the terminals T or by ϵ .
3. If the **interior node** is labeled V_1 where $V_1 \in V$, having its children, X_1, X_2, \dots, X_n are the production of V_1 which are represented from left to right.

Note:

- $V_1 \rightarrow \alpha$ where $V_1 \in V$ and $\alpha \in T$, then V_1 can have any number of children based on its production rules.
- If $V_1 \rightarrow \epsilon$, then ϵ should be the only child of V_1 .

Example

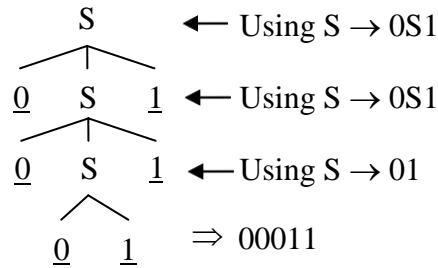
For the grammar $S \rightarrow 0S1|01$, generate the derivation of the string ‘000111’.

Solution:

The derivation is given by

$$\begin{aligned} S &\Rightarrow 0S1 & [\text{Using } S \rightarrow 0S1] \\ S &\Rightarrow 00S11 & [\text{Using } S \rightarrow 0S1] \\ S &\Rightarrow 000111 & [\text{Using } S \rightarrow 01] \end{aligned}$$

The parse tree is given by



Yield of a parse tree

- The terminals present at the leaf nodes are concatenated from left to right generate a string, derived from the grammar.
- The generated string is called the yield of a parse tree.

Example

- In the above mentioned parse tree, the terminal symbols in the leaf nodes, taken from left to right generate the string, „000111”.
- Thus 000111 is the yield of the parse tree.

Types of parse tree

- There are two types of parse tree. They are,
 - Leftmost derivation tree / left derivation tree
 - Rightmost derivation tree / right derivation tree

Leftmost derivation tree

- A derivation tree representing $A \Rightarrow \alpha$ is called a leftmost derivation tree if the production rule is applied only to the leftmost variable at every step.

Rightmost derivation tree

- A derivation / parse tree representing $A \Rightarrow \alpha$ is said to be a rightmost derivation tree if the production rule is applied only to the rightmost variable at every step.

Example

For the grammar given below give the parse tree for leftmost and rightmost derivation of the string ‘1001’. (AU – Apr/May 2006)

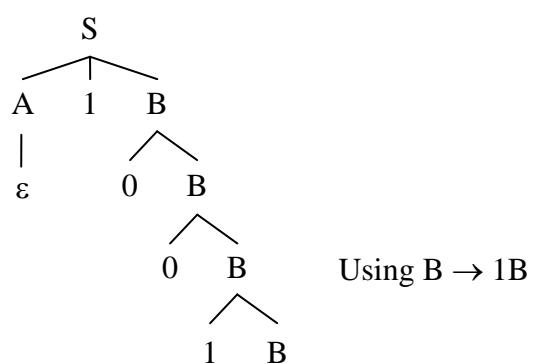
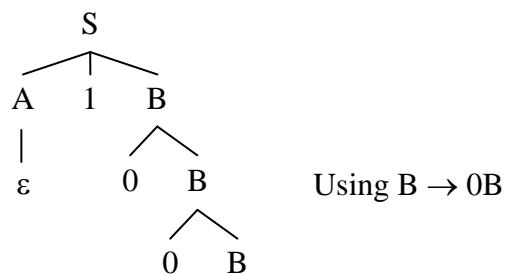
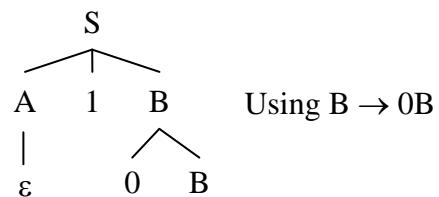
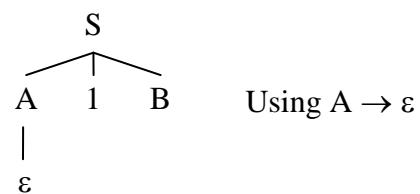
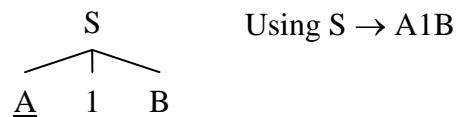
$$S \rightarrow A1B$$

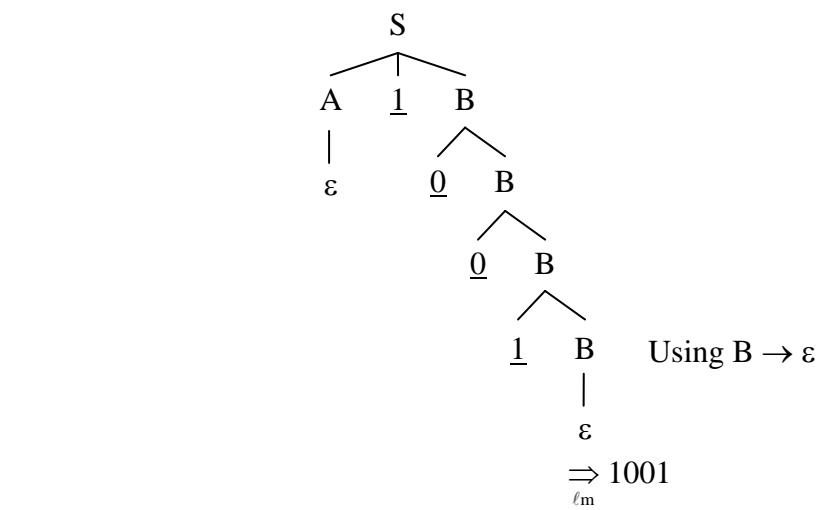
$$A \rightarrow 0A \mid \epsilon$$

$$B \rightarrow 0B \mid 1B \mid \epsilon$$

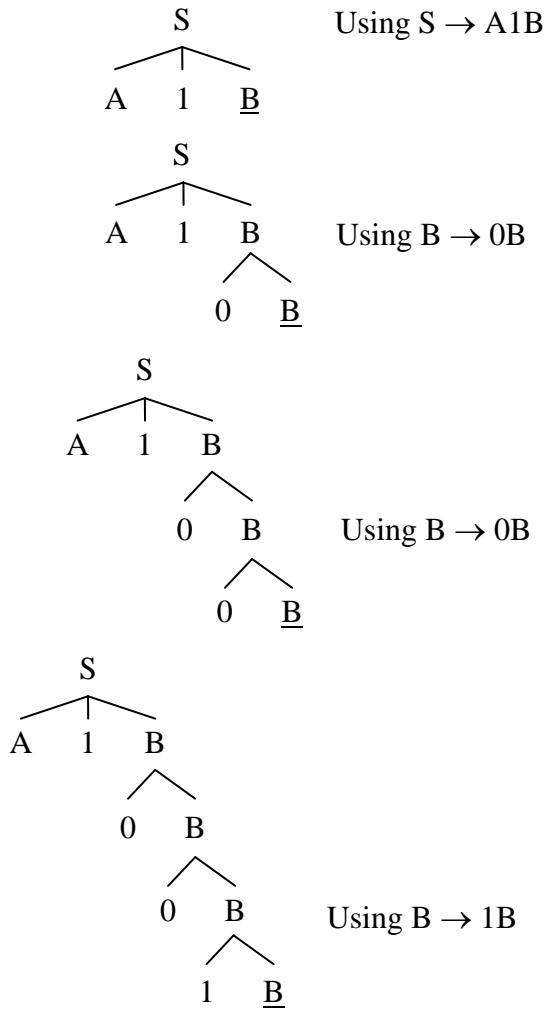
Solution:

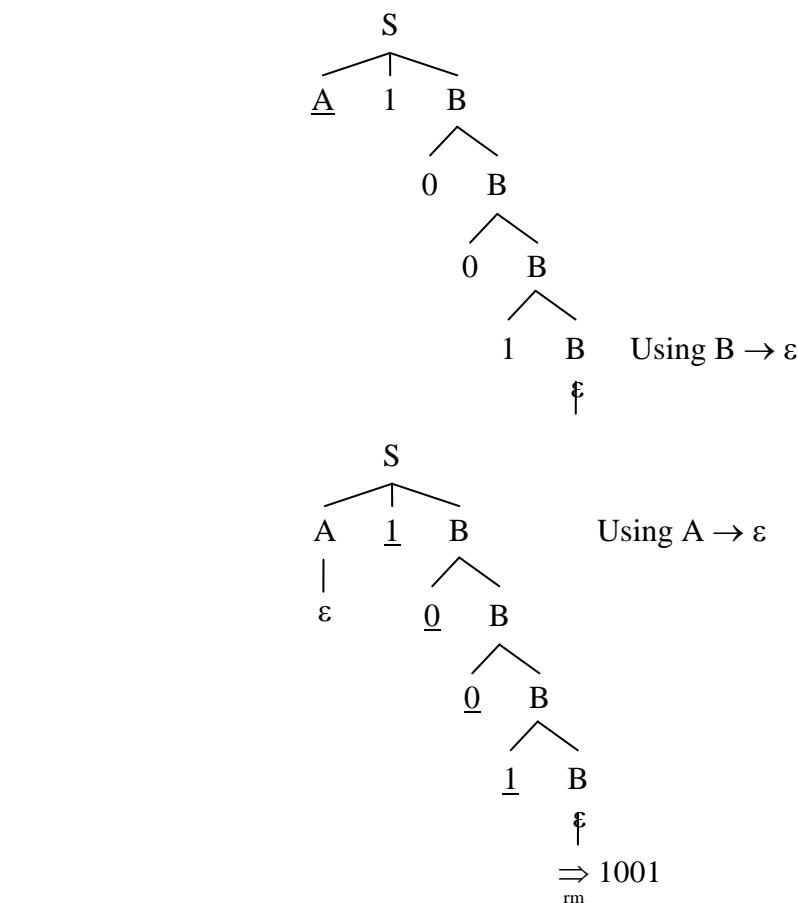
Leftmost derivative tree





Rightmost derivative tree



**Recursive production / inference**

(AU – Dec 2007)

- A production is said to be recursive if the left side variable occurs on the right hand side that substitutes the same production for „n“ number of times.
- This may lead to infinite language since the production rule is applied recursively.

Example:

$S \rightarrow aS$ \Rightarrow thus would lead to application of the same rule recursively as $S \rightarrow a\underline{S} \rightarrow aa\underline{S} \rightarrow aaa\underline{S} \rightarrow \dots$

PROBLEMS ON PARSE TREES

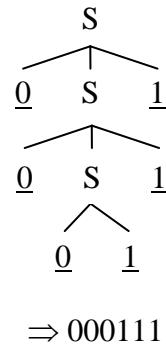
- 1. For the grammar, $S \rightarrow 0S1 \mid 01$, give the derivation of 000111.**

Solution:

Derivation in sentential form

$$\begin{aligned} S &\Rightarrow 0\underline{S}1 & [\text{Using } S \rightarrow 0S1] \\ &\Rightarrow 00\underline{S}11 & [\text{Using } S \rightarrow 0S1] \\ &\Rightarrow 000111 & [\text{Using } S \rightarrow 01] \end{aligned}$$

Parse tree



$$\Rightarrow 000111$$

- 2. Find whether $S = aabb$ is in $L = L(G)$ where G is given by $S \rightarrow XY; X \rightarrow YY \mid a; Y \rightarrow XY \mid b$.**

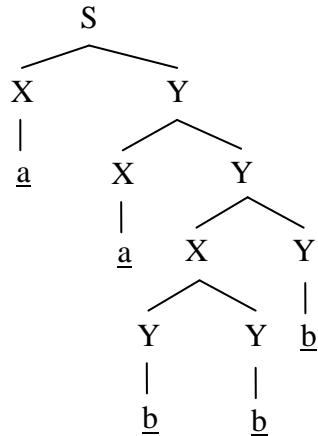
Solution:

Sentential form

$$\begin{aligned} S &\Rightarrow \underline{XY} & [\text{Using } S \rightarrow XY] \\ &\Rightarrow a\underline{Y} & [\text{Using } X \rightarrow a] \\ &\Rightarrow a\underline{XY} & [\text{Using } Y \rightarrow XY] \\ &\Rightarrow aa\underline{Y} & [\text{Using } X \rightarrow a] \\ &\Rightarrow aa\underline{XY} & [\text{Using } Y \rightarrow XY] \\ &\Rightarrow aa\underline{YY} & [\text{Using } X \rightarrow YY] \\ &\Rightarrow aab\underline{Y} & [\text{Using } Y \rightarrow b] \\ &\Rightarrow aabb\underline{Y} & [\text{Using } Y \rightarrow b] \\ &\Rightarrow aabb & [\text{Using } Y \rightarrow b] \end{aligned}$$

← →

Parse tree



$\Rightarrow aabb$

3. For the below given grammar, generate the derivation of $(a+b)^*a+b$.

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

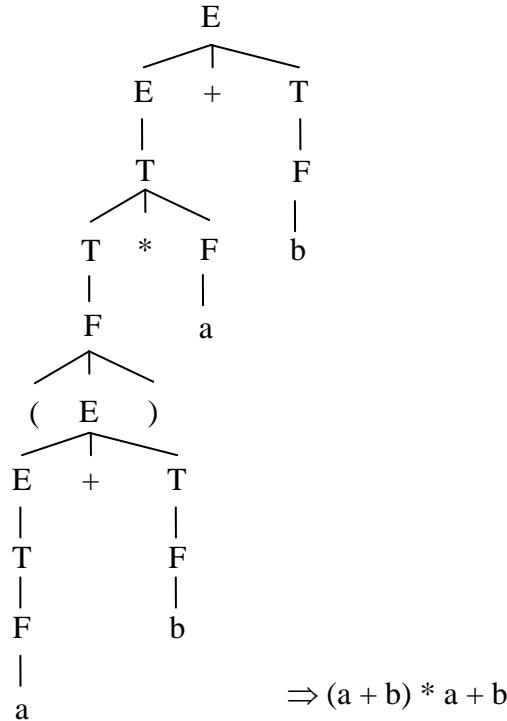
$$F \rightarrow (E) \mid a \mid b$$

Solution:

Sentential form

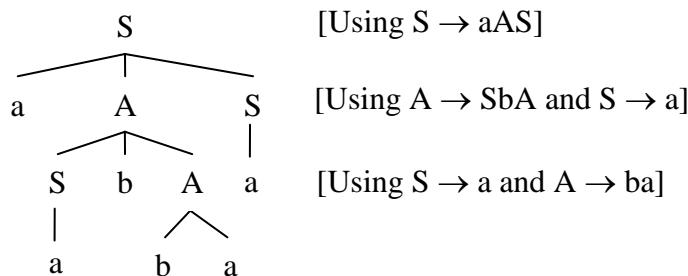
$E \Rightarrow \underline{E} + T$	[Using $E \rightarrow E+T$]
$\Rightarrow \underline{T} + T$	[Using $E \rightarrow T$]
$\Rightarrow \underline{T} * F + T$	[Using $T \rightarrow T * F$]
$\Rightarrow \underline{F} * F + T$	[Using $T \rightarrow F$]
$\Rightarrow (\underline{E}) * F + T$	[Using $F \rightarrow (E)$]
$\Rightarrow (\underline{E} + T) * F + T$	[Using $E \rightarrow E+T$]
$\Rightarrow (\underline{T} + T) * F + T$	[Using $E \rightarrow T$]
$\Rightarrow (\underline{F} + T) * F + T$	[Using $T \rightarrow F$]
$\Rightarrow (a + \underline{T}) * F + T$	[Using $F \rightarrow a$]
$\Rightarrow (a + \underline{F}) * F + T$	[Using $T \rightarrow F$]
$\Rightarrow (a + b) * \underline{F} + T$	[Using $F \rightarrow b$]
$\Rightarrow (a + b) * a + \underline{T}$	[Using $F \rightarrow a$]
$\Rightarrow (a + b) * a + \underline{F}$	[Using $T \rightarrow F$]

$$\Rightarrow (a + b) * a + b \quad [\text{Using } F \rightarrow b]$$



4. Construct a parse tree for $S \Rightarrow aabbba$, given the productions: $S \rightarrow aAS|a$; $A \rightarrow SbA|SS|ba$. (AU – Dec 2003, Dec 2009)

Solution:

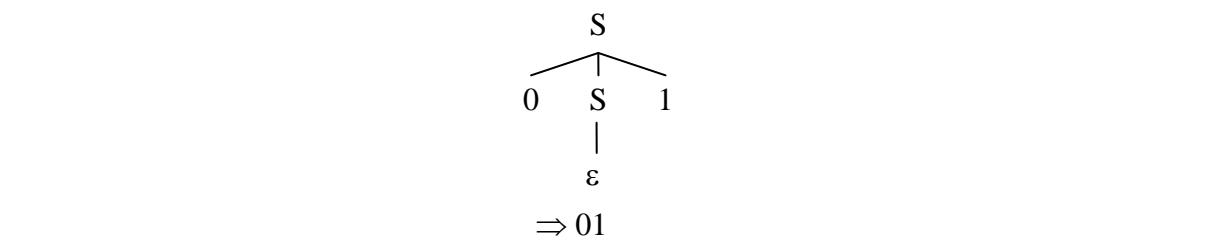


5. Find $L(G)$ where $G = (\{S\}, \{0, 1\}, \{S \rightarrow 0S1, S \rightarrow \epsilon\}, S)$. (AU – Apr/May 2004)

Solution:

$$1. S \Rightarrow 0\underline{S}1$$

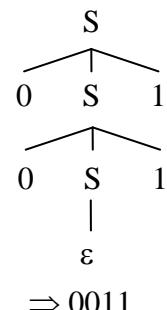
$$\Rightarrow 01 \quad [\text{Since } S \rightarrow \epsilon]$$



2. $S \Rightarrow 0\underline{S}1$

$\Rightarrow 00\underline{S}11$ [Since $S \rightarrow 0S1$]

$\Rightarrow 0011$ [Since $S \rightarrow \epsilon$]

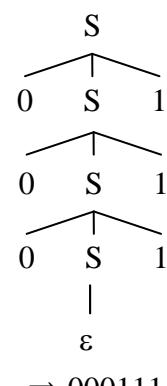


3. $S \Rightarrow 0\underline{S}1$ [Since $S \rightarrow 0S1$]

$\Rightarrow 00\underline{S}11$ [Using $S \rightarrow 0S1$]

$\Rightarrow 000\underline{S}111$ [Using $S \rightarrow 0S1$]

$\Rightarrow 000111$ [Using $S \rightarrow \epsilon$]



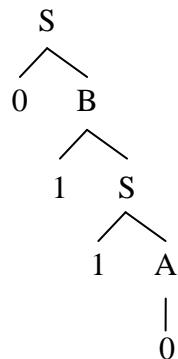
$\Rightarrow 000111$

$\therefore L(G) = \{0^n 1^n \mid n \geq 0\}$

6. Let G be given as, $P = \{S \rightarrow 0B|1A, A \rightarrow 0|0S|1AA, B \rightarrow 1|1S|0BB\}$. Find the rightmost derivation for '0110'. (AU – Apr/May 2007)

Solution:

$$\begin{aligned}
 S &\xrightarrow{\text{rm}} 0\underline{B} & [\text{Using } S \rightarrow 0B] \\
 &\xrightarrow{\text{rm}} 01\underline{S} & [\text{Using } B \rightarrow 1S] \\
 &\xrightarrow{\text{rm}} 011\underline{A} & [\text{Using } S \rightarrow 1A] \\
 &\xrightarrow{\text{rm}} 0110 & [\text{Using } A \rightarrow 0]
 \end{aligned}$$



7. Find the leftmost derivation for $S = aaabbabbba$.

Given the productions

(AU – June 2007, May 2008, June 2013)

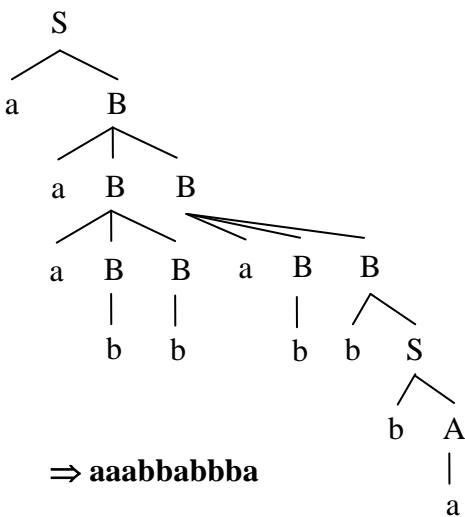
$$\begin{aligned}
 S &\rightarrow aB|bA \\
 A &\rightarrow aS|bAA|a \\
 B &\rightarrow bS|aBB|b
 \end{aligned}$$

Solution:

Leftmost derivation

$$\begin{aligned}
 S &\xrightarrow{\ell_m} a\underline{B} & [\text{Using } S \rightarrow aB] \\
 &\xrightarrow{\ell_m} aa\underline{B}B & [\text{Using } B \rightarrow aBB] \\
 &\xrightarrow{\ell_m} aaa\underline{B}BB & [\text{Using } B \rightarrow aBB] \\
 &\xrightarrow{\ell_m} aaab\underline{B}B & [\text{Using } B \rightarrow b] \\
 &\xrightarrow{\ell_m} aaabb\underline{B} & [\text{Using } B \rightarrow b]
 \end{aligned}$$

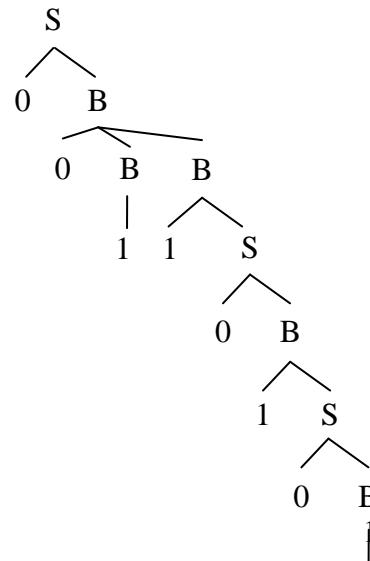
$\Rightarrow aaabba \underline{BB}$	[Using $B \rightarrow aBB$]
$\Rightarrow aaabbab \underline{B}$	[Using $B \rightarrow b$]
$\Rightarrow aaabbabb \underline{s}$	[Using $B \rightarrow bS$]
$\Rightarrow aaabbabb \underline{bA}$	[Using $S \rightarrow ba$]
$\Rightarrow aaabbabbba$	[Using $A \rightarrow a$]



8. Let G be the grammar, $S \rightarrow 0B|1A$, $A \rightarrow 0|0S|1AA$, $B \rightarrow 1|1S|0BB$. For $S=00110101$, find the leftmost derivation. (AU – May 2004)

Solution:

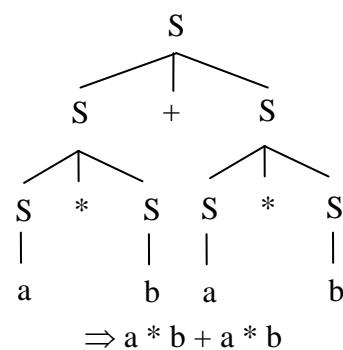
$S \Rightarrow 0 \underline{B}$	[Using $S \rightarrow 0B$]
$\Rightarrow 00 \underline{BB}$	[Using $B \rightarrow 0BB$]
$\Rightarrow 001 \underline{B}$	[Using $B \rightarrow 1$]
$\Rightarrow 0011 \underline{S}$	[Using $B \rightarrow 1S$]
$\Rightarrow 00110 \underline{B}$	[Using $S \rightarrow 0B$]
$\Rightarrow 001101 \underline{S}$	[Using $B \rightarrow 1S$]
$\Rightarrow 0011010 \underline{B}$	[Using $S \rightarrow 0B$]
$\Rightarrow 00110101$	[Using $B \rightarrow 1$]



9. Find the derivation tree of $a^*b + a^*b$ where G is given by $S \rightarrow S + S \mid S * S \mid a \mid b$.
 (AU – June 2007)

Solution:

$$\begin{aligned}
 S &\Rightarrow \underline{S} + S && [\text{Using } S \rightarrow S + S] \\
 &\Rightarrow \underline{S} * S + S && [\text{Using } S \rightarrow S * S] \\
 &\Rightarrow a * \underline{S} + S && [\text{Using } S \rightarrow a] \\
 &\Rightarrow a * b + \underline{S} && [\text{Using } S \rightarrow b] \\
 &\Rightarrow a * b + \underline{S} * S && [\text{Using } S \rightarrow S * S] \\
 &\Rightarrow a * b + a * \underline{S} && [\text{Using } S \rightarrow a] \\
 &\Rightarrow a * b + a * b && [\text{Using } S \rightarrow b]
 \end{aligned}$$



10. Construct a CFG for

(AU – June 2009)

$$(a) L(G) = \{a^m b^n \mid m \neq n; m, n > 0\}$$

$$(b) L(G) = \{a^n b a^n \mid n \geq 1\}$$

Solution:

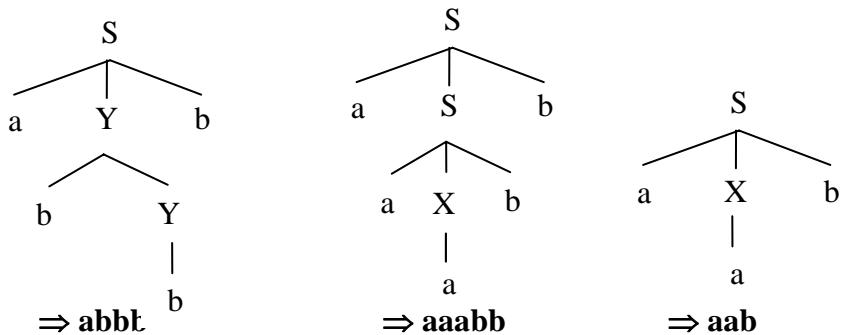
(a) The string generated by the grammar are,

$$S \Rightarrow \{aab, abb, aabbb, aaab, abbbb, \text{etc}\}$$

The production rules are given as,

$$P \Rightarrow \begin{cases} S \rightarrow aSb \mid aXb \mid aYb \\ X \rightarrow aX \mid a \\ Y \rightarrow bY \mid b \end{cases}$$

Example



Where, $V = \{S, X, Y\}$, $T = \{a, b\}$, $S = \{S\}$

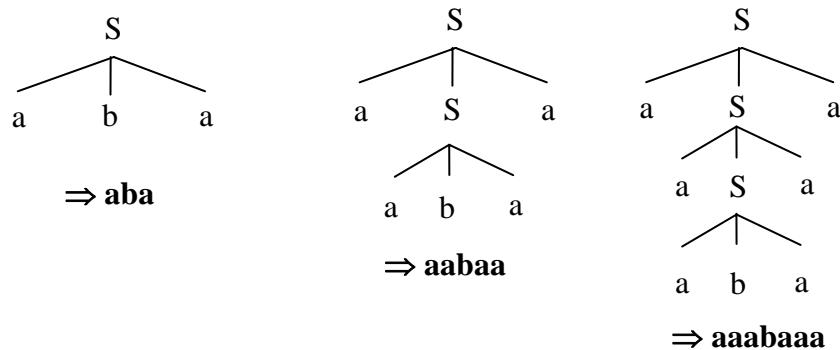
(b) The string generated by the grammar are,

$$S \Rightarrow \{aba, aabaa, aaabaaa, \text{etc.,}\}$$

The production rules are given as,

$$P \Rightarrow \begin{cases} S \rightarrow aSa \\ S \rightarrow aba \end{cases}$$

Where $V = \{S\}$, $T = \{a, b\}$, $S = \{S\}$



AMBIGUITY

Ambiguous grammars

(AU – June 2009, June 2013, Dec 2012, June 2014)

- A grammar, G is said to be ambiguous if there exists two different parse trees for at least one string „ ω “ where $\omega \in T^*$, each parse tree with the same symbol.
- If each string ω has at most one parse tree in the grammar, then it is unambiguous grammar.

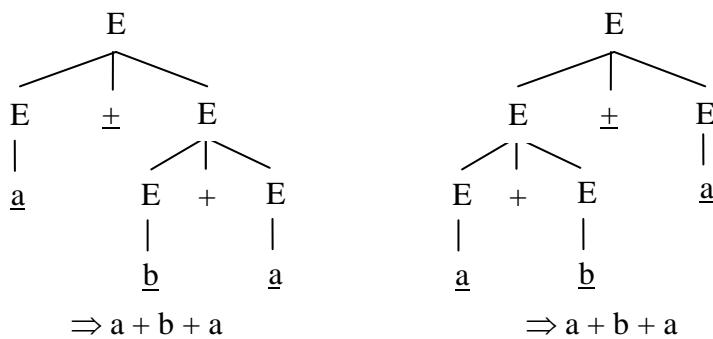
Example: Ambiguous grammar

Consider the grammar given below, prove that it is ambiguous.

$$E \rightarrow E + E \mid a \mid b$$

Solution:

Consider $a + b + a$, then the parse trees are derived as,



- Since there are two different parse trees with same start symbol, leads to the same string, $\omega \in T^*$, the grammar is ambiguous.

Removing ambiguity from grammar

- The only possibility of removing ambiguity from grammar is to introduce one or more different variables.

Example

Consider the above mentioned example that contains ambiguity. Remove the ambiguity from the grammar, $E \rightarrow E + E \mid a \mid b$.

Solution:

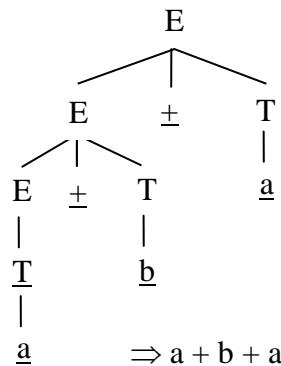
Given grammar, $E \rightarrow E + E \mid a \mid b$

The grammar can be re-written by introducing a new variable T as,

$$E \rightarrow E + T \mid T$$

$$T \rightarrow a \mid b$$

Thus the parse tree for $a + b + a$ will be



Thus there is no standard algorithm for removing ambiguity, other than rewriting the grammar by some new variables.

Inherent ambiguity

- A context free language, L is internally ambiguous if all its grammars are ambiguous.

Example

$$\begin{aligned} \text{Let } L &= \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \\ &= \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\} \end{aligned}$$

Here L consists of

- Equal number of a and b and equal number of c and d.
- Equal number of a and d and equal number of b and c.

The production can be defined as,

$$S \rightarrow AB \mid C$$

$$A \rightarrow aAb \mid ab \quad \Rightarrow \text{equal no of a and b}$$

$B \rightarrow cBd \mid cd \Rightarrow$ equal no of c and d

$C \rightarrow aCd \mid aDd \Rightarrow$ equal no of a and d

$D \rightarrow bDc \mid bc \Rightarrow$ equal no of b and c

Consider the string “aabbcddd” | n = m = 2, then there can be two possible leftmost derivations as,

1) LMD-1

$$S \xrightarrow[\ell_m]{} \underline{AB} \quad [S \rightarrow AB]$$

$$\xrightarrow[\ell_m]{} a\underline{Ab}B \quad [A \rightarrow aAb]$$

$$\xrightarrow[\ell_m]{} aabb\underline{B} \quad [A \rightarrow ab]$$

$$\xrightarrow[\ell_m]{} aabbc\underline{B}d \quad [B \rightarrow cBd]$$

$$\xrightarrow[\ell_m]{} aabbccdd \quad [B \rightarrow cd]$$

2) LMD-2

$$S \xrightarrow[\ell_m]{} \underline{C} \quad [S \rightarrow C]$$

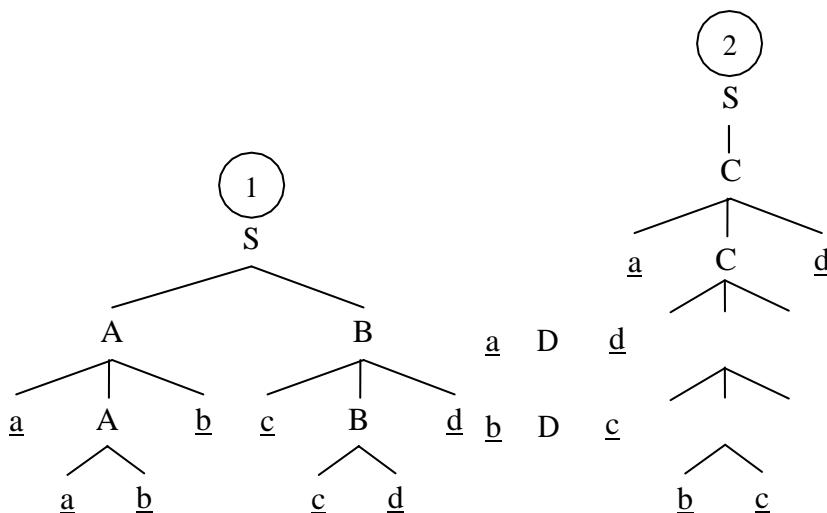
$$\xrightarrow[\ell_m]{} a\underline{Cd} \quad [C \rightarrow aCd]$$

$$\xrightarrow[\ell_m]{} aa\underline{D}dd \quad [C \rightarrow aDd]$$

$$\xrightarrow[\ell_m]{} aab\underline{D}cdd \quad [D \rightarrow bDc]$$

$$\xrightarrow[\ell_m]{} aabbccdd \quad [D \rightarrow bc]$$

The parse tree of LMD-1 and LMD-2 are given as



PROBLEMS ON AMBIGUITY IN GRAMMARS

1. Consider the grammar,

$$E \rightarrow E + E \mid E * E \mid (E) \mid I$$

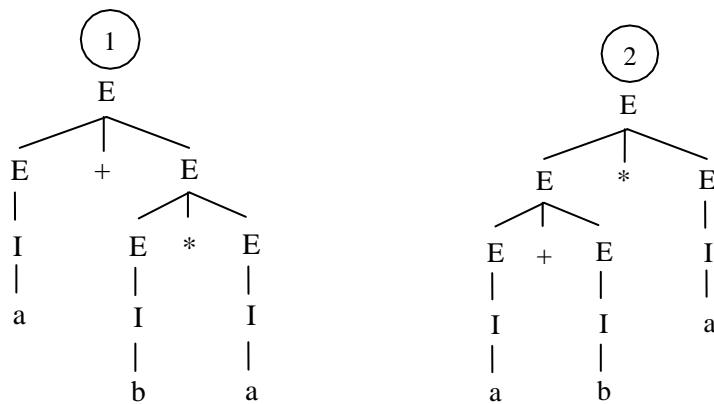
$$I \rightarrow a + b$$

(a) Show that the grammar is ambiguous

(b) Remove the ambiguity.

Solution:

(a) Ambiguity in grammar



From ① and ② we have generated two different parse trees for the same string, $a+b^*a$ using the same production.

Thus, the grammar is ambiguous.

(b) Removing ambiguity

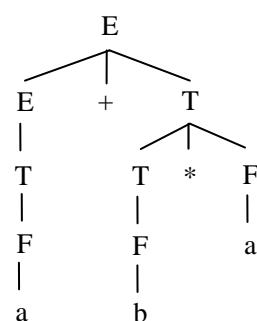
Introducing new variable T to remove ambiguity, we get the production rules as

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid a \mid b$$

The only parse tree for $a + b^* a$ using the modified grammar is given as



$$\Rightarrow a + b^* a$$

2. Test whether the following grammars are ambiguous. (AU – Dec 2006, Dec 2013)

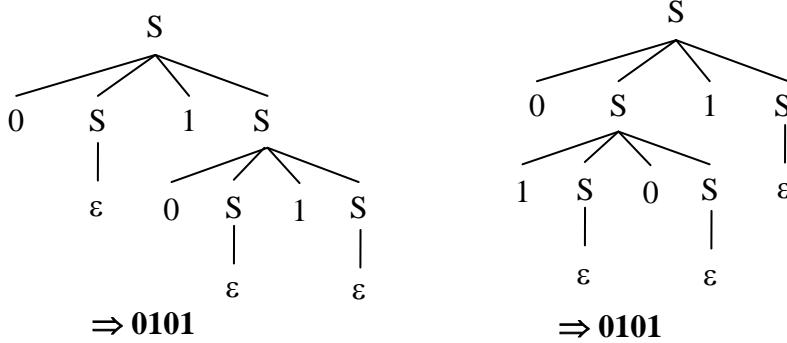
$$(a) S \rightarrow 0S1S \mid 1S0S \mid \epsilon$$

$$(b) S \rightarrow AA$$

$$A \rightarrow aAb \mid bAa \mid \epsilon$$

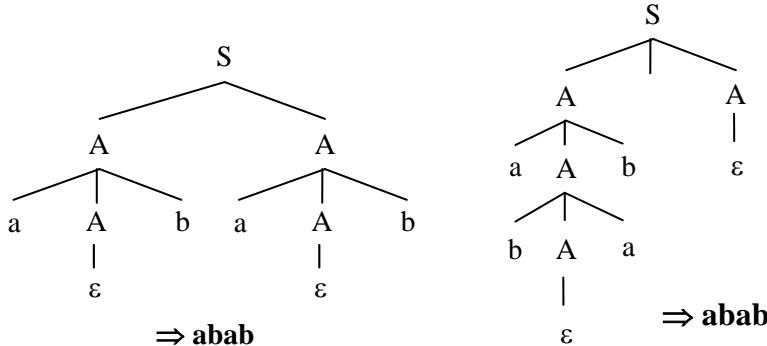
Solution:

(a) Consider $S = 0101$, the parse tree for LMD and RMD are given as,



Since there are two different parse trees for the same string, the grammar is ambiguous.

(b) Consider $S = abab$, the parse trees are,



There are two different parse trees for the same string $S = abab$. Hence the grammar is ambiguous.

3. Show that $id + id * id$ can be generated by two distinct leftmost derivations in the grammar, $E \rightarrow E + E \mid E * E \mid (E) \mid id$. (AU – May 2005, Dec 2005, Dec 2010)

Solution:

Leftmost derivation – 1

$$E \xrightarrow{\ell_m} E+E$$

$$\xrightarrow{\ell_m} id + E$$

- $\Rightarrow_{\ell_m} id + E * E$
- $\Rightarrow_{\ell_m} id + id * E$
- $\Rightarrow_{\ell_m} id + id * id$

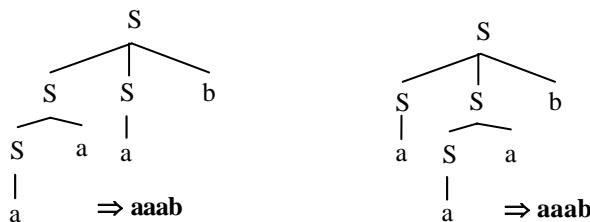
Leftmost derivation – 2

$$\begin{aligned}
 E &\Rightarrow E * E \\
 &\quad \ell_m \\
 &\Rightarrow E + E * E \\
 &\quad \ell_m \\
 &\Rightarrow id + E * E \\
 &\quad \ell_m \\
 &\Rightarrow id + id * E \\
 &\quad \ell_m \\
 &\Rightarrow id + id * id
 \end{aligned}$$

4. Show that $S \rightarrow a \mid Sa \mid bSS \mid SSb \mid SbS$ is ambiguous. (AU – Dec 2007, Dec 2008)

Solution:

Consider the string $S = \text{aaab}$.



From the above parse trees, we have proved that the grammar is ambiguous.

5. Show that the given grammar is ambiguous

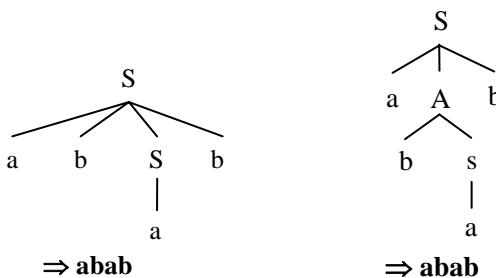
(AU – June 2007)

$$S \rightarrow a \mid abSb \mid aAb$$

$$A \rightarrow bS \mid aAAb$$

Solution:

Consider the string $S = abab$.



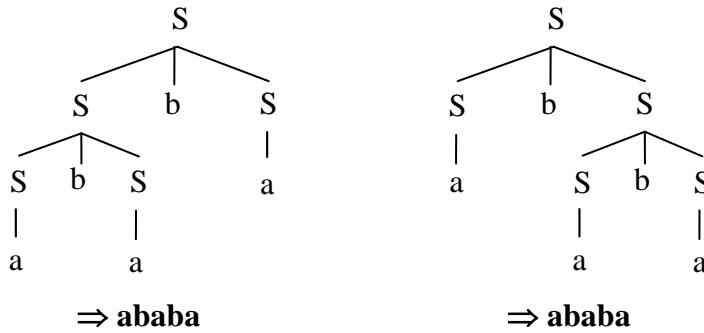
From the above parse trees, we have proved that the given grammar is ambiguous.

6. If G is the grammar defined by, $S \rightarrow SbS|a$. Show that G is ambiguous.

(AU – May 2004, Dec 2009, June 2004)

Solution:

Consider the string $S = ababa$.



From the above two different parse trees, we conclude that the grammar is ambiguous.

7. Is the grammar $S \rightarrow SS | (S) | S(S)S | \epsilon$ is ambiguous?

(AU – Nov/Dec 2011)

Solution:

LMD

$$\begin{aligned} S &\xrightarrow{\ell_m} \underline{SS} & [\text{Using } S \rightarrow SS] \\ &\xrightarrow{\ell_m} (\underline{S}) S & [\text{Using } S \rightarrow (S)] \\ &\xrightarrow{\ell_m} () \underline{S} & [\text{Using } S \rightarrow \epsilon] \\ &\xrightarrow{\ell_m} () & [\text{Using } S \rightarrow \epsilon] \end{aligned}$$

RMD

$$\begin{aligned} S &\xrightarrow{\text{rm}} S(\underline{S}) \underline{S} \\ &\xrightarrow{\text{rm}} S(\underline{S}) & [\text{Using } S \rightarrow \epsilon] \\ &\xrightarrow{\text{rm}} \underline{S}() & [\text{Using } S \rightarrow \epsilon] \\ &\xrightarrow{\text{rm}} () & [\text{Using } S \rightarrow \epsilon] \end{aligned}$$

Since the grammar generates two different derivations for the same string, it is ambiguous.

RELATIONSHIP BETWEEN DERIVATION AND DERIVATION TREE

From Trees To Derivations

Theorem

(AU – Dec 2003, Dec 2004, May 2005, Dec 2005)

Let the grammar, $G = (V.T, P, S)$ be context free. Then $S \xrightarrow[G]{*} \alpha$ if and only if there is a derivation tree in grammar G , that generates the string „ α “.

Proof

Let S be a variable, where $S \in V$, then $S \xrightarrow[G]{*} \alpha$ if and only if there is a parse tree called S -tree, starting from the root node (S) to generate the string „ α “.

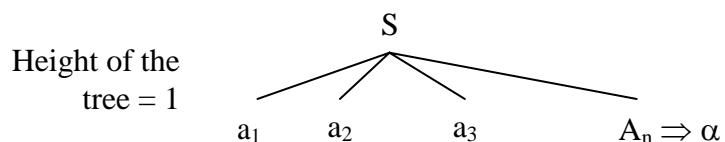
The problem can be easily proved by the principle of mathematical induction.

Basis of Induction

Consider the lowest possible input value and prove the theorem given.

So we assume that there is only one interior node [height = 1] which forms the root node, that yields the string a , by deriving the leaf nodes of S as $S \rightarrow a_1 a_2 a_3 \dots a_n$.

The derivation tree for the generation of $S \xrightarrow{*} \alpha$ is given as



Thus $S \Rightarrow a_1 a_2 \dots a_n \Rightarrow \alpha$ is the input string generated from S .

Inductive Step

Assume that the theorem is true for „ n^{th} “ input data and we need to prove the same for $n = n+1$ data.

Hence we assume that the derivation tree can be drawn for $(n - 1)$ interior nodes.

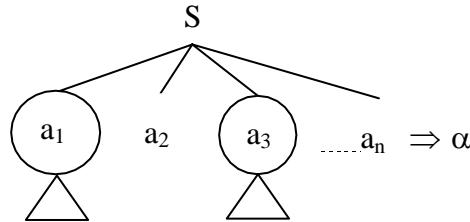
And we need to prove that the derivation tree can be drawn for „ n “ interior nodes to derive „ α “ from S .

Here, we need to analyze that certain nodes shall be leaf nodes, whereas others can be interior nodes.

Hence if a node $X_i \in T$, then $\alpha_i = X_i$

If the node $X_i \in V$, then $X_i \xrightarrow[\ell_m]^* \alpha_i$ is in G .

The derivation tree is given as, [By inductive hypothesis]



where,

$$a_1, a_3 \in V$$

$$a_2, a_n \in T$$

Thus, $S \Rightarrow a_1 a_2 \dots a_n \Rightarrow \alpha$ can be obtained.

Hence the theorem is proved by mathematical induction.

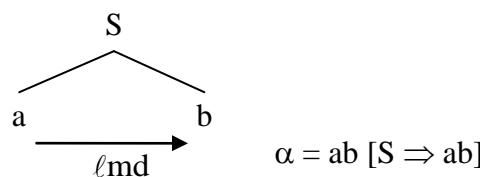
Example

Let $G = (V, T, P, S)$ be a CFG with productions, $S \rightarrow aSb \mid ab$

Let us consider the basis of induction according to it, the lowest possible integer height of the tree should be considered.

Let the height of the tree = 1

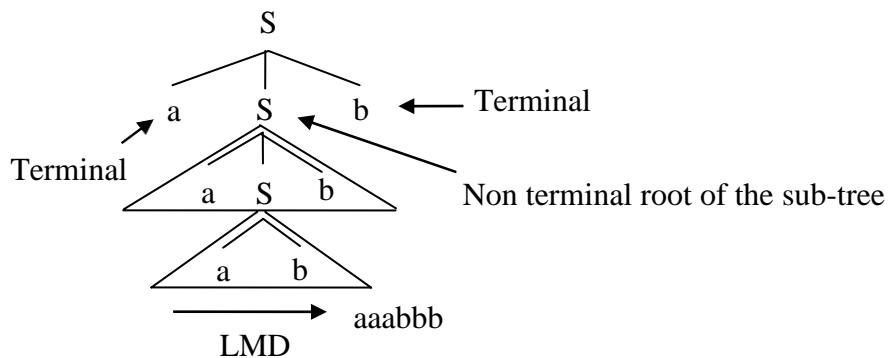
∴ The grammar, S yields only one possible string $S \rightarrow ab$ which is given by



According to the inductive step let us assume that we can generate a derivation tree of height $(n - 1)$ and we need to prove that it is true for the tree of height, n .

Let $n - 1 = 2$

$$\begin{aligned} S &\Rightarrow aSb \\ &\xrightarrow[\ell_m]{} aaSbb \\ &\xrightarrow[\ell_m]{} aaabbba \end{aligned}$$

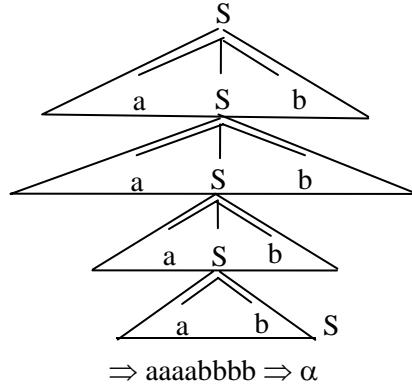


This is assumed.

When $n = 3$ $[(n - 1) + 1]$ the derivation becomes,

$$\begin{aligned}
 S &\Rightarrow aSb \\
 &\stackrel{\ell_m}{\Rightarrow} aaSbb \\
 &\stackrel{\ell_m}{\Rightarrow} aaaSbbb \\
 &\stackrel{\ell_m}{\Rightarrow} aaaabb \Rightarrow \alpha
 \end{aligned}$$

$$\therefore S \Rightarrow \alpha$$



Thus if the theorem is true for a tree of height $(n - 1)$, then it is true for height n .

From Derivation To Recursive Inferences

Theorem

Let $G = (V, T, P, S)$ be a CFG and if there is a derivation $A \xrightarrow[\ell_m]{*} \alpha$, where α is in T^* .

Then the recursive inference procedure applied to G determines that α is in the language of non-terminal „ A “.

Proof

We shall prove the theorem by the principle of mathematical induction.

Basis of induction

Let us consider the lowest possible input of the grammar.

Let $A \Rightarrow \alpha$, then there must be a production $A \rightarrow \alpha$ in G. then we can infer that α is in the language of variable A.

Inductive step

Assume that the theorem holds for the derivation has $(n-1)$ number of steps and we need to prove that it holds for „n“ no. of steps.

Let the derivation be stated as,

$$A \Rightarrow X_1 X_2 X_3 \dots X_k$$

$$\stackrel{*}{\Rightarrow} \alpha$$

Then α can be broken as

$$\alpha = \alpha_1 \alpha_2 \dots \alpha_k$$

where,

$$X_i \stackrel{*}{\Rightarrow} \alpha_i$$

The derivation $X \Rightarrow \omega_i$ can take at most „n-1“ number of steps to generate.

Now we have a production,

$$A \rightarrow X_1 X_2 \dots X_k$$

Infer α_i to be in the language of X_i

$\therefore \alpha_1 \alpha_2 \dots \alpha_k$ is inferred as in the language of A.

Leftmost derivation of a tree yields string**Theorem**

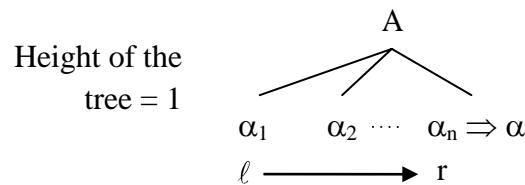
Let $G = (V, T, P, S)$ be a CFG. Suppose there is a parse tree with root labeled by $A \mid A \in V$ and with yield α where $\alpha \in T^*$. Then there is a leftmost derivation $A \xrightarrow{\ell_m} \alpha$ in G.

Proof

The theorem is proved by mathematical induction on the height of the parse tree.

Basis of induction

Let the height of the tree be 1, the tree looks like,



Here the root is labeled A, $A \in V$ and the children are the leaf nodes which are read from the left to right to generate α .

Thus $A \xrightarrow{\ell_m} \alpha$ is proved.

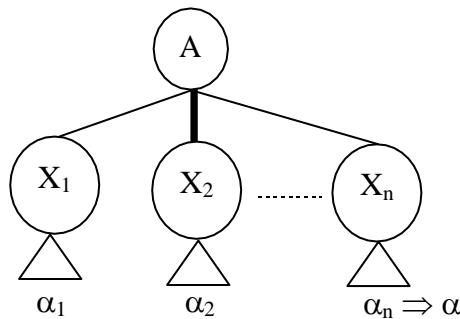
Inductive step

We assume the height of the tree as $n \mid n > 1$, then $A \xrightarrow{\ell_m} \alpha^*$ stands.

We need to prove the same for $n = n+1$.

There are two issues:

The child of A can be terminal / variable as given



1. If $A_i \in T$, then $X_i \Rightarrow \alpha_i$ [$\omega_i \rightarrow \text{terminal}$]
2. If $A_i \in V$, then there must be some leftmost derivation, $A_i \xrightarrow{*} \alpha_i$

For n nodes, $\alpha_i = \alpha_1 \alpha_2 \alpha_3 \dots \alpha_n$

$$A \xrightarrow[\ell_m]^* A_1 A_2 A_3 \dots A_n \quad (\text{for } i = 1, 2, 3, \dots, n)$$

$$\xrightarrow[\ell_m]^* \alpha_1 \alpha_2 \dots \alpha_{i-1} A_i A_{i+1} A_{i+2} \dots A_n$$

If A_i is a terminal, then

$$A_i \xrightarrow[\ell_m]^* \alpha_1 \alpha_2 \dots \alpha_i A_{i+1} A_{i+2} \dots A_n$$

If A_i is a variable, then

$$A \xrightarrow[\ell_m]{} \beta_1 \xrightarrow[\ell_m]{} \beta_2 \xrightarrow[\ell_m]{} \dots \xrightarrow[\ell_m]{} \alpha_i$$

and

$$A \Rightarrow \alpha_1 \alpha_2 \dots \alpha_{i-1} A_i A_{i+1} \dots A_n$$

$$\Rightarrow \alpha_1 \alpha_2, \dots, \beta_1$$

$$A_i A_{i+1} \dots A_n \xrightarrow[\ell_m]^* \alpha_1 \alpha_2 \beta_2 A_i$$

$$A_{i+1} \dots A_n \xrightarrow[\ell_m]^* \alpha_1 \alpha_2, \alpha_3, \dots, \alpha_i A_{i+1} A_{i+2} \dots A_n$$

Thus,

$$A \xrightarrow[\ell_m]^* \alpha_1 \alpha_2, \dots, \alpha_i A_{i+1} A_{i+2} \dots A_n$$

$$\xrightarrow[\ell_m]^* \Theta$$

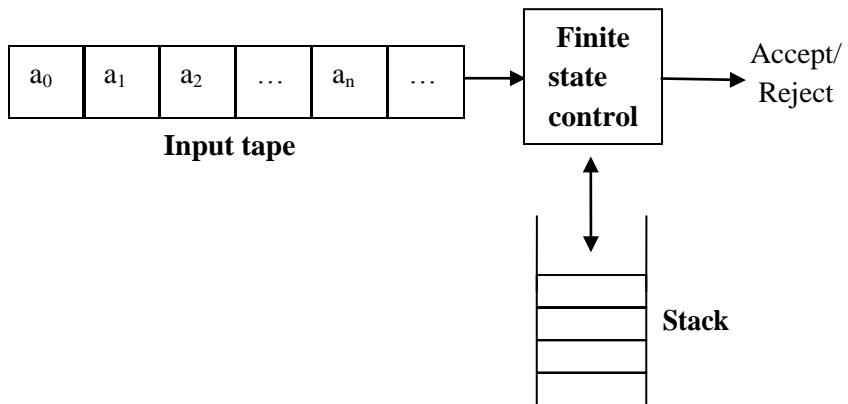
Thus the theorem has been proved.

PUSHDOWN AUTOMATA

- A Push down automata shortly called as a PDA is a ϵ -NFA with stack, which is used to define regular languages.
- Any grammar can be converted into a PDA and vice-versa. [CFL can be recognized by PDA].
- The stack in the PDA provides memory to store intermediate states and results, which increases the capability of the automaton.

- Due to the availability of stack, the PDA becomes more powerful than a finite automaton.

Notion of a PDA



- The input tape contains the input symbols that are to be processed by the PDA.
- The input symbols are defined by the alphabet (Σ) of the language.
- The finite state control reads one input at a time from the input tape.
- Based on the symbol on the top of the stack and with the input symbol,
 - Transition is made to the current state
(or)
 - Transition is made to the next state
(or)
 - Performs spontaneous transition using „ ϵ “ as its input at some times. (no input is read).
- The transition of a PDA thus, depends on
 - Current state
 - Current input / ϵ .
 - Top symbol of the stack.
- The stack stores a set of stack symbols generated from the input tape using the operations
 - PUSH – Inserts an element into the stack
 - POP – Deletes an element out of the stack
 - CHECK EMPTY – checks if the stack is empty
 - NOP – performs No operation

- So, based on the inputs and stack symbols, transitions are made.
- The finite state control performs the state changes.
- If the final state is reached, the PDA accepts the input symbol.
- After recognizing all the inputs, if the final state is not reached, it means that the PDA rejects the input.

Definition of Pushdown Automata

- A PDA can be formally defined by 7-tuples.
- The specification of a PDA, P is given by,

$$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F) \quad \text{where,}$$

$Q \rightarrow$ Finite set of states

$\Sigma \rightarrow$ Finite set of input symbols [alphabet]

$\Gamma \rightarrow$ Finite set of stack symbols

$\delta \rightarrow$ Transition function

$\delta: Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow Q \times \Gamma^*$ defined as, $\delta(q, a, x) = (q^1, x^1)$

where,

$q, q^1 \rightarrow$ states in Q

$a \rightarrow$ input symbol in Σ or ϵ

$x, x^1 \rightarrow$ stack symbol in Γ

$q_0 \rightarrow$ Initial/ start state [$q_0 \in Q$]

$z_0 \rightarrow$ Initial start symbol [$z_0 \in \Gamma$]

$F \rightarrow$ finite set of final/ accepting states [$F \subseteq Q$]

Informal definition of PDA

- A PDA is a non-deterministic FA with stack. The stack is to store the string taken from the input tape.
- The PDA accepts / rejects the input based on the transitions / moves made by the finite state system

MOVES / TRANSITIONS

A transitions is given by,

$$Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow Q \times \Gamma^*$$

Here, $Q \times (\Sigma \cup \epsilon) \times \Gamma$ implies that a transition is based on,

- Current state [$q \in Q$]
- Next input [$\Sigma \cup \epsilon$]
- Stack symbol [top-most element of the stack] [Γ]

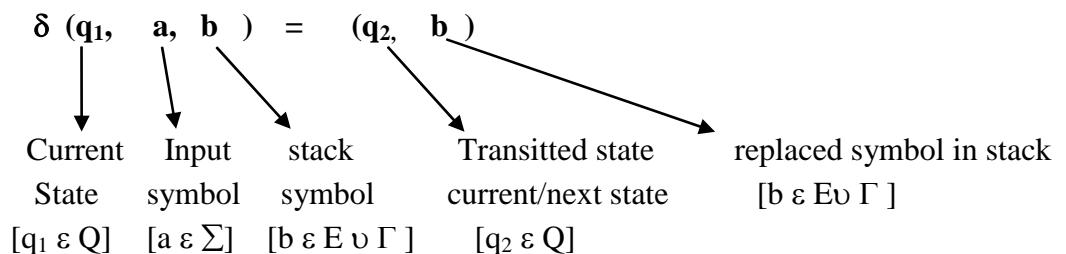
and $Q \times \Gamma^*$ implies,

- The next state reached after transition [$q' \in Q$]
- $q' \rightarrow$ current /another state defined in Q
- Stack symbol remaining after [Γ^*]
 - PUSH
 - POP
 - NOP

Transitions with stack operations

1) Read input with no-operation on stack

The transition that reads an input from the input tape , but performs no-operations on the stack is given as ,



Since, there is no-operation performed on stack, the stack symbols are neither added up / deleted off.

Thus the transition becomes (q_2, b) since the θ move does n't modify the stack.

2) Push operation on stack

Consider that the input = „a“ is pushed on to the stack. Then the transition becomes,

$\delta(q_1, a, b) = (q_2, a b)$, „a“ is accepted by the stack.

Where,

$q_1, q_2 \rightarrow$ current and next state [$q_1, q_2 \in Q$]

$A \rightarrow$ input symbol to be processed [$a \in \Sigma$]

$B \rightarrow$ stack symbol [$b \in \Sigma \cup \Gamma$]

Here the top most symbols on stack is „b“ is replaced by „a“ followed by „b“, since the input „a“ is pushed / inserted on to the stack.

3) Pop operation on stack

The transition of a pop operation is given as, $\delta(q_1, a, b) = (q_2, \epsilon)$, „a“ cancels „b“ from the stack.

The above transition cancels the top most stack symbol by the input processing.

Example

$$L = \{w \in (a, b)^* \mid w \text{ is of the form } a^n b^n, n \geq 1\}$$

Algorithm

Consider $n = 3$

$$\therefore w = a^3 b^3 = a a a b b b$$

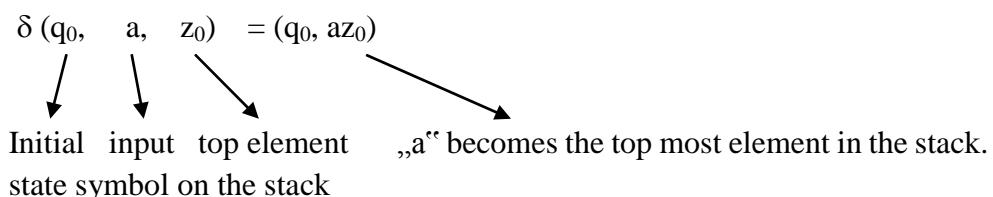
Step-1: Push „n“ number of „a“'s into the stack.

Step-2: For every „b“, pop out an „a“ from the stack.

Step-3: At the end of the string, the machine stops as it reaches the final state.

Transitions/moves

- Initially, the stack is empty. On processing the first „a“, the PDA pushes it onto the stack.



- On processing subsequent $(n-1)$ number of „a“'s, the stack pushes each „a“ being processed.

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

- On processing the first „b“, with the top most element of the stack, being „a“, the topmost element in the stack gets popped out of the stack.
- This is indicated by a state change from q_0 to q_1 .

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

- For each subsequent $(n-1)$ b“s, pop every „a“ out of the stack.

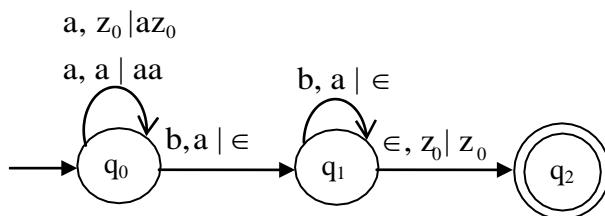
$$\delta(q_1, b, a) = (q_1, \epsilon)$$

- Finally when there are no more input strings, the input becomes „ ϵ “ and stack becomes „ z_0 “, which is considered as the end of the process, denoted by a state change $[q_1 \rightarrow q_2]$.

$$\delta(q_1, \epsilon, z_0) = (q_2, z_0)$$

- Q_2 is considered as the final state.

Thus the PDA transition diagram becomes,



where,

The PDA, $p = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ with

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

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

$$\Gamma = \{z_0, a, b\}$$

δ is given as,

$$\delta(q_0, a, z_0) = (q_0, a z_0)$$

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

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

$$\delta(q_1, b, a) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, z_0) = (q_2, z_0)$$

$$Q_0 = \{q_0\}$$

$$Z_0 = \{z_0\}$$

$$F = \{q_2\}$$

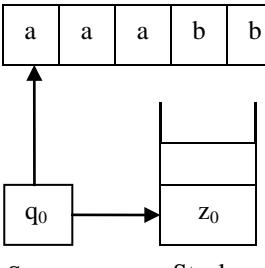
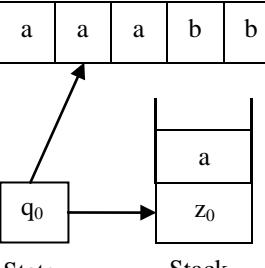
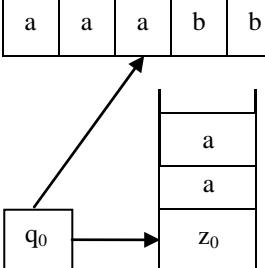
INSTANTANEOUS DESCRIPTION OF A PDA

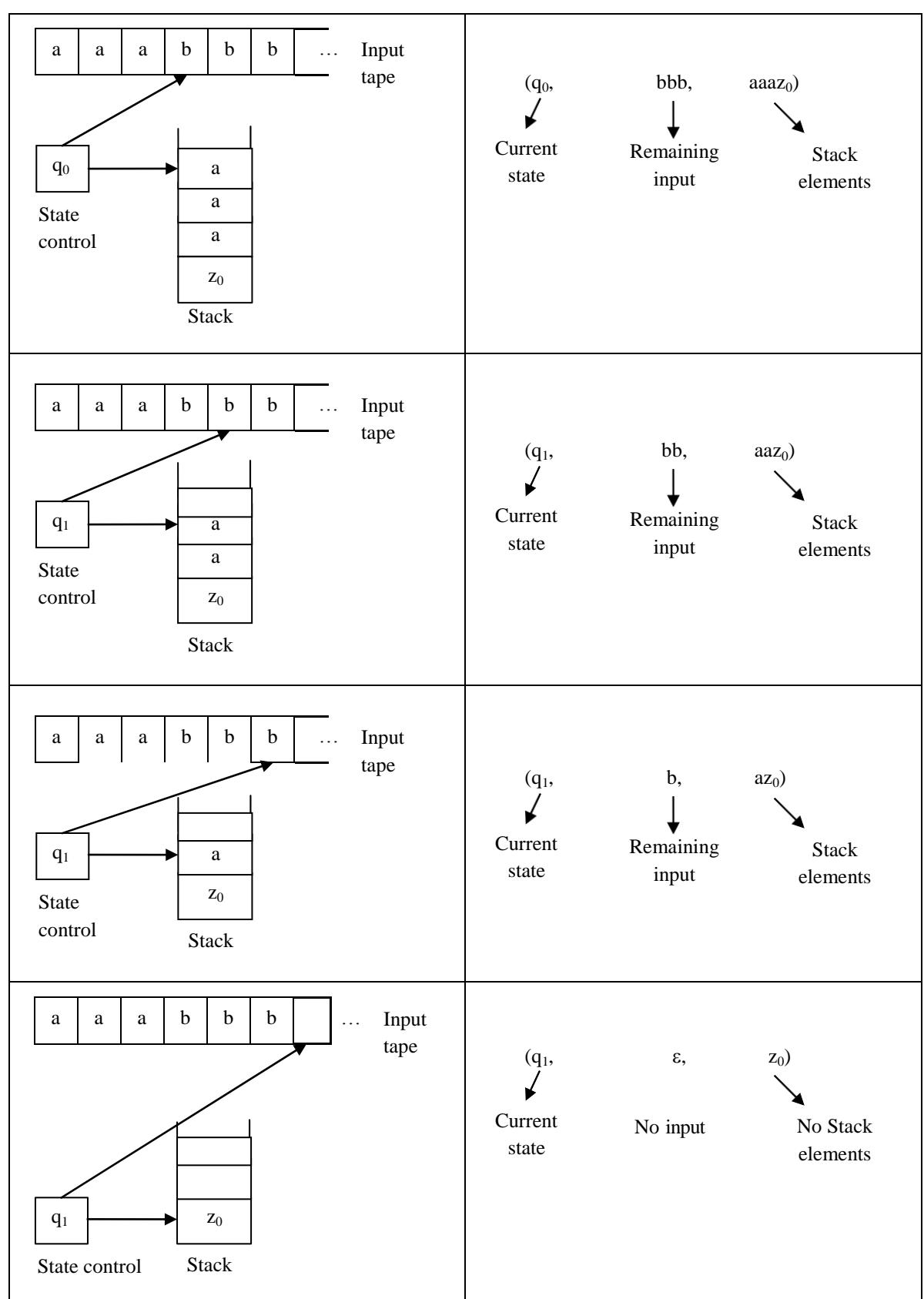
- Instantaneous description of a PDA is an informal notation or description of a PDA.
- It is a pictorial / diagrammatic representation of a string processed by a PDA.
- This is used to depict the processing of a string by a PDA.

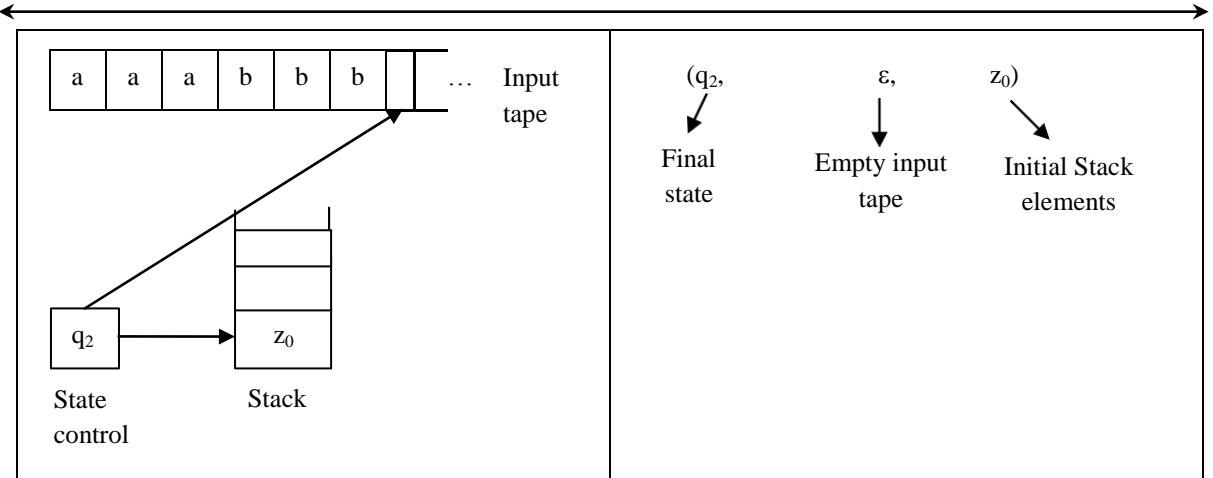
Example

$$L = \{a^n b^n \mid n \geq 1\} \quad [\text{before mentioned example}]$$

$$\text{Let } n = 3, \text{ then, } L = a^3 b^3 = aaabbb$$

Graphical representation (instantaneous description)	Tuple format
 <p>Input tape: a a a b b b ...</p> <p>State control: q_0</p> <p>Stack: z_0</p>	$(q_0, \text{aaabbb}, z_0)$ <p>Current state: q_0</p> <p>Input string (yet to be processed): aaabbb</p> <p>Stack elements: z_0</p>
 <p>Input tape: a a a b b b ...</p> <p>State control: q_0</p> <p>Stack: a z_0</p>	$(q_0, aabbb, az_0)$ <p>Current state: q_0</p> <p>Remaining input: aabbb</p> <p>Stack elements: az_0</p>
 <p>Input tape: a a a b b b ...</p> <p>State control: q_0</p> <p>Stack: a a z_0</p>	$(q_0, abbb, aaz_0)$ <p>Current state: q_0</p> <p>Remaining input: abbb</p> <p>Stack elements: aaz_0</p>





THE LANGUAGE OF A PDA

The language of a PDA can be accepted of two ways

- Acceptance by final state
- Acceptance of empty stack

The two methods are equivalent that the language that is accepted by a PDA by final state if and only if the language is accepted by empty stack.

However the PDA, P generated for a language is different than that is accepted by final state is different with that of the PDA generated by empty stack.

Acceptance By Final State

A PDA that accepts its input and enters the final / accepting state is called as a PDA accepted by final state.

Let the PDA, $P = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$. Then the language of the PDA, P is $L(P)$ which is accepted by final state is given by,

$$L(P) = \{ \omega \mid (q_0, \omega, z_0) \xrightarrow[P]{*} (q, \epsilon, \alpha) \}$$

where,

$q_0 \rightarrow$ initial state

$\omega \rightarrow$ input string

$z_0 \rightarrow$ initial stack symbol

$q \in Q \rightarrow$ final state [$q \in F$]

$\alpha \rightarrow$ stack symbols

Acceptance By Empty Stack

The PDA that accepts its input by emptying the stack is the PDA accepted by empty stack.

Let P be a PDA defined by $\{Q, \Sigma, \Gamma, \delta, q_0, z_0, F\}$. The language of the PDA accepted by empty stack is defined by,

$$N(P) = \{ \omega \mid (q_0, \omega, z_0) \xrightarrow{*} (q, \epsilon, \alpha) \}$$

Final state No input to process Stack empty

where,

- $q_0 \rightarrow$ initial state
- $\omega \rightarrow$ input string
- $z_0 \rightarrow$ initial stack symbol
- $q \rightarrow$ final state

Important Theorems

From Empty Stack To Final State

(AU – Nov/Dec 2011)

Theorem

If $L = N(P_N)$ for some PDA $P_N = \{Q, \Sigma, \Gamma, \delta_N, q_0, z_0, F\}$ then there is a PDA, P_F such that $L = L(P_F)$.

Proof

The theorem states that if there is a PDA, which is accepted by empty stack, then there should be a PDA, which is accepted by final state.

Consider, $x_0 \notin \Gamma$, that is the start symbol of P_F . Since x_0 is the start-symbol, it will be the bottom-most element of the stack (when processed).

Let P_0 be a new start state of P_N . P_0 pushes z_0 onto the stack and enters q_0 state . [$p_0 \rightarrow$ start state of P_N]

Let P_F be the final state of P_F , which stimulates P_N until the stack of P_N is empty.

P_F is stated as,

$$P_F = (Q \cup \{P_0, P_f\}, \Sigma, \Gamma \cup \{x_0\}, \delta_F, p_0, x_0, \{P_f\})$$

where,

$Q \cup \{P_0, P_f\}$ – finite set of states including p_0, p_f

Σ - set of input symbols

$\Gamma \cup \{x_0\}$ – stack symbols including x_0

- δ_F – transition function
 p_0 – start state of the PDA
 x_0 – initial stack symbol
 P_f – final state of the PDA

And δ_F is defined by

$$1. \quad \delta_F(p_0, \epsilon, x_0) = (q_0, z_0 x_0)$$

This shows a spontaneous transition that pushes the symbol z_0 on to the stack.

2. For all $q \in Q, a \in \{\Sigma \cup \epsilon\}$ and stack symbols $Y \in \Gamma$, $\delta_F(q, a, Y)$ contains all the pairs in $\delta_N(q, a, Y)$.
3. For every state $q \in Q$, $\delta_F(q, \epsilon, x_0)$ contains (p_f, ϵ)

To prove

The string ω is in $L(P_F)$ if and only if ω is in $N(P_N)$.

IF CASE

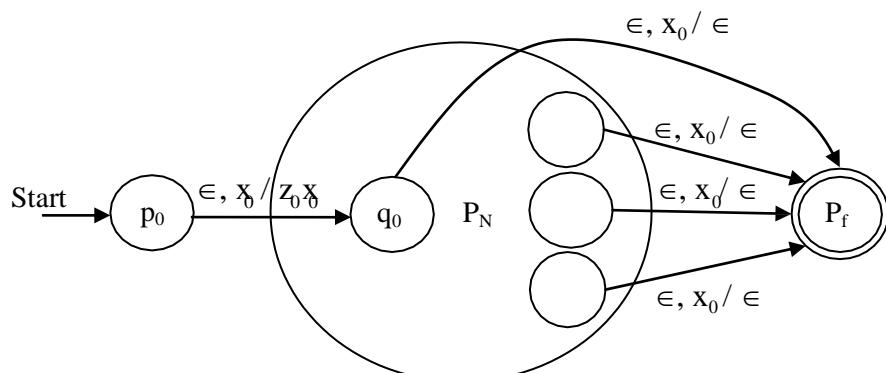
$$\text{Given } (q_0, \omega, z_0) \xrightarrow[P_N]{*} (q, \epsilon, \epsilon) \quad [q_0, \epsilon \in Q, q_0 \rightarrow \text{initial state}]$$

$$(p_0, \omega, x_0) \xrightarrow[P_F]{*} (p_0, \omega, z_0 x_0) \quad [\text{Using transition rule 1 as given above}]$$

$$\xrightarrow[P_F]{*} (q, \epsilon, x_0) \quad [\text{Using rule 2}]$$

$$\xrightarrow[P_F]{*} (p_f, \epsilon, \epsilon) \quad [\text{Using rule 3}]$$

Thus P_F accepts ω by final state.



P_F stimulates P_N and accepts if P_N empties its stack

ONLY – IF CASE

The rule – 1 of the transition rules causes P_N to enter the initial computation of P_F except that P_F will have its own bottom-of stack marker z_0 . [which are the symbols of P_F stack]

It is proved that

$$(q_0, \omega, z_0) \xrightarrow[P_N]{*} (q, \in, \in) \text{ [Rule 3]}$$

Thus ω is in $N(P_N)$.

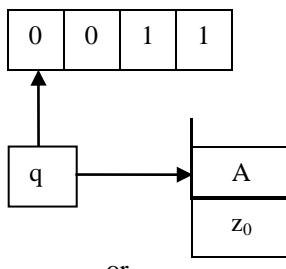
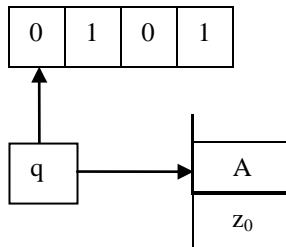
Example

Design a PDA that process equal number of 0's and 1's.

Algorithm

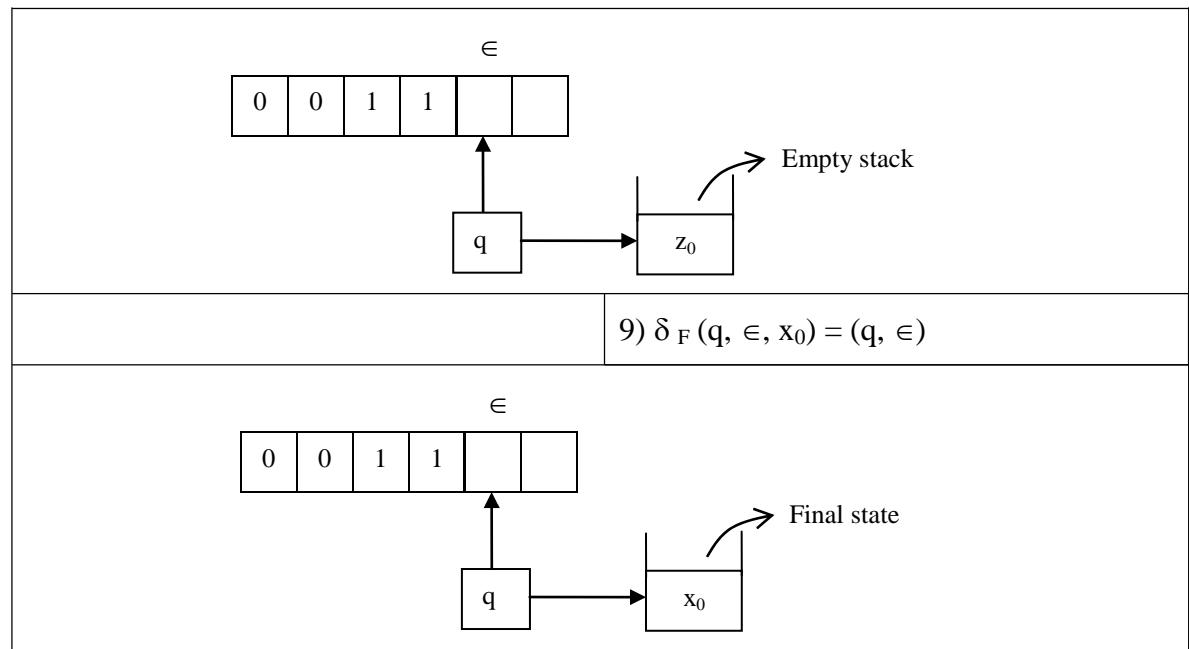
Step 1: Push the initial set of 0's / 1's onto the stack, indicating 0 as A and 1 as B.

Step 2: Pop out equivalent set of 0's / 1's for every incoming 1/0 present in the input tape.

PDA by empty stack	PDA by final state
<p>1) $\delta_N(q, 0, z_0) = (q, Az_0)$</p>  <p>or</p> 	<p>0) $\delta_F(p, \in, x_0) = (q, z_0 x_0)$</p> <p>Simulates P_N and pushes z_0 into the stack with x-as bottom of stack marker.</p> <p>1) $\delta_F(q, 0, z_0) = (q, Az_0)$</p> <p>Pushes „A“ onto the stack as „0“ is processed as the first input.</p>
2) $\delta_N(q, 1, z_0) = (q, Bz_0)$	2) $\delta_F(q, 1, z_0) = (q, Bz_0)$

3.61 Theory of Computation

3) $\delta_N(q, 0, A) = (q, AA)$	3) $\delta_F(q, 0, A) = (q, AA)$
4) $\delta_N(q, 0, B) = (q, \in)$	4) $\delta_F(q, 0, B) = (q, \in)$
5) $\delta_N(q, 1, A) = (q, \in)$	5) $\delta_F(q, 1, A) = (q, \in)$
6) $\delta_N(q, 1, B) = (q, BB)$	7) $\delta_F(q, 1, B) = (q, BB)$
7) $\delta_N(q, \in, z_0) = (q, \in)$	8) $\delta_F(q, \in, z_0) = (q, \in)$



From Final State To Empty Stack

Theorem

(AU – Nov/Dec 2011)

Let L be $L(P_F)$ for some PDA $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, z_0, F)$. Then there is a PDA, P_N such that $L = N(P_N)$.

Proof:

Let P_F simulates P_N as,

$$P_N = (Q \cup \{P_0, P\}, \Sigma, \Gamma \cup \{x_0\}, \delta_N, p_0, x_0, F)$$

Where δ_N can be defined as

$$1) \delta_N(p_0, \in, x_0) = (q_0, z_0, x_0)$$

The start symbol of P_F is pushed onto the stack.

- 2) For all states $q \in Q$, input symbols $a \in \{\Sigma \cup \in\}$ and stack symbols $y \in \Gamma$,
- $$\delta_N(q, a, y) = \delta_F(q, a, y) \text{ for all pairs of transitions.}$$

Every transition in P_N should also be in P_F

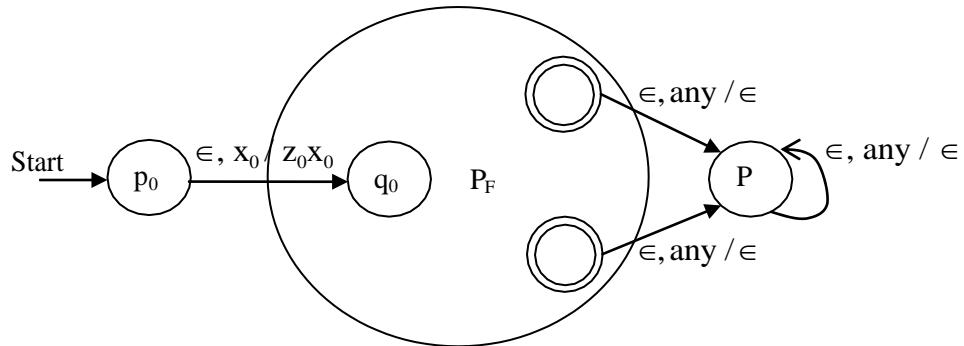
$$3) \delta_N(q, \in, y) = (p, \in), \text{ for all final states } q \in F, \text{ stack symbols } y \in \{\Gamma \cup x_0\}$$

P_N starts to empty the stack contents whenever P_F accepts.

$$4) \delta_N(p, \in, y) = (p, \in), \text{ for all stack symbols } y \in \{\Gamma \cup x_0\}$$

P_N pops every stack symbols, when P_F has accepted at some state, P .

PN stimulates P_F empties its stack, when P_N enters final state.



To prove

For a string, ω is in $N(P_N)$ if and only if ω is in $L(P_F)$.

If-CASE

$$(q, \omega, z_0) \xrightarrow[P_F]{*} (q, | \in \alpha) \quad [\text{Assume}]$$

Where

$q \rightarrow \text{final state}$

$\alpha \rightarrow \text{stack symbols}$

Invoking x_0 be the bottom of stack symbols,

$$(q_0, \omega, z_0 x_0) \xrightarrow[P_N]{*} (q, \in, \alpha x_0)$$

$$(p_0, \omega, x_0) \xrightarrow[P_N]{*} (q_0, \omega, z_0 x_0) \quad [\text{Rule 1}]$$

$$\xrightarrow[P_N]{*} (q, \in, a x_0) \quad [\text{Rule 3}]$$

$$\xrightarrow[P_N]{*} (p, \in, \in) \quad [\text{Rule 4}]$$

Thus ω is accepted by PDA by empty stack, P_N .

ONLY-IF CASE

The only way for emptying the stack of P_N is by entering into the state, P which is stimulated by P_F (entering final state).

Thus P_N becomes,

$$(p_0, \omega, x_0) \xrightarrow[P_N]{\cdot} (q_0, \omega, z_0 x_0) \quad [\text{Rule 1}]$$

$$\xrightarrow[P_N]{\cdot}^* (q, \epsilon, \alpha x_0) \quad [\text{Rule 3}]$$

$$\xrightarrow[P_N]{\cdot}^* (p, \epsilon, \epsilon)$$

Where,

$$q \rightarrow \text{Final state of } P_F$$

Between $(q_0, \omega, z_0 x_0) \xrightarrow[P_N]{\cdot}^* (q, \epsilon, \alpha x_0)$, all the transitions are the transition moves of P_F .

Thus ω is in $L(P_F)$

DETERMINISTIC PUSHDOWN AUTOMATA

(AU – Nov/Dec 2010)

PDA is classified into two types based on the transitions.

1. Deterministic PDA [DPDA]
2. Non-deterministic PDA [NPDA]

Deterministic PDA

A PDA that has only one transition for an input at any particular state is called as DPDA.

It does not have any choices of transition in any state for the inputs.

$\delta(q, a, x)$ is always singleton for the input symbol, a in Σ and state, q in Q .

Formal definition

DPDA, P is given by

$$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

where,

$Q \rightarrow$ Finite set of states

$\Sigma \rightarrow$ Set of input symbols

$\Gamma \rightarrow$ Finite set of stack symbols

$\delta \rightarrow$ Transition functions given by

- (q, a, x) has at most one transition for any q in Q , a in Σ or ϵ , X in Γ .
- If $\delta(q, a, x)$ is non empty for a in Σ , then $\delta(q, \epsilon, x)$ must be empty.

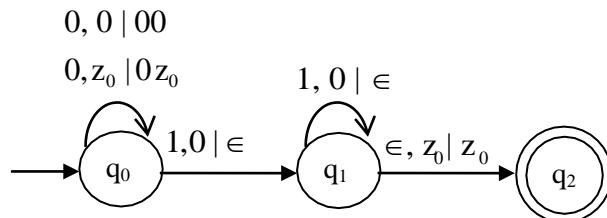
$q_0 \rightarrow$ Start state

$z_0 \rightarrow$ Initial stack symbol

$F \rightarrow$ Final state

Example

DPDA for $L = \{0^n 1^n \mid n > 1\}$



Note

- 1) If L is a regular language, then $L = L(P)$ for some DPDA, P .
- 2) A language L is $N(P)$ for some DPDA, P if and only if L has the prefix property and L is $L(P^1)$ for some DPDA, P^1 .
- 3) If $L = N(P)$ for some DPDA, P then L has an unambiguous context-free grammar.
- 4) If $L = L(P)$ for some DPDA, P then L has an unambiguous CFG.
- 5) The two modes of accepting a deterministic PDA is
 - Empty stack
 - Final state

Problems

1. Let $L = \{a^m b^n \mid m > n\}$. Construct a PDA

- Accepted by empty stack
- Accepted by final state

Solution:

Algorithm

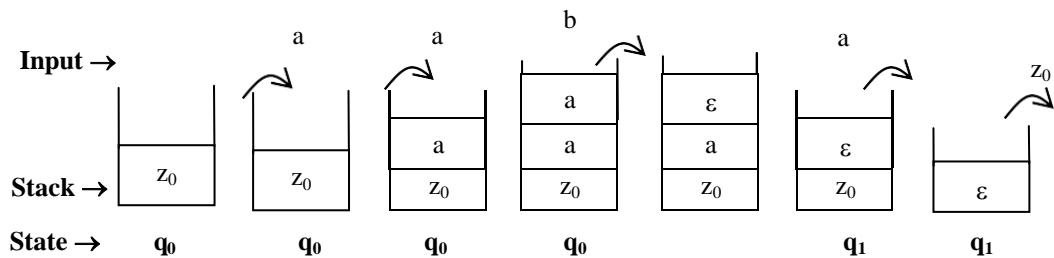
1. Push the „m“ number of a ’s onto the stack.

2. Pop out every „a“ for each „b“
 3. Pop out the extra (m-n) number for a“s from the stack
 4. Pop out z_0 from the stack to make the stack empty.

Instantaneous description

Let $m = 2$ and $n = 1$ ($2 > 1$)

$$\omega = a^2 b^1 = aa b$$



Transition functions

Acceptance by empty stack	Acceptance by final state
<ol style="list-style-type: none"> $\delta(q_0, a, z_0) = (q_0, a z_0)$ $\delta(q_0, a, a) = (q_0, a a)$ $\delta(q_0, b, a) = (q_1, \epsilon)$ $\delta(q_1, b, a) = (q_1, \epsilon)$ $\delta(q_1, \epsilon, a) = (q_1, \epsilon)$ $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$ 	<ol style="list-style-type: none"> $\delta(q_0, a, z_0) = (q_0, a z_0)$ $\delta(q_0, a, a) = (q_0, a a)$ $\delta(q_0, b, a) = (q_1, \epsilon)$ $\delta(q_1, b, a) = (q_1, \epsilon)$ $\delta(q_1, \epsilon, a) = (q_2, \epsilon)$

PDA accepted by empty stack,

$$P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, z_0)$$

where,

$$Q \rightarrow \{q_0, q_1\}$$

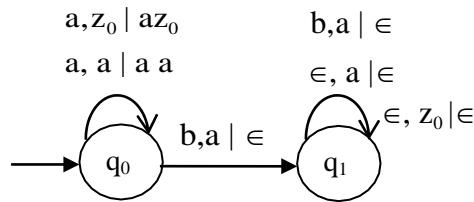
$$\Sigma \rightarrow \{a, b\}$$

$$\Gamma \rightarrow \{z_0, a, b\}$$

$$\delta_N \rightarrow \text{As shown in the transition diagram}$$

$$q_0 \rightarrow \{q_0\}$$

$$z_0 \rightarrow \{z_0\}$$



PDA accepted by final state

$$P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, z_0, F)$$

where,

$$Q \rightarrow \{q_0, q_1, q_2\}$$

$$\Sigma \rightarrow \{a, b\}$$

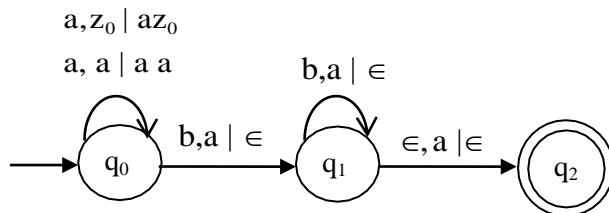
$$\Gamma \rightarrow \{z_0, a, b\}$$

$\delta_N \rightarrow$ As shown in transition diagram

$$q_0 \rightarrow \{q_0\}$$

$$z_0 \rightarrow \{z_0\}$$

$$F \rightarrow \{q_2\}$$



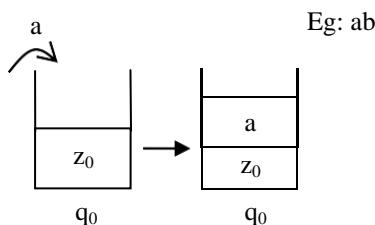
2. Design a PDA that accepts a set of strings over {a, b} with equal number of a's and b's. (AU – June 2007)

Solution:

Step 1: Stack contains z_0 at top. When input = a, it is pushed into stack.

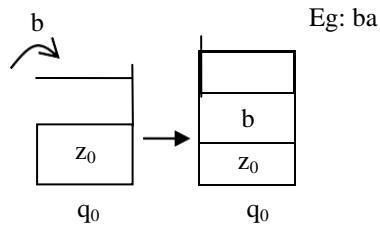
$$\delta(q_0, a, z_0) = (q_0, a z_0)$$

If $a = 2$; $b = 2$



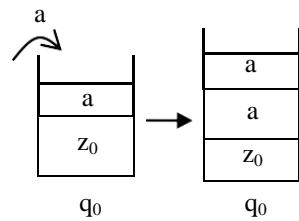
Step 2: Stack contains z_0 at top. When input = b, it is pushed into stack.

$$\delta(q_0, b, z_0) = (q_0, bz_0)$$



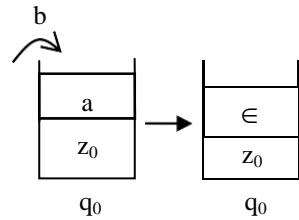
Step 3: Stack contains top = a and if input = a, then excess a is increased by 1.

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



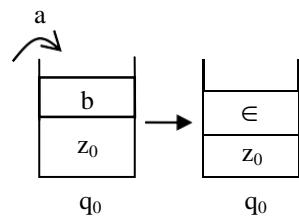
Step 4: Stack contains top = a and if input = b, then excess a is decreased by 1.

$$\delta(q_0, b, a) = (q_0, \in)$$



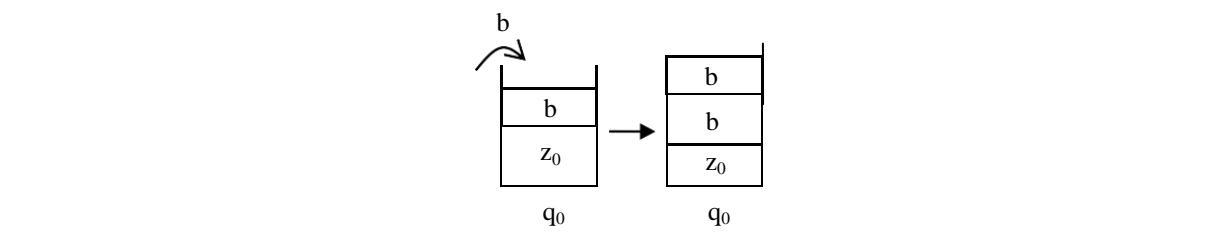
Step 5: Stack with top = b, input = a, excess b is decreased by 1.

$$\delta(q_0, a, b) = (q_0, \in)$$



Step 6: Stack contains top = b, input = b, excess b is increased by 1.

$$\delta(q_0, b, b) = (q_0, bb)$$

**Transition function**

PDA accepting through an empty stack	PDA accepting through final state
$\delta(q_0, a, z_0) = (q_0, a z_0)$ $\delta(q_0, b, z_0) = (q_0, b z_0)$ $\delta(q_0, a, b) = (q_0, \in)$ $\delta(q_0, b, a) = (q_0, \in)$ $\delta(q_0, a, a) = (q_0, aa)$ $\delta(q_0, b, b) = (q_0, bb)$ $\delta(q_0, \in, z_0) = (q_0, \in)$	$\delta(q_0, a, z_0) = (q_0, a z_0)$ $\delta(q_0, b, z_0) = (q_0, b z_0)$ $\delta(q_0, a, b) = (q_0, \in)$ $\delta(q_0, b, a) = (q_0, \in)$ $\delta(q_0, a, a) = (q_0, aa)$ $\delta(q_0, b, b) = (q_0, bb)$ $\delta(q_0, \in, z_0) = (q_1, z_0)$
Transition diagram <pre> graph LR start(()) --> q0((q0)) q0 -- "a, a aa" --> q0 q0 -- "a, b bb" --> q0 q0 -- "a, z0 bz0" --> q0 q0 -- "a, z0 az0" --> q0 q0 -- "a, b \u2225" --> q0 q0 -- "b, a \u2225" --> q0 q0 -- "\u2225, z0 \u2225" --> q0 </pre>	$a, a aa$ $b, b bb$ $b, z_0 bz_0$ $a, z_0 az_0$ $a, b \in$ $b, a \in$ <pre> graph LR start(()) --> q0((q0)) q0 -- "\u2225, z0 z0" --> q1(((q1))) </pre>
PDA: $M = (\{q_0\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \phi)$	PDA: $M = (\{q_0, q_1\}, \{a, b\}, \{a, b, z_0\}, \delta, q_0, z_0, \{q_1\})$

3. Give the transition table for PDA recognizing the language, $L = \{a^n x \mid n \geq 0 \text{ & } x \in \{a,b\}^* \text{ & } |x| \leq n\}$.

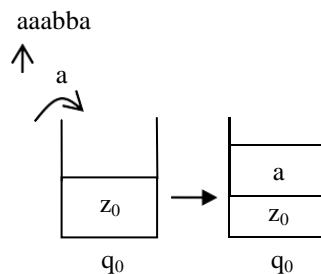
Solution:

Algorithm:

Step 1: Push the first $,a^n$ into the stack

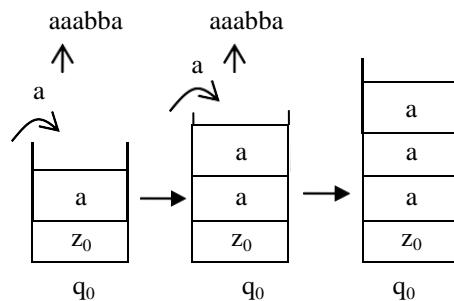
$$\delta(q_0, a, z_0) = (q_0, a z_0)$$

Let $w = a^3 bb a$



Step 2: Push (n-1) a's on to the stack

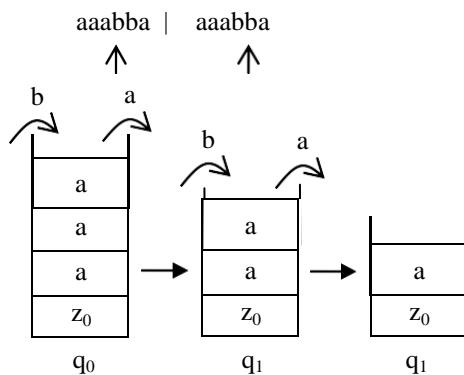
$$\delta(q_0, a, a) = (q_0, a a)$$



Step 3: When input = x [$x \rightarrow b$], pop out the „a“ from the stack.

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

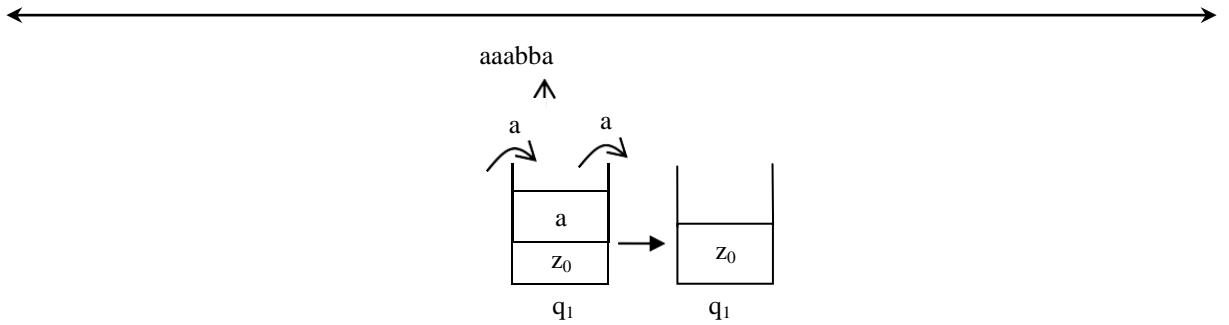
\uparrow
 x



Step 4: On subsequent a's or b's in x, and „a“ should be popped out.

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

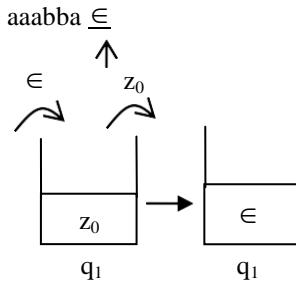
$$\delta(q_1, b, a) = (q_1, \in)$$



Step 5: After the end of input string, contents of the stack are popped out one by one.

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

$$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$$



Thus the PDA is given by

$$M = (\{q_0, q_1\}, \{a, b\}, \{z_0, a\}, \delta, q_0, z_0, \phi)$$

Where δ is given as

$$\left. \begin{array}{l} \delta(q_0, a, z_0) = (q_0, a z_0) \\ \delta(q_0, a, a) = (q_0, a a) \end{array} \right\} \rightarrow \text{Accepting } a^n$$

$$\delta(q_0, \epsilon, z_0) = (q_0, \epsilon) \rightarrow \text{for null string}$$

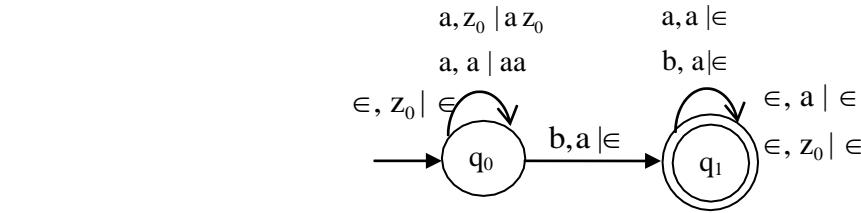
$$\delta(q_0, b, a) = (q_1, \epsilon) \rightarrow \text{pop out a for every b}$$

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

$$\delta(q_1, b, a) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, a) = (q_1, \epsilon) \rightarrow \text{Emptying the stack}$$

$$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$$



4. Let $L = \{a^n b^n c^m d^m \mid n, m \geq 1\}$. Find a PDA for L.

Solution:

Algorithm:

Step 1: Push „a“ onto the stack

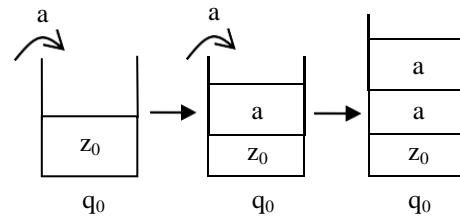
$$\delta(q_0, a, z_0) = (q_0, a z_0)$$

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

Let $n = 2$ and $m = 1$

$$W = aabbcd$$

\nwarrow



Step 2: For every „b“ as input, „a“ should be popped out.

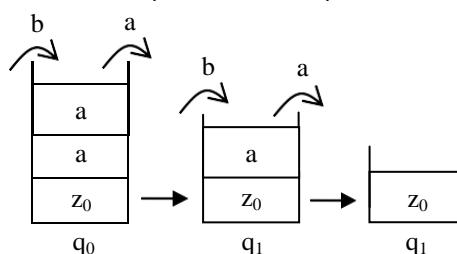
$$\delta(q_0, b, a) = (q_1, \in)$$

$$\delta(q_1, b, a) = (q_1, \in)$$

$$W = aabbcd \quad w = aabbcd$$

\uparrow

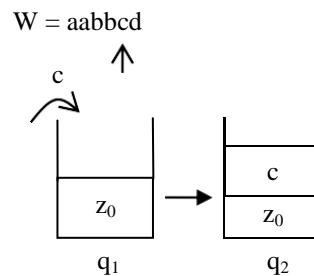
\uparrow



Step 3: Push „c“ onto the stack

$$\delta(q_1, c, z_0) = (q_0, c z_0)$$

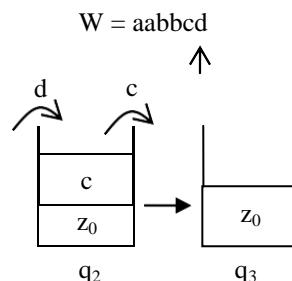
$$\delta(q_0, c, c) = (q_0, c c)$$



Step 4: For every „d“ as input „c“ should be popped out

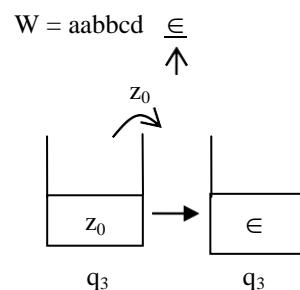
$$\delta(q_2, d, c) = (q_3, \in)$$

$$\delta(q_3, d, c) = (q_3, \in)$$



Step 5: String is accepted as,

$$\delta(q_3, \in, z_0) = (q_3, \in)$$



\therefore The PDA is given as

$$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{z_0, a, c\}, \delta, q_0, z_0, \phi)$$

with δ as given in the above algorithm.

5. Let $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ & } i + j = k\}$. Provide the transition function.

(i) Accepted by final state

(ii) Accepted by empty stack

Solution:

Algorithm:

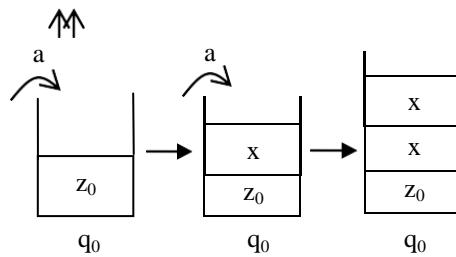
Step 1: For every input symbol „a“, symbol „x“ is pushed onto the stack.

$$\delta(q_0, a, z_0) = (q_0, x z_0)$$

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

$$i = 2, j = 2 \Rightarrow k = 2 + 2 = 4$$

$$W = aabbcccc$$



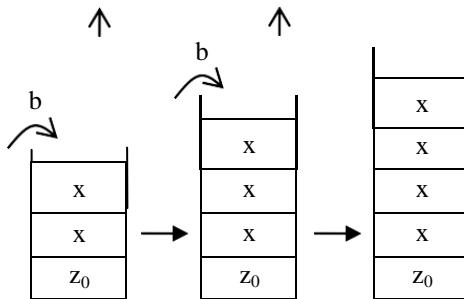
Step 2: For every input symbol „b“ symbol „x“ is pushed onto the stack.

$$\delta(q_0, b, z_0) = (q_1, x z_0)$$

$$\delta(q_0, b, c) = (q_1, x x)$$

$$\delta(q_1, b, x) = (q, x x)$$

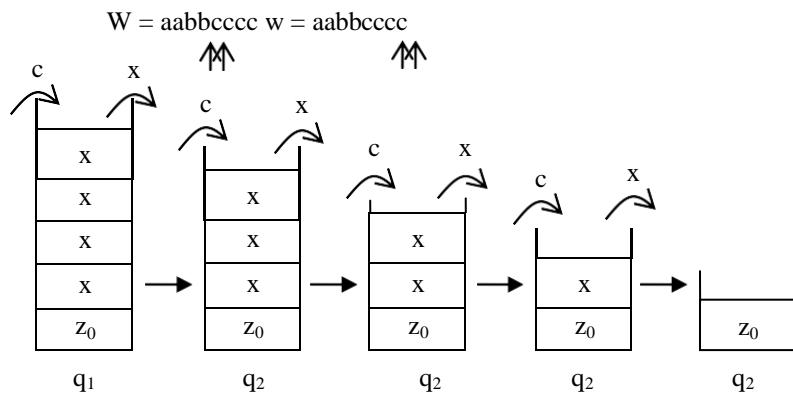
$$W = aabbcccc \quad w = aabbccdd$$



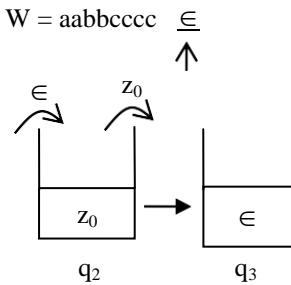
Step 3: For every input symbol „c“, „x“ is popped out from the stack.

$$\delta(q_1, c, x) = (q_2, \in)$$

$$\delta(q_2, c, x) = (q_2, \in)$$



Step 4: $\delta(q_2, \in, z_0) = (q_3, z_0)$
 $\delta(q_0, \in, z_0) = (q_3, z_0)$ Null string



PDA: Through final state	PDA: Through empty stack
$M = (\{q_0, q_1, q_2, q_3\}, \{a, b, c\}, \{x, z_0, \}, q_0, z_0, \delta, \{q_3\})$ where δ is given as $\delta(q_0, a, z_0) = (q_0, x z_0)$ $\delta(q_0, a, x) = (q_0, x x)$ $\delta(q_0, b, z_0) = (q_1, x z_0)$ $\delta(q_0, b, x) = (q_1, x x)$ $\delta(q_1, b, x) = (q_1, x x)$ $\delta(q_1, c, x) = (q_2, \in)$ $\delta(q_2, c, x) = (q_2, \in)$ $\delta(q_2, \in, z_0) = (q_3, z_0)$ $\delta(q_0, \in, z_0) = (q_3, z_0)$	$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{x, z_0, \}, q_0, z_0, \delta, \phi)$ where δ is given as $\delta(q_0, a, z_0) = (q_0, x z_0)$ $\delta(q_0, a, x) = (q_0, x x)$ $\delta(q_0, b, z_0) = (q_1, x z_0)$ $\delta(q_0, b, x) = (q_1, x x)$ $\delta(q_1, b, x) = (q_1, x x)$ $\delta(q_1, c, x) = (q_2, \in)$ $\delta(q_2, c, x) = (q_2, \in)$ $\delta(q_2, \in, z_0) = (q_2, \in)$ $\delta(q_0, \in, z_0) = (q_0, \in)$

6. Design a PDA for the language $\{a^m b^n c^m \mid m, n \geq 1\}$ using empty stack.

(AU – Dec 2008)

Algorithm:

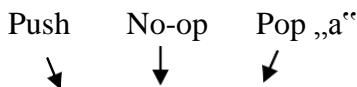
- 1) Push the sequence of a's onto the stack.
- 2) Skip the sequence of b's by doing no-operation
- 3) For every „c“, pop out an „a“ from the stack.

Transitions:

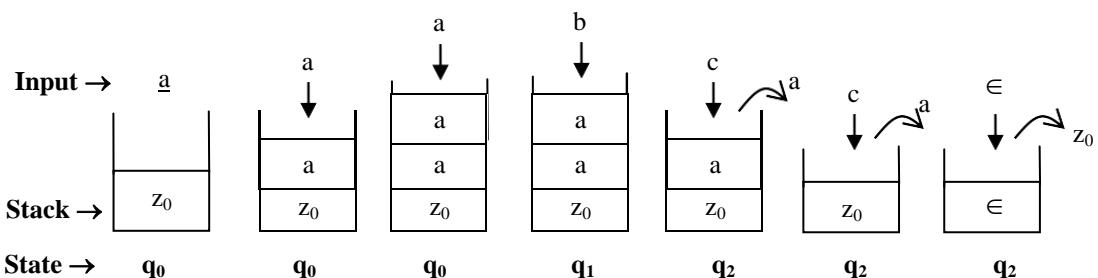
$\delta(q_0, a, z_0) = (q_0, a z_0)$	Push „m“ number of a's onto the stack
$\delta(q_0, a, a) = (q_0, a a)$	
$\delta(q_0, b, a) = (q_1, a)$	Skip „n“ no. of b's and do no operation on stack
$\delta(q_0, b, a) = (q_1, a)$	
$\delta(q_1, c, a) = (q_2, \in)$	Pop out every „a“ for each „c“ from the stack
$\delta(q_2, c, a) = (q_2, \in)$	
$\delta(q_2, \in, z_0) = (q_2, \in)$	Empty the stack

Instantaneous description

Let $n = 2, m = 1$



Then $w = \underline{a} \ a \quad \underline{b} \quad \underline{cc}$



Therefore, $M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{z_0, a, c\}, \delta, \{q_0\}, \{z_0\}, \phi)$

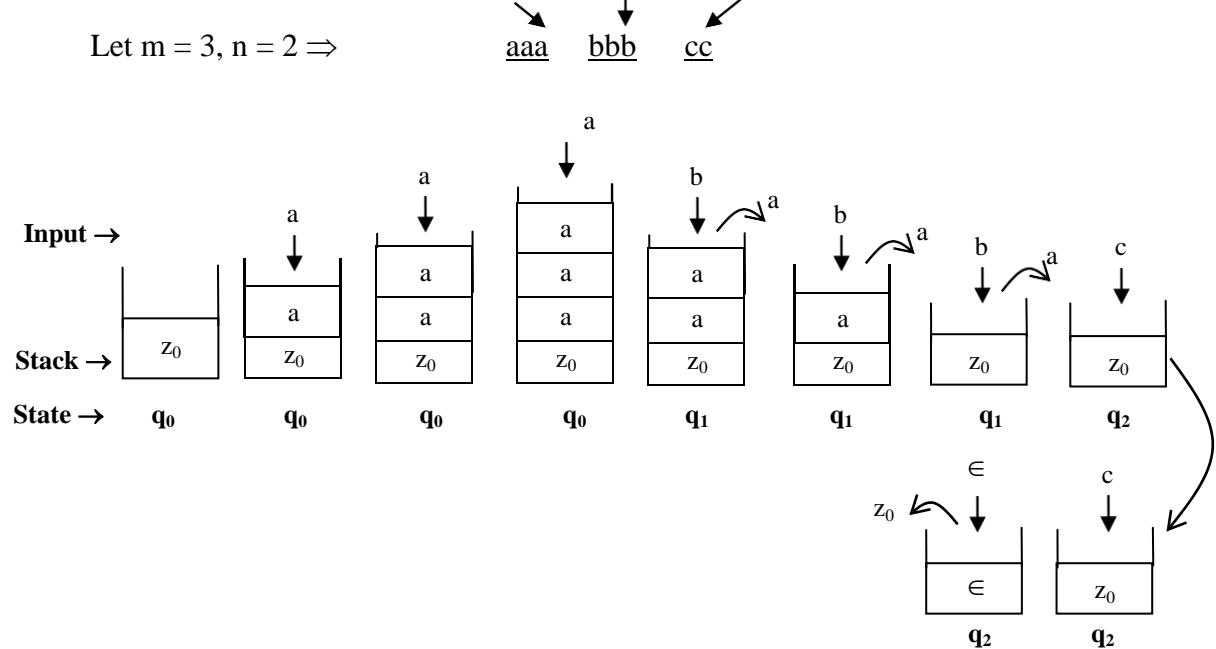
7. Construct a PDA for the language accepting by empty stack.

$$L = \{a^m b^m c^n \mid m, n \geq 1\}$$

(AU – June 2007)

Algorithm

1. Push „m“ number of ‘a’s onto the stack
2. For every ‘b’ being processed, pop out an equivalent ‘a’ from the stack.
3. Skip ‘c’ and make the stack empty.

Instantaneous Description

$$\therefore \text{PDA, } M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{z_0, a, b\}, \delta, \{q_0\}, \{z_0\}, \phi)$$

where δ is given by

$$\delta(q_0, a, z_0) = (q_0, a z_0) \quad \text{Step 1}$$

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

$$\delta(q_0, b, a) = (q_1, \epsilon) \quad \text{Step 2}$$

$$\delta(q_1, b, a) = (q_1, \epsilon)$$

$$\delta(q_1, c, z_0) = (q_2, \epsilon)$$

$$\delta(q_2, c, z_0) = (q_2, \epsilon) \quad \text{Step 3}$$

$$\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$$

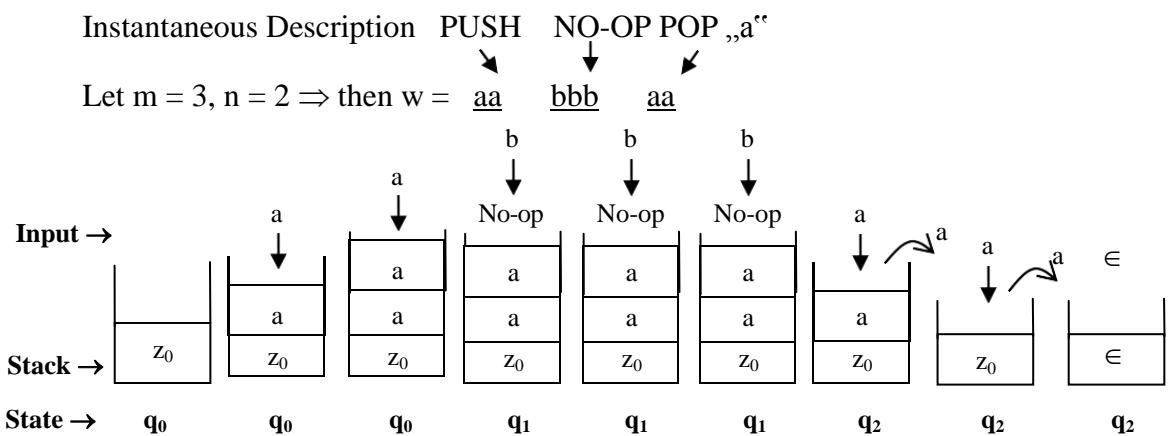
8. Construct a PDA accepting $L = \{a^n b^m a^n \mid m, n \geq 1\}$ by nullstore.

(AU – Dec 2004)

Algorithm

1. Push „n“ number of a’s onto the stack
 2. Skip „m“ number of b’s
 3. For every „a“ after „b“, pop out one „a“ from the stack and empty the stack elements

Instantaneous description



PDA :: M = ($\{q_0, q_1, q_2\}$, $\{a, b\}$, $\{z_0, a\}$, δ , $\{q_0\}$, $\{z_0\}$, ϕ)

where δ is given by

$$\delta(q_0, a, z_0) = (q_0, a z_0) \quad \text{Step 1}$$

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

$$\delta(q_0, b, a) = (q_1, a) \quad \text{Step 2}$$

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

$$\delta(q_1, a, a) = (q_2, \in)$$

$$\delta(q_2, a, a) = (q_2, \in) \quad \text{Step 3}$$

$$\delta(q_2, \in, z_0) = (q_2, \in)$$

9. Design a PDA for accepting $L = \{WCW^R \mid \omega \in \{a, b\}\}$.

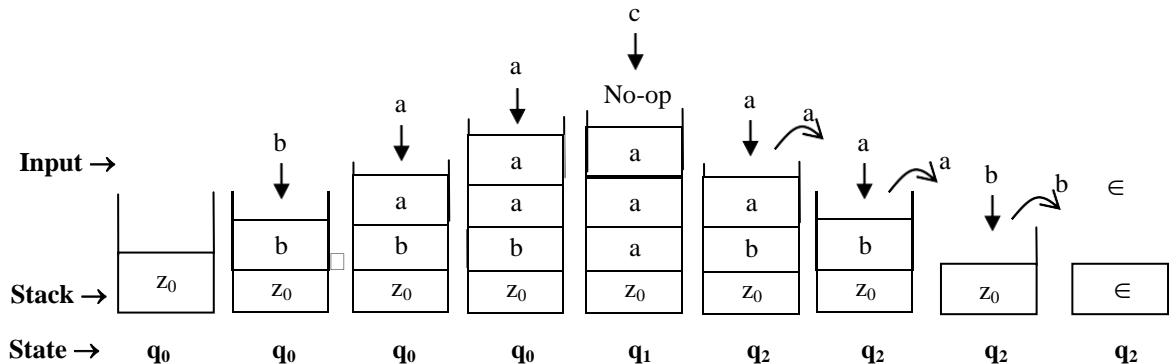
(AU – Nov/Dec 2010, Nov/Dec 2013)

Algorithm

1. Push the string ω containing a's and b''s onto the stack.
 2. Skip „c“.
 3. Pop out every symbol of the stack by the input string.

Instantaneous description

Let $w = baa$; $L = baacaab$



PDA:

$$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{z_0, a, b\}, \delta, \{q_0\}, \{z_0\}, \phi)$$

δ is given by

$$\delta(q_0, a, z_0) = (q_0, a z_0)$$

$$\delta(q_0, b, z_0) = (q_0, b z_0)$$

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

$$\delta(q_0, a, b) = (q_0, ab) \quad \text{Step 1}$$

$$\delta(q_0, b, a) = (q_0, ba)$$

$$\delta(q_0, b, b) = (q_0, bb)$$

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

Step 2

$$\delta(q_0, c, b) = (q_1, b)$$

$$\delta(q_1, a, a) = (q_2, \in)$$

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

$$\delta(q_2, a, a) = (q_2, \in) \quad \text{Step 3}$$

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

$$\delta(q_2, \in, z_0) = (q_2, \in)$$

10. Construct a PDA for the language $L = \{a^n b^{2n} \mid n \geq 0\}$

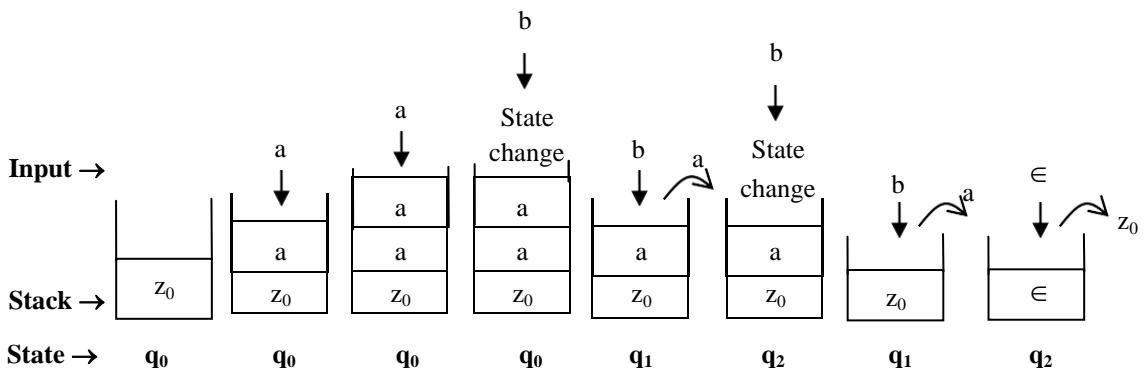
(AU – May 2007, Dec 2007, Dec 2008)

Algorithm

1. Push „n“ number of „a“ on to the stack.
2. For every odd no of „b“ make a state change indicating that one „b“ is processed.
3. For every even no. of „b“, pop out an „a“ from the stack, indicating that a pair of „b“ is processed.

Instantaneous description

Let $n = 2 \Rightarrow L = a^2 b^4 = aabbba$


PDA:

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{z_0, a\}, \delta, \{q_0\}, \{z_0\}, \phi)$$

δ is given by

$$\begin{aligned} \delta(q_0, a, z_0) &= (q_0, a z_0) && \text{Step 1} \\ \delta(q_0, a, a) &= (q_0, aa) \\ \delta(q_0, b, a) &= (q_1, a) \longrightarrow \text{Step 2} \\ \delta(q_1, b, a) &= (q_2, \epsilon) \\ \delta(q_2, b, a) &= (q_2, \epsilon) \quad \text{Step 3} \\ \delta(q_2, \epsilon, z_0) &= (q_2, \epsilon) \end{aligned}$$

EQUIVALENCE OF PDA AND CFG

(AU – May/June 2013, Nov/Dec 2010, Nov/Dec 2013)

For a given CFG, $G = (V, T, P, S)$, we can construct a PDA, M that simulates the leftmost derivation of G .

The PDA accepting $L(G)$ by empty stack is given by

$$M = (\{q\}, T, VUT, \delta, q, S, \phi)$$

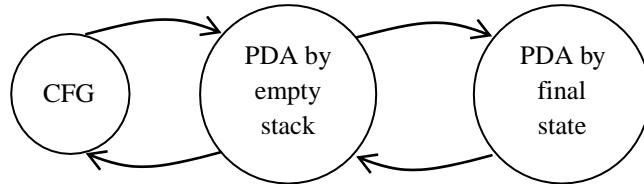
where δ is defined by

1. For each variable $A \in V$, include a transition

$$\delta(q, \epsilon, A) \Rightarrow \{(q, \alpha) \mid A \rightarrow \alpha \text{ is a production in } G\}$$

2. For each terminal, $a \in T$, include a transition

$$\delta(q, a, a) \Rightarrow \{(q, \epsilon)\}$$



From Grammar To PDA

Theorem

If PDA, P is constructed from CFG, G then $N(P) = L(G)$

(Or)

If L is generated by a CFG, then there exists a PDA accepting L .

Proof

To prove

ω is in $N(P)$ if and only ω is in $L(G)$.

IF – CASE

Let ω is in $L(G)$.

The left most derivation of ω is given as,

$$S \Rightarrow \gamma_1 \xrightarrow{\ell_m} \gamma_2 \xrightarrow{\ell_m} \square \xrightarrow{\ell_m} \gamma_n = \omega$$

Let us use mathematical induction principle on i to show that

$$(q, \omega, S) \xrightarrow[\mathbf{P}]{}^* (q, y_i, \alpha_i)$$

where

$$\gamma_i = x_i \alpha_i \quad [\alpha_i \text{ is the tail of } \gamma_i]$$

$y_i, \alpha_i \rightarrow$ Left sentential forms of γ_i

and,

$$x_i \omega_i = \omega \text{ [string]}$$

Basis of Induction:

Let us assume the length $i=1$

Then,

$$S = \gamma_i$$

Since $\gamma_i = x_i \alpha_i$

$$\Rightarrow \gamma_1 = x_1 \alpha_1 \quad [x_1 = \epsilon]$$

$$\gamma_1 = \alpha_1$$

And $x_i y_i = \omega$

$$\Rightarrow x_1 y_1 = \omega$$

$$y_1 = \omega \text{ [since } x_1 = \epsilon \text{]}$$

$\therefore (q, \omega, S) \xrightarrow[P]{*} (q, y_i, \alpha_i)$ becomes,

$$(q, \omega, S) \xrightarrow{*} (q, y_1, \alpha_1)$$

$$\xrightarrow{*} (q, \omega, S) \quad [\text{since } S = \gamma_1 = \alpha_1; \alpha_1 = S \text{ and } y_1 = \omega]$$

Rule 1:

$$\delta(q, \epsilon, A) \Rightarrow \{(q, \beta) \mid A \rightarrow \beta \text{ is in } P\}$$

Rule 2:

For $a \in T$

$$\delta(q, a, a) \Rightarrow (q, \epsilon)$$

Thus,

$$(q, \omega, S) \xrightarrow{*} (q, \omega, S) \text{ on } x_1 = \epsilon \text{ moves.}$$

Basis of induction is proved.

Inductive step

Assume the $(q, \omega, S) \xrightarrow[P]{*} (q, y_i, \alpha_i)$ is true for „i“ and prove that $(q, \omega, S) \xrightarrow[P]{*} (q, y_{i+1}, \alpha_{i+1})$ is true for „i+1“.

Consider a variable A that the derivation $\gamma_i = \gamma_{i+1}$ replaces the variable A by a production, β .

This is done by Rule – 1 on the construction of P that replaces A by β [at the top of the stack].

By applying Rule – 2, we shall match any terminal on the top of the stack by next input symbol.

Thus, $ID(q, y_{i+1}, \alpha_{i+1})$ gives the value for γ_{i+1} .

Here

$$\alpha_n = \in \text{ [since tail of } \gamma_n = \text{empty}]$$

$$\therefore \gamma_n = x_n = \omega$$

Thus $(q, \omega, S) \xrightarrow[P]{*} (q, \in, \in)$ since $y_n = \in$ and $\alpha_n = \in$.

Therefore, the inductive step is proved that PDA, P accepts ω by empty stack.

ONLY-IF CASE

Let A be a variable, that is to be popped out of a stack while executing a PDA, P then, A derives a the input string, x in G if,

$$(q, x, A) \xrightarrow[P]{*} (q, \in, \in), \text{ then}$$

$$A \xrightarrow[G]{*} x$$

This is proved by mathematical induction principle on the number of moves taken by P.

Basis of induction

When grammar has the production $A \rightarrow \in$, then the PDA becomes,

$$X = \in \text{ [by rule 1] } [\delta(q, \in, A) \Rightarrow \{(q, \beta) \mid A \rightarrow \beta \text{ is in } P\}]$$

Since $A \Rightarrow \in$

Thus basis of induction is proved.

Inductive step

Assume that P takes „n“ moves, where $n > 1$. The first move must be of rule – 1 (as given above).

Here A is replaced by another production.

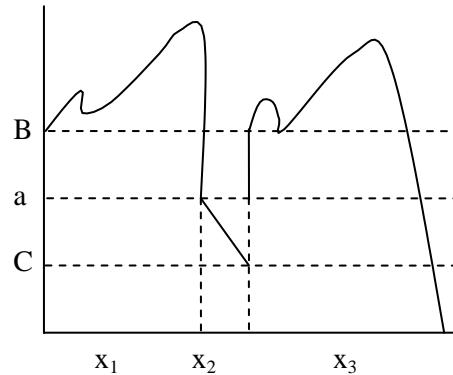
Suppose

$$A \rightarrow Y_1, Y_2, \dots, Y_k \mid Y_i \in \{TUV\}$$

The next $(n - 1)$ moves of P, takes on „x“ from the input tape, that pops out.

x is broken into x_1, x_2, \dots, x_k , where x_i is the input processed until y_i is popped out of the stack.

Let x_2 is the next input portion that is processed, popping out y_2 off the stack, and so on.



PDA consuming x and popping BaC out of stack

Here, x is divided into 3 parts x_1, x_2, x_3 with $x_2 = a$.

In general, if y_i is a terminal, then x_i must be that terminal.

Thus,

$$(q, x_i x_{i+1} \dots x_k, y_i) \xrightarrow{*} (q, x_{i+1} \dots x_k, \epsilon) \quad \text{for } i = 1, 2, \dots, k$$

None, of these sequences can exceed $(n-1)$ moves, so, $y_i \Rightarrow x_i$

Here each symbol of x_i matches against y_i and thus, $y_i \Rightarrow x_i$

Therefore, the derivation becomes,

$$A \Rightarrow Y_1 Y_2 Y_3 \dots Y_k$$

*

$$\Rightarrow x_1 Y_2 Y_3 \dots Y_k$$

.

.

.

*

$$\Rightarrow x_1 x_2 x_3 \dots x_k$$

*

$$(i.e) \quad A \Rightarrow x$$

Let $A = S$ and $x = \omega$, $N(P)$ is given as,

$$(q, \omega, S) \xrightarrow{*} (q, \in, \in) \quad [\text{since } S \Rightarrow \omega]$$

$\therefore \omega$ is in $L(G)$

Hence the theorem is proved by induction principle.

PROBLEMS

1. Find a PDA for the given grammar $S \rightarrow 0S1 \mid 00 \mid 11$

Solution:

$$\begin{aligned} M &= (\{q\}, T, VUT, \delta, q, S, \phi) \\ &= (\{q\}, \{0, 1\}, \{0, 1, S\}, \delta, q, S, \phi) \text{ where } \delta \text{ is} \\ \delta(q, \in, S) &= \{(q, 0S1), (q, 00), (q, 11)\} \\ \delta(q, 0, 0) &= \{(q, \in)\} \\ \delta(q, 1, 1) &= \{(q, \in)\} \end{aligned}$$

2. Convert the grammar to PDA that accepts the same language by empty stack.

$$S \rightarrow 0S1 \mid A$$

$$A \rightarrow 1A0 \mid S \mid \in$$

Solution:

a) **For each variable $A \in V$, include a transition**

$$\begin{aligned} \delta(q, \in, A) &\Rightarrow \{(q, \alpha) \mid A \rightarrow \alpha \text{ is a production in } G\} \\ \delta(q, \in, S) &\Rightarrow \{(q, 0S1), (q, A)\} \\ \delta(q, \in, A) &\Rightarrow \{(q, 1A0), (q, S), (q, \in)\} \end{aligned}$$

b) **For each terminal $a \in T$, include a transition**

$$\begin{aligned} \delta(q, a, a) &\Rightarrow \{(q, \in)\} \\ \delta(q, 0, 0) &\Rightarrow \{(q, \in)\} \\ \delta(q, 1, 1) &\Rightarrow \{(q, \in)\} \end{aligned}$$

Therefore the PDA is given by

$$M = (\{q\}, \{0, 1\}, \{S, A, 0, 1\}, \delta, q, S, \phi)$$

where δ is

$$\begin{aligned}\delta(q, \epsilon, S) &\Rightarrow \{(q, 0S1), (q, A)\} \\ \delta(q, \epsilon, A) &\Rightarrow \{(q, 1A0), (q, S), (q, \epsilon)\} \\ \delta(q, 0, 0) &\Rightarrow \{(q, \epsilon)\} \\ \delta(q, 1, 1) &\Rightarrow \{(q, \epsilon)\}\end{aligned}$$

3. Let G be the grammar given by

$$\begin{aligned}S &\rightarrow aABB \mid aAA \\ A &\rightarrow aBB \mid a \\ B &\rightarrow bBB \mid A\end{aligned}$$

Construct PDA for the above grammar

Solution:

The equivalent PDA, M is given by

$$M = (\{q\}, \{a, b\}, \{a, b, S, A, B\}, \delta, q, S, \phi)$$

where δ is given by

$$\begin{aligned}\delta(q, \epsilon, S) &\Rightarrow \{(q, aABB), (q, aAA)\} \\ \delta(q, \epsilon, A) &\Rightarrow \{(q, aBB), (q, a)\} \\ \delta(q, \epsilon, B) &\Rightarrow \{(q, bBB), (q, A)\} \\ \delta(q, a, a) &\Rightarrow \{(q, \epsilon)\} \\ \delta(q, b, b) &\Rightarrow \{(q, \epsilon)\}\end{aligned}$$

4. Let G be the grammar given by

$$\begin{aligned}S &\rightarrow 0BB \\ B &\rightarrow 0S \mid 1S \mid 0\end{aligned}$$

Construct a PDA. Test if 010^4 is in the language.

Solution:

The equivalent PDA, M is given by

$$M = (\{q\}, \{0, 1\}, \{0, 1, S, B\}, \delta, q, S, \phi)$$

where δ is

$$\delta(q, \epsilon, S) \Rightarrow \{(q, 0BB)\}$$

$$\delta(q, \epsilon, B) \Rightarrow \{(q, 0S), (q, 1S), (q, 0)\}$$

$$\delta(q, 0, 0) \Rightarrow \{(q, \epsilon)\}$$

$$\delta(q, 1, 1) \Rightarrow \{(q, \epsilon)\}$$

Acceptance of 010^4 by M

$$\begin{aligned}
 \delta(q, 010000, S) &\mapsto (q, 010000, 0BB) \\
 &\mapsto (q, 10000, BB) \\
 &\mapsto (q, 10000, 1SB) \\
 &\mapsto (q, 0000, SB) \\
 &\mapsto (q, 0000, 0BBB) \\
 &\mapsto (q, 000, BBB) \\
 &\mapsto (q, 000, 0BB) \\
 &\mapsto (q, 00, BB) \\
 &\mapsto (q, 00, 0B) \\
 &\mapsto (q, 0, B) \\
 &\mapsto (q, 0, 0) \\
 &\mapsto (q, \epsilon)
 \end{aligned}$$

The string 010^4 is accepted by $M \in L$.

5. Construct a PDA for the given grammar.

$$S \rightarrow 0AB$$

$$A \rightarrow 1A \mid 1$$

$$B \rightarrow 0B \mid 1A \mid 0$$

Solution:

$$M = (\{q\}, \{0, 1\}, \{0, 1, S, A, B\}, \delta, q, S, \phi)$$

where δ is given by

$$\delta(q, \epsilon, S) \Rightarrow \{(q, 0AB)\}$$

$$\delta(q, \epsilon, A) \Rightarrow \{(q, 1A), (q, 1)\}$$

$$\delta(q, \epsilon, B) \Rightarrow \{(q, 0B), (q, 1A), (q, 0)\}$$

$$\delta(q, 0, 0) \Rightarrow \{(q, \epsilon)\}$$

$$\delta(q, 1, 1) \Rightarrow \{(q, \epsilon)\}$$

6. Construct a PDA for the grammar

(AU – Dec 2007, Dec 2008)

$$S \rightarrow aB \mid bA; A \rightarrow a \mid aS \mid bAA; B \rightarrow b \mid bS \mid aBB$$

Solution:

The equivalent PDA, M is given by

$$M = (\{q\}, \{a, b\}, \{a, b, S, A, B\}, \delta, q, S, \phi)$$

where δ is given by

$$\begin{aligned}\delta(q, \epsilon, S) &\Rightarrow (\{q, aB\}, \{q, bA\}) \\ \delta(q, \epsilon, A) &\Rightarrow \{(q, a), (q, aS), \{q, bAA\}\} \\ \delta(q, \epsilon, B) &\Rightarrow \{(q, b), (q, bS), (q, aBB)\} \\ \delta(q, a, a) &\Rightarrow \{(q, \epsilon)\} \\ \delta(q, b, b) &\Rightarrow \{(q, \epsilon)\}\end{aligned}$$

7. Construct a PDA for $S \rightarrow aAA; A \rightarrow aS \mid bS \mid a$.

(AU – Dec 2008)

Solution:

The equivalent PDA, M is given by

$$M = (\{q\}, \{a, b\}, \{a, b, S, A\}, \delta, q, S, \phi)$$

where δ is given by

$$\begin{aligned}\delta(q, \epsilon, S) &\Rightarrow \{(q, aAA)\} \\ \delta(q, \epsilon, A) &\Rightarrow \{(q, aS), (q, bS), (q, a)\} \\ \delta(q, a, a) &\Rightarrow \{(q, \epsilon)\} \\ \delta(q, b, b) &\Rightarrow \{(q, \epsilon)\}\end{aligned}$$

8. Construct a PDA for $S \rightarrow aSbb \mid abb$.

(AU – June 2009)

Solution:

The equivalent PDA, M is given by

$$M = (\{q\}, \{a, b\}, \{a, b, S\}, \delta, q, S, \phi)$$

where δ is given by

$$\begin{aligned}\delta(q, \epsilon, S) &\Rightarrow \{(q, aSbb), (q, abb)\} \\ \delta(q, a, a) &\Rightarrow \{(q, \epsilon)\} \\ \delta(q, b, b) &\Rightarrow \{(q, \epsilon)\}\end{aligned}$$

9. Construct a PDA for $G = (\{S, A\}, \{a, b\}, P, S)$ with $P : S \rightarrow AA \mid a; A \rightarrow SA \mid b$.
 (AU – June 2009)

Solution:

The equivalent PDA, M is given by

$$M = (\{q\}, \{a, b\}, \{a, b, S, A\}, \delta, q, S, \phi)$$

where δ is given by

$$\begin{aligned}\delta(q, \epsilon, S) &\Rightarrow \{(q, AA), (q, a)\} \\ \delta(q, \epsilon, A) &\Rightarrow \{(q, SA), (q, b)\} \\ \delta(q, a, a) &\Rightarrow \{(q, \epsilon)\} \\ \delta(q, b, b) &\Rightarrow \{(q, \epsilon)\}\end{aligned}$$

From PDAs To Grammars

Theorem

Let $P = \{Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi\}$ be a PDA. Then there exists a context free grammar, G such that $L(G) = N(P)$.

Proof

Let $G = (V, T, P, S)$ where,

$V \rightarrow$ set of variables having

- (i) Special start symbol, S
- (ii) All symbols of the form $[p \times q]$, where p and q are in Q, X being the stack symbol in Γ .

$T \rightarrow$ set of terminals in Σ

$P \rightarrow$ production rules to the form,

- (i) $S \rightarrow [q_0 Z_0 P]$ [for $p \in Q$]
- (ii) $\delta[q, a, X]$ contain $(r, y_1, y_2 \dots y_k)$ where
 - a. $a \in \{\Sigma \cup \epsilon\}$
 - b. $k \geq 0$, in which the pair is (r, ϵ)

then, the production rules are given by

$$[q X r_k] \rightarrow [r Y_1 r_1] [r_1 Y_2 r_2] \dots [r_{k-1} Y_k r_k]$$

To prove

$$[q \ X \ p] \xrightarrow{*} \omega \text{ if and only if } (q, \omega, X) \xrightarrow{*} (p, \in, \in).$$

Both the if-case and only-if-case can be proved by induction principle.

IF-CASE

$$\text{Suppose } (q, \omega, X) \xrightarrow{*} (p, \in, \in)$$

To prove

$$[q \ X \ p] \xrightarrow{*} \omega \text{ by induction on the number of transitions made by the PDA.}$$

Basis of induction

Let $\omega \Rightarrow$ single symbol or \in

Then $\delta(q, \omega, X)$ contain (p, \in)

G is constructed as,

$$G \Rightarrow [q \ X \ p] \rightarrow \omega \text{ is a production}$$

$$\therefore [q \ X \ p] \xrightarrow{*} \omega$$

Thus basic of induction if proved

Inductive step

Assume that $(q, \omega, X) \xrightarrow{*} (p, \in, \in)$ consumes „n“ number of steps

The first move shall be,

$$(q, \omega, X) \xrightarrow{*} (r_0, x, Y_1 Y_2 Y_3 \dots Y_k)$$

$$\xrightarrow{*} (p, \in, \in)$$

Where

$$\omega = ax \quad [a \text{ in } \Sigma \text{ or } a = \in]$$

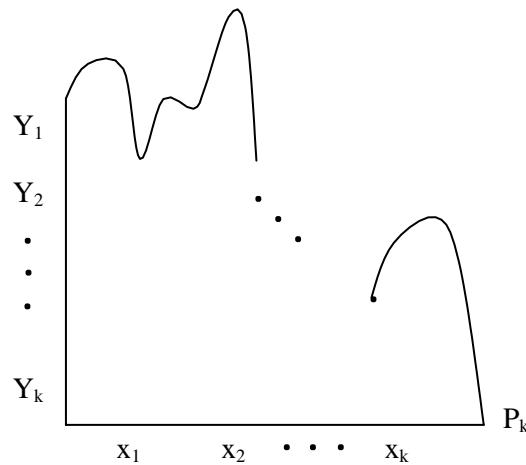
The pair $(r_0, X Y_1 Y_2 \dots Y_k)$ must be in $\delta(q, a, X)$

G can be constructed using,

$$[q \ X \ r_k] \rightarrow a [r_0 \ Y_1 \ r_1] [r_1 \ Y_2 \ r_2] \dots [r_{k-1} \ Y_k \ r_k]$$

where

1. $r_k = p$
2. $r_1, r_2, \dots, r_{k-1} \in Q$



PDA makes a sequence of moves that pops a symbol out of the stack

In the above diagram, each Y_1, Y_2, \dots, Y_k gets popped off the stack at each $P_1, P_2 \dots, P_k$ states.

Let $x = \omega_1, \omega_2, \dots, \omega_k$ where $w_i \rightarrow$ input symbol consumed while Y_i is popped off when $P = P_i$

$$\therefore (r, \omega, Y) \xrightarrow{*} (r, \in, \in)$$

This sequence of transitions can take at most „n“ moves, hence

$$[r_{i-1} Y_i r_i] \xrightarrow{*} \omega_i$$

Thus the final productions are given by

$$\begin{aligned} [q X r_k] &\rightarrow a [r_0 Y_1 r_1] [r_1 Y_2 r_2] \dots [r_{k-1} Y_k r_k] \\ &\xrightarrow{*} a \omega_1 [r_1 Y_2 r_2] [r_2 Y_3 r_3] \dots [r_{k-1} Y_k r_k] \\ &\xrightarrow{*} a \omega_1 \omega_2 [r_2 Y_3 r_3] \dots [r_{k-1} Y_k r_k] \\ &\xrightarrow{*} \vdots \\ &\Rightarrow a \omega_1 \omega_2 \dots \omega_k = \omega \end{aligned}$$

where, $r_k = p$

ONLY-IF-CASE

The proof is done on the induction on the number of steps in the derivations.

Basis of induction

This involves only one step of derivation

$$[q \ X \ p] \Rightarrow \omega$$

There is only one transition of P with X being popped out with the state change from q to P.

(i.e) (p, \in) must be in $\delta(q, a, X)$ [$a = \omega$]

Then, $(q, \omega, X) \xrightarrow{*} (p, \in, \in)$ [$\omega = \in$ and „X“ popped out]

Thus basis of induction is proved

Inductive step

Assume that $[q, X \ p] \Rightarrow \omega$ is derived in „n“ steps [$n > 1$]

The first derivations is of the form

$$\begin{aligned} [q \ X \ r_k] \rightarrow a \ [r_0 \ Y_1 \ r_1] \ [r_1 \ Y_2 \ r_2] \dots \ [r_{k-1} \ Y_k \ r_k] \\ \xrightarrow{*} \Rightarrow \omega \end{aligned}$$

Where $r_k = p$

Let ω be broken into $\omega = a \ \omega_1 \ \omega_2 \ \dots \ \omega_k$ such that,

$$\begin{aligned} \xrightarrow{*} \\ [r_{i-1} \ Y_i \ r_i] \Rightarrow \omega_i \quad [\text{for } i = 1, 2, \dots, k] \end{aligned}$$

Using inductive hypothesis on i,

$$\begin{aligned} \xrightarrow{*} \\ (r_{i-1}, \omega_i, Y_i) \xrightarrow{*} (r_i, \in, \in) \end{aligned}$$

The moves beyond ω_i

$$(r_{i-1}, \omega_1, \omega_2, \dots, \omega_{i-1}, Y_{i-1}, Y_i, Y_{i+1}, \dots, Y_k) \xrightarrow{*} (r_i, \omega_1, \omega_2, \dots, \omega_{i-1}, Y_{i-1}, Y_i, Y_{i+1}, \dots, Y_k)$$

The final sequence becomes

$$\begin{aligned} (q, aw_1 w_2 \dots w_k) \xrightarrow{*} (r_0, \omega_1 \omega_2 \dots \omega_k, Y_1, Y_2 \dots Y_k) \\ \xrightarrow{*} (r_1, \omega_2 \omega_3 \dots \omega_k, Y_2, Y_3 \dots Y_k) \end{aligned}$$

$$\vdash^*(r_2, \omega_3 \ \omega_4 \dots \ \omega_k, Y_3, Y_4 \dots Y_k)$$

$$\vdash^* (r_k, \in, \in)$$

Since, $r_k = p$

$$(q, \omega, X) \vdash^* (p, \in, \in)$$

Thus the theorem is proved by mathematical induction as,

$$S \xrightarrow{*} \omega \text{ if and only if } [q_0 \ Z_0 p] \xrightarrow{*} \omega$$

PROBLEMS

1. Convert PDA to CFG (AU – May 2004, 2007, 2008, Dec 2008, 2009, 2012)

$$M = (\{p, q\}, \{0, 1\}, \{x, z\}, \delta, q, z)$$

$$\delta(q, 1, z) = (q, xz)$$

$$\delta(q, 1, x) = (q, xx)$$

$$\delta(q, \in, x) = (q, \in)$$

$$\delta(q, 0, x) = (p, x)$$

$$\delta(p, 1, x) = (p, \in)$$

$$\delta(p, 0, z) = (q, z)$$

Solution:

$$(1) S \rightarrow [q^z q]$$

$$S \rightarrow [q^z p]$$

Using $S \rightarrow$ [Initial state ^{Initial stack symbol} States]

$$S \rightarrow [q^z Q]$$

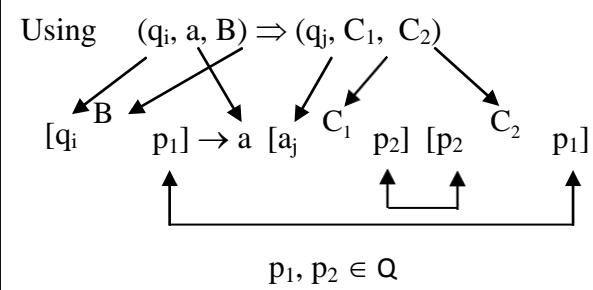
$$(2) \delta(q, 1, z) = (q, xz)$$

$$[q^z q] \rightarrow 1[q^x q] [q^z q]$$

$$[q^z q] \rightarrow 1[q^x p] [p^z q]$$

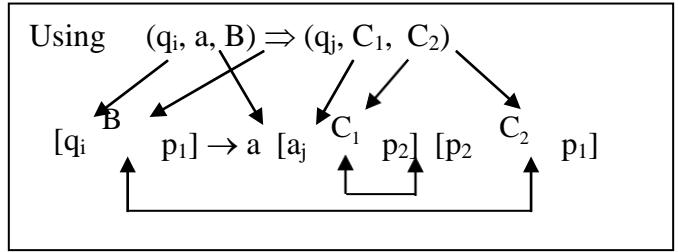
$$[q^z p] \rightarrow 1[q^x q] [q^z p]$$

$$[q^z p] \rightarrow 1[q^x p] [p^z p]$$



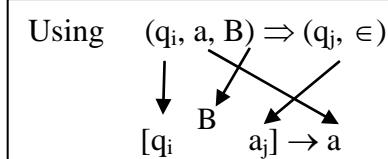
(3) $\delta(q, 1, x) = (q, xx)$

$$\begin{aligned}[q^x q] &\rightarrow 1[q^x q] [q^x q] \\[q^x q] &\rightarrow 1[q^x p] [p^x q] \\[q^x p] &\rightarrow 1[q^x q] [q^x p] \\[q^x p] &\rightarrow 1[q^x p] [p^x p]\end{aligned}$$



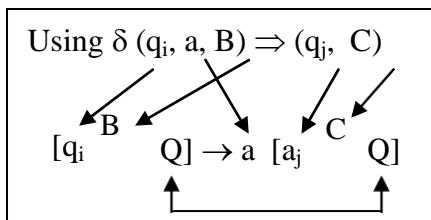
(4) $\delta(q, \epsilon, x) = (q, \epsilon)$

$$[q^x q] \rightarrow \epsilon$$



(5) $\delta(q, 0, x) = (p, x)$

$$\begin{aligned}[q^x q] &\rightarrow 0[p^x q] \\[q^x p] &\rightarrow 0[p^x p]\end{aligned}$$



(6) $\delta(p, 1, x) = (p, \epsilon)$

$$[p^x p] \rightarrow 1$$

[Using the rule state for (4)]

(7) $\delta(p, 0, z) = (q, z)$

$$\begin{aligned}[p^z q] &\rightarrow 0[q^z q] \\[p^z p] &\rightarrow 0[q^z p]\end{aligned}$$

[Using the rule stated for (5)]

Renaming variables

$[q^z q] \rightarrow A$	$[q^x q] \rightarrow E$
$[q^z p] \rightarrow B$	$[q^x p] \rightarrow F$
$[p^z q] \rightarrow C$	$[p^x q] \rightarrow G$
$[p^z p] \rightarrow D$	$[p^x p] \rightarrow H$

Production

$$S \rightarrow A \mid B \quad F \rightarrow 1EF \mid 1FH \mid 0H$$

$$A \rightarrow 1EA \mid 1FC$$

$$B \rightarrow 1EB \mid 1FD$$

$$H \rightarrow 1$$

$$C \rightarrow 0A$$

$$E \rightarrow 1EE \mid \epsilon \quad D \rightarrow 0B$$

$E \rightarrow 1FG \mid 0G \Rightarrow$ Eliminate the production



No production for G.

2. Give the CFG for the PDA,

(AU – Dec 2003, May 2005, Dec 2011)

$$M = (\{q_0, q_1\}, \{0, 1\}, \{z_0, x\}, \delta, q_0, z_0, \phi)$$

$$\delta(q_0, 1, z_0) = (q_0, xz_0)$$

$$\delta(q_0, 1, x) = (q_0, xx)$$

$$\delta(q_0, 0, x) = (q_1, x)$$

$$\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$$

$$\delta(q_1, 1, x) = (q_1, \epsilon)$$

$$\delta(q_1, 0, z_0) = (q_0, z_0)$$

Solution:

$$(1) S \rightarrow [q_0^{z_0} q_0]$$

Using $S \rightarrow$ [Initial state $\xrightarrow{\text{Initial stack symbol}}$ States]

$$S \rightarrow [q_0^{z_0} q_1]$$

$$S \rightarrow [q^z Q]$$

$$(2) \delta(q_0, 1, z_0) = (q_0, xz_0)$$

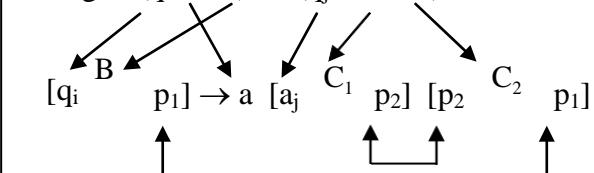
$$[q_0^{z_0} q] \rightarrow 1 [q_0^x q] [q_0^{z_0} q_0]$$

$$[q_0^{z_0} q_0] \rightarrow 1 [q_0^x q_1] [q_1^{z_0} q_0]$$

$$[q_0^{z_0} q_1] \rightarrow 1 [q_0^x q_0] [q_0^{z_0} q_1]$$

$$[q_0^{z_0} q_1] \rightarrow 1 [q_0^x q_1] [q_1^{z_0} q_1]$$

Using $(q_i, a, B) \Rightarrow (q_j, C_1, C_2)$



$p_1, p_2 \in Q$

$$(3) \delta(q_0, 1, x) = (q_0, xx)$$

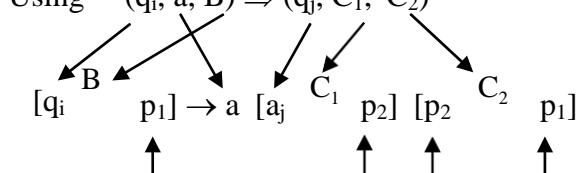
$$[q_0^x q_0] \rightarrow 1 [q_0^x q_0] [q_0^x q_0]$$

$$[q_0^x q_0] \rightarrow 1 [q_0^x q_1] [q_1^x q_0]$$

$$[q_0^x q_1] \rightarrow 1 [q_0^x q_0] [q_0^x q_1]$$

$$[q_0^x q_1] \rightarrow 1 [q_0^x q_1] [q_1^x q_1]$$

Using $(q_i, a, B) \Rightarrow (q_j, C_1, C_2)$



$p_1, p_2 \in Q$

- (4) $\delta(q_0, \emptyset, x) = (q_1, x)$
- $$[q_0 \xrightarrow{x} q_0] \rightarrow 1 [q_1 \xrightarrow{x} q_0]$$
- $$[q_0 \xrightarrow{x} q_1] \rightarrow 1 [q_1 \xrightarrow{x} q_1]$$
- (5) $\delta(q_0, \in, z_0) = (q_0, \in)$

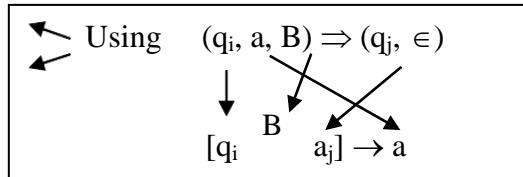
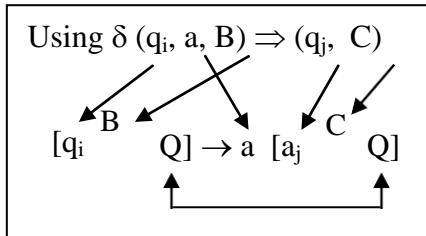
$$[q_0 \xrightarrow{z_0} q_1] \rightarrow \in$$

- (6) $\delta(q_1, 1, x) = (q_1, \in)$
- $$[q_1 \xrightarrow{x} q_1] \rightarrow 1$$

- (7) $\delta(q_1, \emptyset, z_0) = (q_0, z_0)$

$$[q_1 \xrightarrow{z_0} q_0] \rightarrow 0 [q_0 \xrightarrow{z_0} q_0]$$

$$[q_1 \xrightarrow{z_0} q_1] \rightarrow 0 [q_0 \xrightarrow{z_0} q_1]$$



Renaming the variable

$[q_0 \xrightarrow{z_0} q_0] \rightarrow A$	$[q_0 \xrightarrow{x} q_0] \rightarrow E$
$[q_0 \xrightarrow{z_0} q_1] \rightarrow B$	$[q_0 \xrightarrow{x} q_1] \rightarrow F$
$[q_1 \xrightarrow{z_0} q_0] \rightarrow C$	$[q_1 \xrightarrow{x} q_0] \rightarrow G$
$[q_1 \xrightarrow{z_0} q_1] \rightarrow D$	$[q_1 \xrightarrow{x} q_1] \rightarrow H$

Productions

$S \rightarrow A \mid B$	$F \rightarrow 1EF \mid 1FH \mid 0H$
$A \rightarrow 1EA \mid 1FC$	$H \rightarrow 1$
$B \rightarrow 1EB \mid 1FD \mid \in$	$C \rightarrow 0A$
$E \rightarrow 1EE \mid \boxed{1FG} \mid \boxed{0G}$	$D \rightarrow 0B$

Since G has no productions, 1FG and 0G are eliminated from E.

Final productions

$S \rightarrow A \mid B$	$F \rightarrow 1EF \mid 1FH \mid 0H$
$A \rightarrow 1EA \mid 1FC$	$H \rightarrow 1$
$B \rightarrow 1EB \mid 1FD \mid \in$	$C \rightarrow 0A$
$E \rightarrow 1EE$	$D \rightarrow 0B$

3. For the PDA, $(\{q_0, q_1\}, \{0, 1\}, \{0, 1, z_0\}, \delta, q_0, z_0, \phi)$ where δ is

$$\begin{aligned}\delta(q_0, \epsilon, z_0) &= (q_1, \epsilon) \\ \delta(q_0, 0, z_0) &= (q_0, 0z_0) \\ \delta(q_0, 0, 0) &= (q_0, 00) \\ \delta(q_0, 1, 0) &= (q_0, 10) \\ \delta(q_0, 1, 1) &= (q_0, 11) \\ \delta(q_0, 0, 1) &= (q_1, \epsilon) \\ \delta(q_1, 0, 1) &= (q_1, \epsilon) \\ \delta(q_1, 0, 0) &= (q_1, \epsilon) \\ \delta(q_1, \epsilon, z_0) &= (q_1, \epsilon)\end{aligned}$$

Obtain a CFG accepting the above PDA.

Solution:

(1) Production for the start symbol

Using the rule, $S \rightarrow [\text{Start symbol}]$ Initial stack symbol States in Q]

$$S \rightarrow [q_0 \underset{z_0}{\overset{\longrightarrow}{|}} q_0] \quad (1)$$

$$S \rightarrow [q_0 \underset{z_0}{\overset{\longrightarrow}{|}} q_1] \quad (2)$$

(2) Production for $\delta(q_0, \epsilon, z_0) = (q_1, \epsilon)$

Rule: $\delta(q_i, a, B) \Rightarrow (q_j, \epsilon)$

$$\begin{array}{c} \swarrow \\ [q_i \underset{B}{\overset{\longrightarrow}{|}} q_j] \rightarrow a \end{array}$$

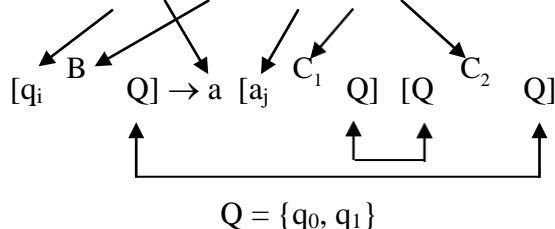
$\therefore \delta(q_0, \epsilon, z_0) \Rightarrow (q_1, \epsilon)$ becomes

$$\begin{array}{c} \swarrow \\ [q_0 \underset{z_0}{\overset{\longrightarrow}{|}} q_1] \rightarrow \epsilon \end{array} \quad (3)$$

(3) Production for $\delta(q_0, 0, z_0) = (q_0, \underline{Q} z_0)$

$C_1 C_2$

Rule $\delta(q_i, a, B) \Rightarrow (q_j, C_1, C_2)$





$\therefore \delta(q_0, 0, z_0) = (q_0, 0 z_0)$ becomes

$$[q_0^z_0 q_0] \rightarrow 0 [q_0^0 q_0] [q_0^z_0 q_0] \quad \text{----- (4)}$$

$$[q_0^z_0 q_0] \rightarrow 0 [q_0^0 q_1] [q_1^z_0 q_0] \quad \text{----- (5)}$$

$$[q_0^z_0 q_1] \rightarrow 0 [q_0^0 q_0] [q_0^z_0 q_1] \quad \text{----- (6)}$$

$$[q_0^z_0 q_1] \rightarrow 0 [q_0^0 q_1] [q_1^z_0 q_1] \quad \text{----- (7)}$$

$$(4) \text{ Production for } \delta(q_0, 0, 0) = (q_0, \underline{0} \ \underline{0}) \\ C_1 \ C_2$$

Using the rule applied for (3), we get

$$[q_0^0 q_0] \rightarrow 0 [q_0^0 q_0] [q_0^0 q_0] \quad \text{----- (8)}$$

$$[q_0^0 q_0] \rightarrow 0 [q_0^0 q_1] [q_1^0 q_0] \quad \text{----- (9)}$$

$$[q_0^0 q_1] \rightarrow 0 [q_0^0 q_0] [q_0^0 q_1] \quad \text{----- (10)}$$

$$[q_0^0 q_1] \rightarrow 0 [q_0^0 q_1] [q_1^0 q_1] \quad \text{----- (11)}$$

$$(5) \text{ Productions for } \delta(q_0, 1, 0) = (q_0, \underline{1} \ \underline{0}) \\ C_1 \ C_2$$

Using the rule stated for (3), we get

$$[q_0^0 q_0] \rightarrow 1 [q_0^1 q_0] [q_0^0 q_0] \quad \text{----- (12)}$$

$$[q_0^0 q_0] \rightarrow 1 [q_0^1 q_1] [q_1^0 q_0] \quad \text{----- (13)}$$

$$[q_0^0 q_1] \rightarrow 1 [q_0^1 q_0] [q_0^0 q_1] \quad \text{----- (14)}$$

$$[q_0^0 q_1] \rightarrow 1 [q_0^1 q_1] [q_1^0 q_1] \quad \text{----- (15)}$$

$$(6) \text{ Production for } \delta(q_0, 1, 1) = (q_0, \underline{1} \ \underline{1}) \\ C_1 \ C_2$$

Using the rule stated for (3), we get

$$[q_0^1 q_0] \rightarrow 1 [q_0^1 q_0] [q_0^1 q_0] \quad \text{----- (16)}$$

$$[q_0^1 q_0] \rightarrow 1 [q_0^1 q_1] [q_1^1 q_0] \quad \text{----- (17)}$$

$$[q_0^1 q_1] \rightarrow 1 [q_0^1 q_0] [q_0^1 q_1] \quad \text{----- (18)}$$

$$[q_0^1 q_1] \rightarrow 1 [q_0^1 q_1] [q_1^1 q_1] \quad \text{----- (19)}$$

(7) Production for $\delta(q_0, 0, I) = (q_1, \sqsubseteq)$

Using the rule stated in (2), we get

$$[q_0 \stackrel{1}{q_0}] \rightarrow 0 \quad \dots \quad (20)$$

(8) Production for $\delta(q_1, 0, I) = (q_1, \sqsubseteq)$

Using the rule specified in (2)

$$[q_1 \stackrel{1}{q_1}] \rightarrow 0 \quad \dots \quad (21)$$

(9) Production for $\delta(q_1, 0, 0) = (q_1, \in)$

Using the rule specified in (2)

$$[q_1 \stackrel{0}{q_1}] \rightarrow 0 \quad \dots \quad (22)$$

(10) Production for $\delta(q_1, \in, z_0) = (q_1, \sqsubseteq)$

Using the rule stated for (3), we get

$$[q_1 \stackrel{z_0}{q_1}] \rightarrow \in \quad \square \quad (23)$$

Simplification of the grammar

Let,	$[q_0 \stackrel{z_0}{q_0}] \rightarrow A$	$[q_1 \stackrel{z_0}{q_0}] \rightarrow C$
	$[q_0 \stackrel{z_0}{q_1}] \rightarrow B$	$[q_1 \stackrel{z_0}{q_1}] \rightarrow D$
	$[q_0 \stackrel{z_0}{q_0}] \rightarrow E$	$[q_1 \stackrel{0}{q_0}] \rightarrow G$
	$[q_0 \stackrel{0}{q_1}] \rightarrow F$	$[q_1 \stackrel{0}{q_1}] \rightarrow H$
	$[q_0 \stackrel{1}{q_0}] \rightarrow I$	$[q_1 \stackrel{1}{q_0}] \rightarrow K$
	$[q_0 \stackrel{1}{q_1}] \rightarrow J$	$[q_1 \stackrel{1}{q_1}] \rightarrow L$

By substituting the renamed variables in the productions, we get,

$$S \rightarrow A \mid B$$

$$B \rightarrow \in$$

$$A \rightarrow 0EA \mid 0FC$$

$$B \rightarrow 0EB \mid 0FD$$

$$E \rightarrow 0EE \mid 0FG$$

$$F \rightarrow 0EF \mid 0FH$$

$E \rightarrow 1IE \mid 1JG$
 $F \rightarrow 1IF \mid 1JH$
 $I \rightarrow 1II \mid 1JK$
 $J \rightarrow 1IJ \mid 1JL$
 $J \rightarrow 0$
 $L \rightarrow 0$
 $H \rightarrow 0$
 $D \rightarrow \epsilon$

After removing the ϵ -productions

The nullable set = {D, B, S}

$S \rightarrow A \mid B$
 $A \rightarrow 0EA \mid 0FC$
 $B \rightarrow 0EB \mid 0FD \mid 0E \mid 0F$
 $E \rightarrow 0EE \mid 0FG \mid 1IE \mid 1JG$
 $F \rightarrow 0EF \mid 0FH \mid 1IF \mid 1JH$
 $H \rightarrow 0$
 $I \rightarrow 1II \mid 1JK$
 $J \rightarrow 1IJ \mid 1JL \mid 0$
 $L \rightarrow 0$

Removing the non-generating symbols

Non – generating symbols = {A, C, D, E, G, I}

Resultant productions

$S \rightarrow B$
 $B \rightarrow 0F$
 $F \rightarrow 0FH \mid 1JH$
 $H \rightarrow 0$
 $J \rightarrow 1JL \mid 0$
 $L \rightarrow 0$

Removing unit productions

Here $S \rightarrow B$ is a unit production that has to be removed.

Final productions

$$S \rightarrow 0F$$

$$F \rightarrow 0FH \mid 1JH$$

$$H \rightarrow 0$$

$$J \rightarrow 1JL \mid 0$$

$$L \rightarrow 0$$

The production generates a language of the form,

$$L = \{0^n 1^m 0^{n+m} \mid n, m \geq 1\}$$

UNIT - 4

PROPERTIES OF CONTEXT FREE LANGUAGES

SIMPLIFICATION OF CFG

- The need for simplifying CFG is to make it easier to analyze and prove facts about CFL.
- CFGs are simplified to study and analyze the closure and decision properties of CFLs.
- Pumping lemma for CFL can also be proved easily with simplified grammar.
- Simplification of CFG requires transforming CFG into an equivalent form that satisfies certain condition/limitations on its form.
- The preliminary simplifications, which are applied on grammars to convert them to normal forms are,
 - Elimination of useless symbols
 - Elimination of ϵ productions
 - Elimination of unit productions

Elimination of Useless Symbols

(Nov/Dec 2007)

- Symbols are of two types
 - Useless
 - Useful
- Useful symbols are the variables and terminals that have some production that leads from the starting symbol to a string of the type $S \xrightarrow{*} \omega$.
- Consider the derivation

$$S \xrightarrow{*} \alpha X \xrightarrow{*} \beta \Rightarrow \omega \text{ where } \omega \in T^*$$

- Here α, β are any terminals and X can be either terminal or variable.
- If X has no production, (if $X \in V$) then X is a useless symbol.
- If X has some production / if X leads to a string, with zero/more substitution, then it is a useful symbol.

- Thus, useless symbols are those variables/terminals that do not appear in any derivation of a terminal string from the start string.

Types of useless symbols

- Non – generating symbols
- Non-reachable symbols

Non generating symbols

- A symbol X which is an element of V [variable] is generating symbol if $X \xrightarrow[G]{\omega}^*$, where $\omega \in T^*$.
- Generating symbols are variables that generate a string containing terminals.
- A non generating symbol is one that cannot generate any terminal.
- The presence of production with non-generating symbol is useless.

Example

$$S \rightarrow aA \mid bB \mid a \mid b$$

$$A \rightarrow Aa \mid a$$

$$B \rightarrow bB$$

Here $B \rightarrow bB$ is a recursive grammar that its substitution to S results in endless derivation and doesnot give lead to any string of terminals.

$$S \Rightarrow b\underline{B} \Rightarrow bb\underline{B} \Rightarrow bbb\underline{B} \Rightarrow bbbb\underline{B} \Rightarrow \dots$$

Finding non generating symbols

- All the terminals symbols are generating symbols.
- If there is a production rule, say $X \rightarrow \alpha$ and if $\alpha \in (VUT)^*$, then A is a generating symbol.
- A symbol that is not in the set of generating symbols is said to be non-generating symbol.

Eliminating Non-Generating Symbols

- The only way to simplify a grammar containing non-generating symbol is to eliminate such symbols directly from the grammar.

Example

Consider the example given above. After eliminating the production for $B \rightarrow bB$, the resulting grammar includes.

$S \rightarrow aA | a | b \Rightarrow$ Eliminating the variable B

$A \rightarrow aA | a$

Non reachable symbols

- A symbol is said to be reachable if and only if it can be reached from the start symbol.

If $\underset{G}{\overset{*}{S \Rightarrow \alpha}} , \text{ where } \alpha \text{ contains } X [X \in V]$

- Then X is said to be a reachable symbol.
- If the derivation from the start symbol to a string of terminals does not depend on a variable X then X is non-reachable symbol.

Example

Let

$S \rightarrow aAa$

$A \rightarrow Sb | bCC$

$C \rightarrow abb$

$E \rightarrow aC$

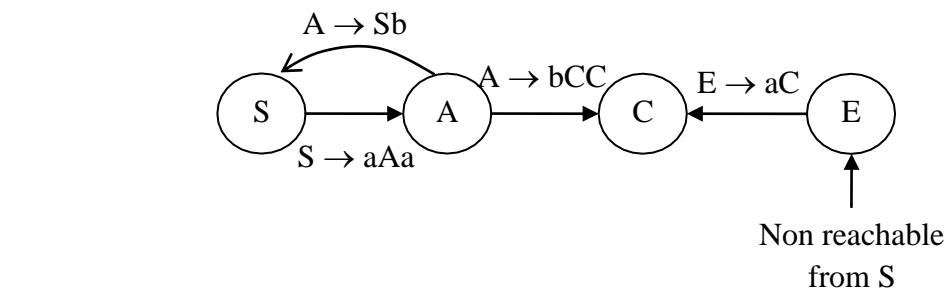
- In the given grammar, S depends on A, A depends on S and C, E depends on C, and C does not depend on any variable.
 - Here E is the only production which is not being depended by any other productions.
- $E \rightarrow$ Non reachable symbol

Finding non reachable symbols

- Dependency graph is used to identify the non-reachable symbols.
- A dependency graph is drawn for all the production rules.
- If there is no path from the start symbol to any variable X then X is non-reachable symbol.

Example

Consider the above mentioned problem. The dependency graph is given by



Here all the variables are having a directed edge from the starting symbol, S except the variable, E.

Elimination of Non-Reachable Symbols

A grammar containing non-reachable symbol can be simplified by deleting all the production containing the non reachable symbol.

Example

After eliminating the non-reachable symbol, E we get the following grammars as,

$$\begin{aligned} S &\rightarrow aAa \\ A &\rightarrow Sb \mid bCC \\ C &\rightarrow abb \end{aligned}$$

Problems on elimination of useless symbols

1. Eliminate the non generating symbols from the given grammar.

(AU – Nov / Dec 2008)

$$\begin{aligned} S &\rightarrow AB \mid CA \\ B &\rightarrow BC \mid AB \\ C &\rightarrow aB \mid b \\ A &\rightarrow a \end{aligned}$$

Solution:

Here $A \rightarrow a$, $C \rightarrow b$, hence A and C are generating.

The production with A and C is $S \rightarrow CA$.

Hence eliminate other productions except A, C and S.

The final grammar is given as

$$\begin{aligned} S &\rightarrow CA \\ A &\rightarrow a \\ C &\rightarrow b \end{aligned}$$

2. Eliminate the non generating symbols from the given grammar.

$$P = \{ S \rightarrow aAa, A \rightarrow Sb \mid bCC, C \rightarrow abb, E \rightarrow aC \}$$

Solution:

Every production contains terminal symbols and $S \rightarrow A \rightarrow C$, C contains only terminals and $E \rightarrow C$.

Hence all the symbols are generating.

$$\therefore P = \{ S \rightarrow aAa, A \rightarrow Sb \mid bCC, C \rightarrow abb, E \rightarrow aC \}$$

3. Find and eliminate the non generating symbols from

$$\begin{aligned} S &\rightarrow 0A0 \\ A &\rightarrow S1|1CC|D0A \\ C &\rightarrow 011|DD \\ D &\rightarrow 0D0 \\ E &\rightarrow 0C \end{aligned}$$

Solution:

Given: $S \rightarrow A \rightarrow C \rightarrow 011$ contains terminals, $E \rightarrow C \rightarrow 011$.

The symbols S, A, C, E are all generating.

$D \rightarrow 0D0$ is recursive grammar that will never end.

Hence eliminate the non generating symbol, D and the productions are given as,

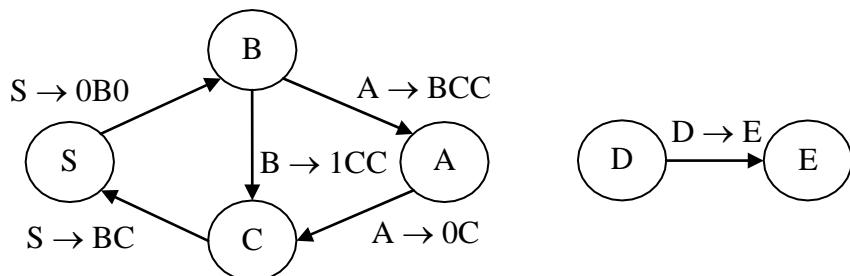
$$\begin{aligned} S &\rightarrow 0A0 \\ A &\rightarrow S1|1CC \quad // D \text{ is eliminated} \\ C &\rightarrow 011| \quad // D \text{ is eliminated} \\ E &\rightarrow 0C \end{aligned}$$

4. Eliminate the non reachable symbols from the given grammar.

$$P = \{ S \rightarrow 0B0|BC, A \rightarrow 0C|BCC, C \rightarrow 0, B \rightarrow 1CC, D \rightarrow E, E \rightarrow 1 \}$$

Solution:

We generate dependency graph to identify the non-reachable symbols.



From the above graph, $S \rightarrow B \rightarrow C$, $S \rightarrow C$ are all reachable from S.

But A, D, E are non reachable from S since they doesn't contain any production from S.

Hence eliminate A, D, E and the resultant production includes

$$S \rightarrow 0B0$$

$$C \rightarrow 0$$

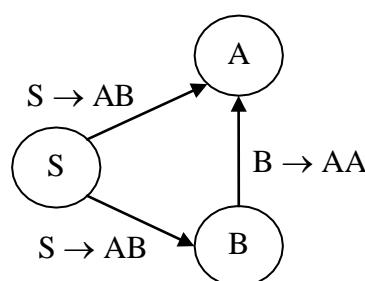
$$B \rightarrow 1CC$$

5. Find out and eliminate the non-reachable symbols from the grammar.

$$P = \{S \rightarrow 0S \mid AB, A \rightarrow 1A, B \rightarrow AA\}$$

Solution:

The dependency graph is given as



Since $S \rightarrow A$ and $S \rightarrow B$, S, A, B are all reachable from S.

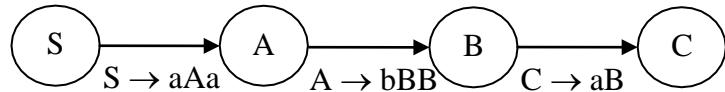
Hence no elimination is required and the resultant grammar is given as

$$P = \{0S \mid AB, A \rightarrow 1A, B \rightarrow AA\}$$

6. Eliminate the non - reachable symbols from the grammar, $P = \{S \rightarrow aAa, A \rightarrow bBB, B \rightarrow ab, C \rightarrow aB\}$

Solution:

The dependency graph is given as



From the above graph,

$$S \rightarrow A \rightarrow B$$

But C is a non-reachable symbol that doesn't have any derivation from S.

Hence eliminate C and the resulting grammar includes,

$$P = \{S \rightarrow aAa, A \rightarrow bBB, B \rightarrow ab\}$$

7. Consider $P = \{S \rightarrow aAa, A \rightarrow aS \mid bD, B \rightarrow aBa \mid b, C \rightarrow abb \mid DD, D \rightarrow aDa\}$. Find the useless symbols and eliminate them.

Solution:

From the grammar, D is non - generating since it is recursive in nature as $D \rightarrow aDa$. Hence eliminate D.

Therefore, the production rules are

$$S \rightarrow aAa$$

$$A \rightarrow aS$$

$$B \rightarrow aBa \mid b$$

$$C \rightarrow abb$$

Elimination of Null Production (ϵ)

- Null productions are those productions that are of the form: $X \rightarrow \epsilon$.
- These are also called as ϵ - productions.

Example

$$S \rightarrow aS \mid \epsilon$$

Here $S \rightarrow aS$ and $S \rightarrow \epsilon$ are the productions $S \rightarrow aS$ doesn't contain null production. But $S \rightarrow \epsilon$ is an epsilon production.

S is said to be nullable variable.

Finding and handling ϵ productions

- Find the nullable variables, which are of the form $X \rightarrow \epsilon$.
- The nullable variable used in the productions is removed.
- Remove the nullable production.

Example

For the above example $S \rightarrow aS \mid \epsilon$, the nullable variable is found to be S since $S \rightarrow \epsilon$.

By eliminating the nullable variable, the production becomes,

$S \rightarrow aS$ and on applying $S \rightarrow \epsilon$ on $S \rightarrow aS$, then S becomes $S \rightarrow a$.

Therefore, $S \rightarrow aS \mid a$ is the final production.

Problems on elimination of ϵ productions

- 1. For the productions $S \rightarrow ABA, A \rightarrow aA | \epsilon, B \rightarrow bB | \epsilon$, find the nullable symbols.**

Solution:

Since $A \rightarrow \epsilon$, A is identified as a nullable variable.

Similarly $B \rightarrow \epsilon$, hence B is also a nullable symbol.

Since $S \rightarrow ABA$, then S is also nullable variable since A and B are nullable symbols.

\therefore Set of nullable symbols = {S, A, B}

Removal of the variables S, A, and B results in non-existence of the grammar / severe effect on the grammar.

- 2. Remove ϵ productions from the grammar, $P = \{S \rightarrow ABA, A \rightarrow \epsilon, B \rightarrow \epsilon\}$**

Solution:

Given: $A \rightarrow \epsilon, B \rightarrow \epsilon$, the $S \rightarrow ABA \rightarrow \epsilon$.

Therefore A, B and S are nullable variables

After eliminating $A \rightarrow \epsilon, B \rightarrow \epsilon$, we get

$S \rightarrow ABA | AB | AA | BA | A | B | \underline{\epsilon} \rightarrow$ can be eliminated

$A \rightarrow \epsilon$

$B \rightarrow \epsilon \rightarrow$ eliminated

- 3. Find the nullable symbols from the following grammar, $P = \{S \rightarrow aS | AB, A \rightarrow \epsilon, B \rightarrow \epsilon\}$ and generate the grammar after addition of effect of nullable symbols.**

Solution:

Since $A \rightarrow \epsilon, B \rightarrow \epsilon, S \rightarrow aS | AB \rightarrow \epsilon$

The nullable set = {A, B, S}

S becomes,

$S \rightarrow aS | a | AB | A | B | \epsilon$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

After removing ϵ productions, the grammar becomes,

$S \rightarrow aS | a$

4. Remove ϵ - productions from

$$S \rightarrow ABA, A \rightarrow 0A \mid \epsilon, B \rightarrow 1B \mid \epsilon$$

Solution:

Here $A \rightarrow \epsilon$, and $B \rightarrow \epsilon$ and hence $S \rightarrow ABA \rightarrow \epsilon$

\therefore The nullable variable $A \rightarrow \epsilon, B \rightarrow \epsilon$.

Hence S becomes

$$\begin{aligned} S &\rightarrow ABA \mid AB \mid BA \mid AA \mid A \mid B \\ A &\rightarrow 0A \mid 0 \\ B &\rightarrow 1B \mid 1 \end{aligned}$$

5. Remove ϵ - productions from the below given grammar.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aAA \mid \epsilon \\ B &\rightarrow bBB \mid \epsilon \end{aligned}$$

Solution:

The nullable set = {A, B, S}

After removing $A \rightarrow \epsilon, B \rightarrow \epsilon$, the grammar becomes,

$$\begin{aligned} S &\rightarrow AB \mid A \mid B \\ A &\rightarrow aAA \mid aA \mid a \\ B &\rightarrow bBB \mid bB \mid b \end{aligned}$$

6. Consider the grammar

$$\begin{aligned} S &\rightarrow ABC \\ A &\rightarrow BC \mid a \\ B &\rightarrow bAC \mid \epsilon \\ C &\rightarrow cAB \mid \epsilon \end{aligned}$$

Identify the nullable symbols and eliminate them.

Solution:

Given: $B \rightarrow \epsilon, C \rightarrow \epsilon, A \rightarrow BC \rightarrow \epsilon, S \rightarrow ABC \rightarrow \epsilon$ the nullable variable are {S, A, B, C}

After removing the nullable symbols, the productions become

$$S \rightarrow ABC \mid AB \mid BC \mid AC \mid A \mid B \mid C$$

$$A \rightarrow BC \mid B \mid C \mid a$$

$$B \rightarrow bAC \mid bA \mid bC \mid b$$

$$C \rightarrow cAB \mid cA \mid cB \mid C$$

Elimination of unit productions

- The unit productions are those productions of the form $X \rightarrow Y$ where X and Y are variables of the grammar.
- The left hand side and right hand side contains only one variable each side.

Example

$$S \rightarrow A \mid bb$$

$$A \rightarrow B \mid b$$

$$B \rightarrow S \mid a$$

Here

$$S \rightarrow A,$$

$$A \rightarrow B,$$

$B \rightarrow S$ are all the unit productions.

Finding and eliminating unit productions

- Identify the non-unit productions in the grammar. Let it be $Y \rightarrow \alpha$, where α is a terminal.
- Select the unit production, $X \rightarrow Y$ where X and Y are variables.
- Add the production $X \rightarrow \alpha$ to the grammar since $X \rightarrow Y$ and $Y \rightarrow \alpha$.
- Remove the unit production, $X \rightarrow Y$ from the grammar.

Example

$$S \rightarrow A \mid bb$$

$$A \rightarrow B \mid b$$

$$B \rightarrow S \mid a$$

The unit productions are

$$S \rightarrow A$$

$$A \rightarrow B$$

$$B \rightarrow S$$

4.1 Theory of Computation

Elimination of unit productions

Since $S \rightarrow A$ and $A \rightarrow B$ and $B \rightarrow S$, we shall make it as $S \rightarrow A \rightarrow B \rightarrow S$.

Consider $S \rightarrow A | bb$. To eliminate A, substitute A's production „b“ [terminal] and since $A \rightarrow B$, substitute B's production „a“ [terminal]

Do not consider the unit productions while substitution.

$$\therefore S \rightarrow bb | b | a$$

Similarly, to eliminate B on $A \rightarrow B | b$, substitute B's production „a“ and since $B \rightarrow S$, substitute $S \rightarrow bb$.

$$\therefore A \rightarrow b | a | bb$$

Finally, to eliminate S on $B \rightarrow S | a$, substitute S's production „bb“ and since $S \rightarrow A$, substitute $A \rightarrow b$

$$\therefore B \rightarrow a | bb | b$$

Thus the final productions are

$$\begin{aligned}S &\rightarrow bb | b | a \\A &\rightarrow b | a | bb \\B &\rightarrow a | bb | b\end{aligned}$$

Problems on eliminating unit productions

1. Eliminate the unit production from the grammar, $P = \{S \rightarrow ABA | BA | AA | AB | A | B, A \rightarrow 0A | 0, B \rightarrow 1B | 1\}$

Solution:

The unit productions are

$$S \rightarrow A$$

$$S \rightarrow B$$

The non unit productions are

$$S \rightarrow ABA | BA | AA | AB | A | B,$$

$$A \rightarrow 0A | 0,$$

$$B \rightarrow 1B | 1$$

Since $S \rightarrow A$, we shall use A's productions $A \rightarrow 0A | 0$ to S.

Similarly, $S \rightarrow B$, we shall use B's productions $B \rightarrow 1B | 1$ to S.

Thus the grammar becomes

$$\begin{aligned} S &\rightarrow ABA \mid BA \mid AA \mid AB \mid 0A \mid 0 \mid 1B \mid 1, \\ A &\rightarrow 0A \mid 0 \\ B &\rightarrow 1B \mid 1 \end{aligned}$$

2. Eliminate unit productions from the grammar

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid I \\ I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{aligned}$$

Solution:

The unit productions in the given grammar are,

$$\begin{aligned} E &\rightarrow T \\ T &\rightarrow F \\ F &\rightarrow I \end{aligned}$$

The non unit productions are

$$\begin{aligned} E &\rightarrow E + T \\ T &\rightarrow T * F \\ F &\rightarrow (E) \\ I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{aligned}$$

After removing the unit productions, we get

$$\begin{aligned} E &\rightarrow E + T \mid T * F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ T &\rightarrow T * F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F &\rightarrow (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{aligned}$$

3. Simplify the following grammar

$$\begin{aligned} S &\rightarrow ASB \mid \epsilon \\ A &\rightarrow aAS \mid a \\ B &\rightarrow SbS \mid A \mid bb \end{aligned}$$

Solution:

Elimination of null production

The epsilon production in the grammar is $S \rightarrow \epsilon$ and the variable S has its impact on A and B productions.

4.1 Theory of Computation

The nullable variable are {S, A, B}

After removing $S \rightarrow \epsilon$, we get the final productions as

$$\begin{aligned} S &\rightarrow ASB \mid AB & [\text{since } S \rightarrow \epsilon, \text{ exclude } S] \\ A &\rightarrow aAS \mid aA \mid a \\ B &\rightarrow SbS \mid Sb \mid bs \mid b \mid A \mid bb \end{aligned}$$

Elimination of unit productions

The unit production in the above grammar is $B \rightarrow A$.

Hence eliminate the unit production in B and substitute the corresponding production of A in B. We get,

$$\begin{aligned} S &\rightarrow ASB \mid AB \\ A &\rightarrow aAS \mid aA \mid a \\ B &\rightarrow SbS \mid Sb \mid bs \mid b \mid bb \mid aAS \mid aA \mid b \end{aligned}$$

Since the grammar doesn't contain any useless symbols, the above production becomes the final one.

4. Simplify the following grammar

$$\begin{aligned} S &\rightarrow aAa \mid bBb \mid BB \\ A &\rightarrow C \\ B &\rightarrow S \mid A \\ C &\rightarrow S \mid \epsilon \end{aligned}$$

Solution:

Elimination of ϵ - productions

The nullable symbol is C since $C \rightarrow \epsilon$.

After removing $C \rightarrow \epsilon$, the grammar becomes,

$$\begin{aligned} S &\rightarrow aAa \mid bBb \mid aa \mid bb \mid BB \mid B & [\text{Since } C \rightarrow S \mid \epsilon] \\ A &\rightarrow C \\ B &\rightarrow S \mid A \\ C &\rightarrow S \end{aligned}$$

Elimination of unit productions

The unit productions in the above grammar are

$$\begin{aligned} S &\rightarrow B \\ A &\rightarrow C \end{aligned}$$

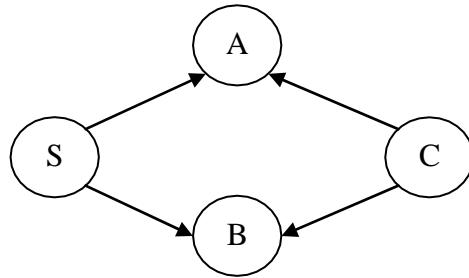
$B \rightarrow S$
 $B \rightarrow A$
 $C \rightarrow S$

The grammar without the above unit productions are

$S \rightarrow aAa \mid aa \mid bBb \mid bb \mid BB$
 $A \rightarrow aAa \mid aa \mid bBb \mid bb \mid BB$ [since $A \rightarrow C \rightarrow S$]
 $B \rightarrow aAa \mid aa \mid bBb \mid bb \mid BB$ [since $B \rightarrow A \rightarrow C \rightarrow S$ and $B \rightarrow S$]
 $C \rightarrow aAa \mid aa \mid bBb \mid bb \mid BB$ [since $C \rightarrow S$]

Elimination of non - generating symbols

From the above grammar



C is a non generating symbol, since it is has no derivation with respect to the start symbol.
Hence $C \rightarrow aAa \mid aa \mid bBb \mid bb \mid BB$ is removed.

5. Simplify the grammar

$S \rightarrow A1$

$A \rightarrow 0$

$B \rightarrow C \mid 1$

$C \rightarrow D$

$D \rightarrow E$

$E \rightarrow 0$

Solution:

There is no ϵ -production in the given grammar.

Elimination of unit productions

The unit productions in the grammar are

$B \rightarrow C$

$$C \rightarrow D$$

$$D \rightarrow E$$

After eliminating the unit productions, we get

$$S \rightarrow A1$$

$$A \rightarrow 0$$

$$E \rightarrow 0$$

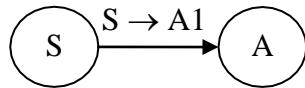
$$B \rightarrow 0 \quad [\text{since } B \rightarrow C \rightarrow D \rightarrow E \rightarrow 0]$$

$$C \rightarrow 0 \quad [\text{since } C \rightarrow D \rightarrow E \rightarrow 0]$$

$$D \rightarrow 0 \quad [\text{since } D \rightarrow E \rightarrow 0]$$

Elimination of useless symbols

Every symbol in the grammar is generating.



From the above dependency graph, S and A are reachable symbols.

The symbols other than A and S are non-generating.

They are {B, C, D, E}.

Eliminating the non-generating symbols, we get

$$S \rightarrow A1$$

$$A \rightarrow 0$$

6. Simplify the grammar equivalent to

$$S \rightarrow aC \mid SB$$

$$A \rightarrow bSCa$$

$$B \rightarrow aSB \mid bBC$$

$$C \rightarrow aBC \mid ad$$

Solution:

Here a, b, d are terminals

Since $C \rightarrow ad$, C is generating symbol and similarly $S \rightarrow aC$, S is also generating symbol.

And $A \rightarrow bSCa$, hence A is generating symbol.

But $B \rightarrow aSB \mid bBC$. Thus, B is the only non-generating symbol.

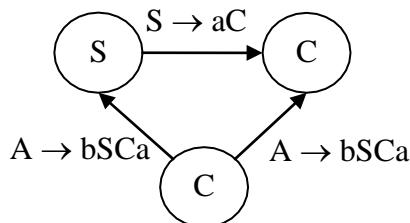


Eliminate B from the grammar and the production becomes,

$$S \rightarrow aC$$

$$A \rightarrow bSCa$$

$$C \rightarrow ad$$



The reachable symbols are S and C. A cannot be reached from S. Eliminating A, the production becomes $P = \{S \rightarrow aC, C \rightarrow ad\}$

7. Convert the following grammar into an equivalent with no unit productions and no useless symbols. (AU – Nov/Dec 2011)

$$S \rightarrow a \mid Ab \mid aBa$$

$$A \rightarrow b \mid \epsilon$$

$$B \rightarrow b \mid A$$

Solution:

(i) **Elimination of ϵ production**

$A \rightarrow \epsilon$ is a null production and since $B \rightarrow A \rightarrow \epsilon$ and $S \rightarrow A \rightarrow \epsilon$, all the symbols $\{S, A, B\}$ are nullable variables.

Therefore, after removing $A \rightarrow \epsilon$, the production becomes,

$$S \rightarrow a \mid Ab \mid aBa \mid b \mid aa$$

$$A \rightarrow b$$

$$B \rightarrow b$$

NORMAL FORMS FOR CFG

- Let the grammar $G = (V, T, P, S)$ be a context free grammar. If the production rules in G satisfy some restrictions, then they are said to be in normal form.
- There are two types of normal forms
 - Chomsky normal form (CNF)
 - Greibach normal form (GNF)

GREIBACH NORMAL FORM (GNF)**(June 2009, June 2013)**

- Greibach normal form is named after Sheila Greibach who constructed GNF grammar initially.
- A grammar $G = (V, T, P, S)$ is said to be in GNF if every production rule is of the form, $X \rightarrow a\alpha$
where,

$\alpha \in T$, an element of the set of terminals

$\alpha \in V^*$ a string of non terminals/epsilon

$X \in V$, an element of the set of variables

- The right hand side of every production starts with a terminal, followed by a string of variables of zero/ more length.

Algorithm to convert a CFG to GNF

- 1) Eliminate useless symbols, unit productions and null productions from the grammar, G .
- 2) The production rule of the form, $A \rightarrow X_1 X_2 \dots X_n$, other than X_1 all other symbol should be a variable, and X_1 could be a terminal.
- 3) Rename the variables in all the productions as $A_1, A_2, A_3, \dots, A_n$ productions.
- 4) Modify the productions with the renamed variables.
- 5) Remove the left recursion from every production of the form $A_k \rightarrow A_k\alpha$, by adding B productions using left recursion algorithm as given below.
- 6) Modify A_i productions to the form $A_i \rightarrow a\alpha$, where a is a terminal and α is a string of variables.
- 7) Modify B_i productions to the form $B_i \rightarrow a\alpha$, where a is a terminal and α is string of variables.

Left recursive algorithm

- A production of the form $A \rightarrow A\alpha$ is said to be left recursive, with the variable on the left hand side appears as the first symbol on the right hand side.

Consider a CFG containing productions of the forms.

$A \rightarrow A\alpha \Rightarrow$ Left recursive grammars

$B \rightarrow \beta \Rightarrow$ For terminating the recursive $A \rightarrow A\alpha$

- The language generated by above production is

$A \rightarrow \underline{A}\alpha \quad [Since A \rightarrow A\alpha]$

$\rightarrow \underline{A} \alpha \alpha \quad [Since A \rightarrow A\alpha]$



$\rightarrow A \alpha \alpha \alpha$ [Since $A \rightarrow A\alpha$]

.

.

.

$\rightarrow A\alpha^n$ [n^{th} time substitution]

$\rightarrow \beta\alpha$ [Since $A \rightarrow \beta$]

- The right recursive grammar for $\beta\alpha^n$ can be written as

$A \rightarrow \beta B | \beta$ [B generates a string a^n , and $A \rightarrow \beta$ is to terminal the recursion]

$B \rightarrow \alpha B | \alpha$

Thus the left recursive grammar is

$A \rightarrow A \alpha | \beta$

which can be written using right recursive grammar as

$A \rightarrow \beta B | \beta$

$B \rightarrow \alpha B | \alpha$

PROBLEMS ON GNF

1. Construct a grammar in GNF which is equivalent to the grammar.

$S \rightarrow AA | a$

$A \rightarrow SS | b$ (AU – Dec 2003, Dec 2006, Dec 2007, Dec 2008 Nov / Dec 2010)

Solution:

Step 1: Simplification of grammar by removing useless symbols, unit and null productions. The given grammar is already in simple form.

Step 2: Renaming of variables as A_1, A_2, \dots, A_n .

Let S be renamed as A_1 and A be renamed as A_2 respectively. (The terminals need not be renamed.)

Therefore the given grammar becomes.

$A_1 \rightarrow A_2 A_2 | a$

$A_2 \rightarrow A_1 A_1 | b$

Step 3: Modification of the productions.

The higher numbered production is modified first.

Hence,

$A_2 \rightarrow \underline{A}_1 A_1 | b$ [Substitution of production to the leftmost variable]

$\rightarrow [A_2 A_2 | a] A_1 | b$

$\underline{A}_2 \rightarrow \underline{A}_2 A_2 A_1 | a A_1 | b$

Stop substitution and modification when the left side symbol and the first symbol on the right side are same.

Step 4: Removing left recursion

The modified production, $\underline{A}_2 \rightarrow A_2 A_2 A_1 | a A_1 | b$

↑

Left recursion grammat

The left recursive grammar can be removed by introducing a new variable B_2 [B_2 is named, since it is substituted for A_2 .]

Break A_2 into two forms

$A_2 \rightarrow A_2 A_2 A_1$ and $A_2 \rightarrow a A_1 | b \rightarrow (1)$

$A_2 \rightarrow a A_1 B_2 | b B_2 \rightarrow (2)$

Remove the first symbol at right side

and (2) \Rightarrow

$A_2 \rightarrow a A_1 B_2 | b B_2 | a A_1 | b$

$B_2 \rightarrow A_2 A_1 B_1 | A_2 A_1$

The resulting grammar

$A_1 \rightarrow A_2 A_2 | a$

$A_2 \rightarrow \underline{a} A_1 B_2 | \underline{b} B_2 | \underline{a} A_1 | \underline{b} \Rightarrow GNF$

$B_2 \rightarrow A_2 A_1 B_1 | A_2 A_1$

Step 5: Production in GNF

Substitute A_2 in B_2

$B_2 \rightarrow \underline{A}_2 A_1 B_1 | \underline{A}_2 A_1$

$\rightarrow [a A_1 B_2 | b B_2 | a A_1 | b] A_1 B_2 |$

$[a A_1 B_2 | b B_2 | a A_1 | b] A_1$

$$\begin{aligned} & \rightarrow \underline{a} A_1 B_2 A_1 B_2 | \underline{b} B_2 | A_1 B_2 | \underline{a} A_1 A_1 B_2 | \\ & \quad \underline{b} A_1 B_2 | \underline{a} A_1 B_2 A_1 | \underline{b} B_2 A_1 | \underline{a} A_1 A_1 | \underline{b} A_1 \Rightarrow \text{GNF} \end{aligned}$$

Substitute A_2 in A_1

$$\begin{aligned} A_1 & \rightarrow \underline{A}_2 A_2 | a \\ & \rightarrow [a A_1 B_2 | aB_2 | aA_1 | b] A_2 | a \end{aligned}$$

$$A_1 \rightarrow \underline{a} A_1 B_2 A_1 | \underline{a} B_2 A_1 | \underline{a} A_1 A_2 | \underline{b} A_2 | a \Rightarrow \text{GNF}$$

The final set of productions is

$$\begin{aligned} A_1 & \rightarrow a A_1 B_2 A_2 | a B_2 A_2 | aA_1 A_2 | b A_2 | a \\ A_2 & \rightarrow a A_1 B_2 | b B_2 | a A_1 | b \\ B_2 & \rightarrow a A_1 B_2 A_1 B_2 | b B_2 A_1 B_2 | a A_1 A_1 B_2 | b A_1 B_2 | \\ & \quad a A_1 B_2 A_1 | b B_2 A_1 | a A_1 A_1 | b A_2 \end{aligned}$$

$$G = (V, T, P, S)$$

$$V = \{A_1, A_2, B_2\}$$

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

$$S = \{A_1\}$$

P \Rightarrow as given above

- 2. Given the GNF for the following CFG.** (AU – May 2004, Dec 2004, Dec 2005, June 2006, June 2007, May 2012, Dec 2009, May 2007)

$$S \rightarrow AB$$

$$A \rightarrow BS | b$$

$$B \rightarrow SA | a$$

Solution:

Step 1: Simplification of grammar

The grammar is already simplified.

Step 2: Renaming the variables

Let S, A, B be renamed as A_1, A_2, A_3 respectively.

The productions with renamed variables are

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 | b$$

$$A_3 \rightarrow A_1 A_2 | a$$

Step 3: Modification of grammars

Consider the highest numbered production.

$$\begin{array}{l} \underline{A_3 \rightarrow A_1 A_2 | a} \\ \quad \uparrow \\ \rightarrow [A_2 A_3] A_2 | a \end{array} \quad [\text{substitute } A_1 \text{ in } A_3]$$

$$\begin{array}{l} \underline{A_3 \rightarrow A_2 A_3 A_2 | a} \\ \quad \uparrow \\ \rightarrow [A_3 A_1 | b] A_3 A_2 | a \end{array} \quad [\text{substitute } A_2 \text{ in } A_3 \text{ till we get } A_3 \text{ as the first symbol in R.H.S}]$$

$$\begin{array}{l} \underline{A_3 \rightarrow A_3 A_1 A_3 A_2 | b A_3 A_2 | a} \\ \quad \uparrow \\ \text{Left recursive grammar} \end{array}$$

Step 4: Removing left recursion

Break down A_3 production as

$$A_3 \rightarrow A_3 A_1 A_3 A_2 \rightarrow (1)$$

$$\text{and } A_3 \rightarrow b A_3 A_2 | a \rightarrow (2)$$

Introduce B_3 in (2)

$$A_3 \rightarrow b A_3 A_2 B_3 | a B_3 | b A_3 A_2 | a$$

With B_3 in (2) Without B_3 in (2)

Eqn (1) becomes

$$B_3 \rightarrow A_1 A_3 A_2 B_3 | A_1 A_3 A_2 \quad [\text{Eqn (1) becomes } B_3 \rightarrow A_1 A_3 A_2 B_3 | A_1 A_3 A_2]$$

Addition of B_3 Without B_3

The resulting productions are

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 | b$$

$$A_3 \rightarrow b A_3 A_2 B_3 | a B_3 | b A_3 A_2 | a \Rightarrow \text{GNF}$$

$$B_3 \rightarrow A_1 A_3 A_2 B_3 | A_1 A_3 A_2$$

Step 5: Find the productions in GNF

Substitute A_3 in A_2

$$A_2 \rightarrow A_3 A_1 | b$$

$$\rightarrow [b A_3 A_2 B_3 | a B_3 | b A_3 A_2 | a] A_1 | b$$

$$A_2 \rightarrow b A_3 A_2 B_3 | a B_3 A_1 | b A_3 A_2 A_1 | a A_1 | b$$

Substitute A_2 in A_1

$$A_1 \rightarrow A_2 A_3$$

$$\rightarrow [b A_3 A_2 B_3 A_1 | a B_3 A_1 | b A_3 A_2 A_1 | a A_1 | b] A_3$$

$$\rightarrow b A_3 A_2 B_3 A_1 A_3 | a B_3 A_1 A_3 | b A_3 A_2 A_1 A_3 | a A_1 A_3 | b A_3$$

Substitute A_1 in B_3

$$B_3 \rightarrow A_1 A_3 A_2 B_3 | A_1 A_3 A_2$$

$$\rightarrow [b A_3 A_2 B_3 A_1 A_3 | a B_3 A_1 A_3 | b A_3 A_2 A_1 A_3 | a A_1 A_3 | b A_3] A_3 A_2 B_3 |$$

$$[b A_3 A_2 B_3 A_1 A_3 | a B_3 A_1 A_3 | b A_3 A_2 A_1 A_3 | a A_1 A_3 | b A_3] A_3 A_2$$

$$\rightarrow b A_3 A_2 B_3 A_1 A_3 A_3 A_2 B_3 | a B_3 A_1 A_3 A_3 A_2 B_3 | b A_3 A_2 A_1 A_3 A_3 A_2 B_3 | \\ a A_1 A_3 A_3 A_2 B_3 | b A_3 A_3 A_2 B_3 | b A_3 A_2 B_3 A_1 A_3 A_3 A_2 | a B_3 A_1 A_3 A_3 A_2 | \\ b A_3 A_2 A_1 A_3 A_3 A_2 | a A_1 A_3 A_3 A_2 | b A_3 A_3 A_2$$

The CFG in GNF is given by

$$G = (V, T, P, S) \text{ where}$$

$$V = \{A_1, A_2, A_3, B_3\}$$

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

$$S = \{A_1\}$$

$$P \Rightarrow \quad A_1 \rightarrow b A_3 A_2 B_3 A_1 A_3 | a B_3 A_1 A_3 | b A_3 A_2 A_1 A_3 | a A_1 A_3 | b A_3$$

$$A_2 \rightarrow b A_3 A_2 B_3 A_1 | a B_3 A_1 | b A_3 A_2 A_1 | a A_1 | b$$

$$A_3 \rightarrow b A_3 A_2 B_3 | a B_3 | b A_3 A_2 | a$$

$$B_3 \rightarrow b A_3 A_2 B_3 A_1 A_3 A_3 A_2 B_3 | a B_3 A_1 A_3 A_3 A_2 B_3 | b A_3 A_2 A_1 A_3 A_3 A_2 B_3 |$$

$$a A_1 A_3 A_3 A_2 B_3 | b A_3 A_3 A_2 B_3 | b A_3 A_2 B_3 A_1 A_3 A_3 A_2 | a B_3 A_1 A_3$$

$$A_3 A_2$$

$$| b A_3 A_2 A_1 A_3 A_3 A_2 | a A_1 A_3 A_3 A_2 | b A_3 A_3 A_2$$

3. Convert the following CFG to GNF

$$S \rightarrow AB$$

$$A \rightarrow BSB \mid BB \mid b$$

$$B \rightarrow aAb \mid a$$

Solution:

Step 1: Simplification of grammar

The grammar doesn't contain useless symbols, null and unit productions.

Step 2: Renaming the variables

Making every symbol except the first symbol as a variable

$$S \rightarrow AB$$

$$A \rightarrow BSB \mid BB \mid b$$

$$B \rightarrow a A \underline{C_b} \mid a, \text{ where } C_b \rightarrow b$$

Renaming the variables S, A, B, C_b as A₁, A₂, A₃, A₄ respectively.

The production becomes

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 A_3 \mid A_3 A_3 \mid b$$

$$A_3 \rightarrow a A_1 A_4 \mid a$$

$$A_4 \rightarrow b$$

Step 3: Modification of grammar

Consider A₄ → b ⇒ already in GNF

Consider A₃ → a A₁ A₄ | a ⇒ already in GNF

Consider A₂ → A₃ A₁ A₃ | A₃ A₃ | b

Substitute A₃ in A₂

$$A_2 \rightarrow [a A_1 A_4 \mid a] A_1 A_3 \mid [a A_1 A_4 \mid a] A_3 \mid b$$

$$\rightarrow a A_1 A_4 A_1 A_3 \mid a A_1 A_3 \mid a A_1 A_4 A_3 \mid a A_3 \mid b$$

Substitute A₂ in A₁₅

$$A_1 \rightarrow A_2 A_3$$

$$\rightarrow [a A_1 A_4 A_1 A_3 | a A_1 A_3 | a A_1 A_4 A_3 | a A_3 | b] A_3$$

$$\rightarrow a A_1 A_4 A_1 A_3 A_3 | a A_1 A_3 A_3 | a A_1 A_4 A_3 A_3 | a A_3 A_3 | b A_3$$

Step 4: Final grammar

$$G = (V, T, P, S) \text{ where}$$

$$V = \{A_1, A_2, A_3, A_4\}$$

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

$$S = \{A_1\}$$

$$P \Rightarrow \quad A_1 \rightarrow a A_1 A_4 A_1 A_3 A_3 | a A_1 A_3 A_3 | a A_1 A_4 A_3 A_3 | a A_3 A_3 | b A_3$$

$$A_2 \rightarrow a A_1 A_4 A_1 A_3 | a A_1 A_3 | a A_1 A_4 A_3 | a A_3 | b$$

$$A_3 \rightarrow a A_1 A_4 | a$$

$$A_4 \rightarrow b$$

4. Convert the following grammar to GNF

$$S \rightarrow BS$$

$$S \rightarrow Aa$$

$$A \rightarrow bc$$

$$B \rightarrow Ac$$

Solution:

Step 1: Simplification of the grammar

The grammar is already in simplified form.

Step 2: Renaming the variables

Making every first symbol in RHS to be a variable

$$S \rightarrow BS | AC_a,$$

$$C_a \rightarrow a$$

$$A \rightarrow b C_b,$$

$$C_b \rightarrow c$$

$$B \rightarrow A C_b$$

Renaming S, A, B, C_a, C_b as A₁, A₂, A₃, A₄, A₅ respectively.

$$\therefore A_1 \rightarrow A_3 A_1 | A_2 A_4$$

$$A_2 \rightarrow b A_5$$

$$A_3 \rightarrow A_2 A_5$$

$$A_4 \rightarrow a$$

$$A_5 \rightarrow c$$

Step 3: Converting the productions to GNF

Here A₅ → c, A₄ → a, A₂ → bA₅ are already in GNF

Consider A₃ → A₂ A₅

Substitute A₂ in A₃

$$A_3 \rightarrow b[A_5] A_5$$

$$\rightarrow bA_5 A_5$$

Substitute A₃ and A₂ in A₁

$$A_1 \rightarrow [bA_5 A_5] A_1 | [bA_5] A_4$$

$$\rightarrow bA_5 A_5 A_1 | bA_5 A_4$$

Step 4: Final production in the grammar G

G = (V, T, P, S), where

V = {A₁, A₂, A₃, A₄, A₅}

T = {a, b}

S = {A₁}

$$P \Rightarrow A_1 \rightarrow bA_5 A_5 A_1 | bA_5 A_4$$

$$A_2 \rightarrow b A_5$$

$$A_3 \rightarrow bA_5 A_5$$

$$A_4 \rightarrow a$$

$$A_5 \rightarrow c$$

5. Find the GNF of the grammar

$$S \rightarrow ABAb \mid ab$$

$$B \rightarrow ABA \mid a$$

$$A \rightarrow a \mid b$$

Solution:

1. The grammar is already in simplified form
2. Conversion of CFG to GNF

$$S \rightarrow ABA C_b \mid a C_b, \text{ where } C_b \rightarrow b$$

$$B \rightarrow ABA \mid a$$

$$A \rightarrow a \mid b$$

Substitute A in B and S after renaming the variables.

$$A_1 \rightarrow A_2 A_3 A_2 A_4 \mid a A_4$$

$$A_2 \rightarrow a \mid b$$

$$A_3 \rightarrow A_2 A_3 A_2 \mid a$$

$$A_4 \rightarrow b$$

Since A_2 and A_4 are already in GNF,

Substitute A_2 in A_1 and A_3

$$A_1 \rightarrow [a \mid b] A_3 A_2 A_4 \mid a A_4$$

$$\rightarrow a A_3 A_2 A_4 \mid b A_3 A_2 A_4 \mid a A_4$$

$$A_3 \rightarrow [a \mid b] A_3 A_2 \mid a$$

$$\rightarrow a A_3 A_2 \mid b A_3 A_2 \mid a$$

Thus the grammar $G = (V, T, P, S)$, becomes,

$$V = \{A_1, A_2, A_3, A_4\}$$

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

$$S = \{A_1\}$$

$$P \Rightarrow \quad A_1 \rightarrow a A_3 A_2 A_4 \mid b A_3 A_2 A_4 \mid a A_4$$

$$A_2 \rightarrow a \mid b$$

$$A_3 \rightarrow a A_3 A_2 \mid b A_3 A_2 \mid a$$

$$A_4 \rightarrow b$$

6. Find the GNF equivalent to the following CFG

$$S \rightarrow AB$$

$$A \rightarrow aA \mid bB \mid b$$

$$B \rightarrow b$$

Solution:

The grammar is already in simplified form and A and B are already in GNF. Hence substitute A in S.

$$\begin{aligned} S &\rightarrow AB \\ &\rightarrow [aA \mid bB \mid b] B \\ &\rightarrow aAB \mid bBB \mid bB \end{aligned}$$

The grammar $G = (V, T, P, S)$ is given by

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

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

$$\begin{aligned} P \Rightarrow \quad S &\rightarrow aAB \mid bBB \mid bB \\ A &\rightarrow aA \mid bB \mid b \\ B &\rightarrow b \\ S &= \{S\} \end{aligned}$$

The grammar need not be renamed if the substitution is simple and is done for one production alone.

7. Convert $S \rightarrow ab \mid aS \mid aaS$ to GNF

Solution:

The grammar is already in simplified form.

Since there is one production to be converted, there is no need of renaming.

$$S \rightarrow ab \mid aS \mid aaS$$

is converted as

$$\begin{aligned} S &\rightarrow a C_b \mid aS \mid a C_a S \text{ where} \\ C_a &\rightarrow a \\ C_b &\rightarrow b \end{aligned}$$

The final grammar $G = (V, T, P, S)$ is given as



$$\begin{aligned}
 V &= \{S, C_a, C_b\} \\
 T &= \{a, b\} \\
 P \Rightarrow S &\rightarrow aC_b \mid aS \mid aC_aS, \\
 C_a &\rightarrow a \\
 C_b &\rightarrow b \\
 S &= \{S\}
 \end{aligned}$$

CHOMSKY NORMAL FORM (CNF)

(May/June 2012)

- A context free grammar, $G = (V, T, P, S)$ is said to be in CNF if each production in G is of the form, $X \rightarrow YZ; X \rightarrow \alpha$, where $X, Y, Z \in V$ and $\alpha \in T$.
- Every CFG without null productions can be easily converted into its equivalent CNF.

Algorithm to convert a CFG to CNF

- Eliminate the useless symbols, unit and null productions from the grammar
- Each variable should be having a derivation of length 2 or more, having only variable as of the form, $A \rightarrow \alpha$ where $|\alpha| \geq 2, \alpha \in V$.
- The production have three more than three variables as derivation, must be broken down into a cascade of productions with derivations containing at most 2 variables.

PROBLEMS ON CNF

- 1. Find the CNF for the CFG, $S \rightarrow aAbB, A \rightarrow aA, B \rightarrow bB \mid b$.**

(AU – Dec 2003, June 2007)

Solution:

Step 1: Simplification of the grammar

The given grammar is already in simple form.

Step 2: Every symbol in α in a production of the form $A \rightarrow \alpha$ where $|\alpha| \geq 2$ should be a variable.

This can be done by converting the terminal to variable as,

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

The productions after substituting C_a and C_b in the given grammar is given as,

$$S \rightarrow C_a A C_b B$$

$$A \rightarrow C_a A$$

$B \rightarrow C_b B \mid b$ [$B \rightarrow b$ cannot be changed as $B \rightarrow C_b$ since there is only one terminal symbol]

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

Step 3: Conversion to CNF

$$S \rightarrow C_a A \underbrace{C_b B}_{C_1} \Rightarrow S \rightarrow C_a C_1$$

$$C_1 \rightarrow A \underbrace{C_b B}_{C_2} \Rightarrow C_1 \rightarrow A C_2$$

$$C_2 \rightarrow C_b B$$

$$A \rightarrow C_a A$$

$$B \rightarrow C_b B \mid b$$

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

where,

$$V = \{S, A, B, C_a, C_b, C_1, C_2\}$$

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

$$S = \{S\}$$

2. Convert the grammar given to CNF. (AU – Dec 2005, Dec 2006)

$$S \rightarrow bA \mid aB$$

$$A \rightarrow bAA \mid aS \mid a$$

$$B \rightarrow aBB \mid bS \mid b$$

Solution:

Step 1: Simplification of grammar

The grammar is already in simplified form.

Step 2: Every symbol α in the production of the form $A \rightarrow \alpha$ where $|\alpha| \geq 2$ should be a variable.

This is done by adding the following two productions.

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

The set of productions after substituting $C_a \rightarrow a$ and $C_b \rightarrow b$ is,

$$S \rightarrow C_b A \mid C_a B$$

$A \rightarrow C_b AA \mid C_a S \mid a$ (No need to apply variables, since they are single, with no other variable / terminals.)

$$B \rightarrow C_a BB \mid C_b S \mid b$$

Step 3: Conversion to CNF

$$S \rightarrow C_b A$$

$$S \rightarrow C_a B$$

$$A \underset{\substack{\swarrow \\ \searrow}}{\rightarrow} C_b AA \Rightarrow A \rightarrow C_b C_1$$

$$C_1 \quad C_1 \rightarrow AA$$

$$AA \rightarrow C_a S$$

$$A \rightarrow a$$

$$B \rightarrow C_a BB \Rightarrow B \rightarrow C_a C_2$$

$$\underset{\substack{\swarrow \\ \searrow}}{C_2} \quad C_2 \rightarrow BB$$

$$B \rightarrow b$$

where,

$$V = \{S, A, B, C_a, C_b, C_1, C_2\}$$

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

$$S = \{S\}$$

3. Find the CNF equivalent to

(AU – May 2004, Dec 2010)

$$S \rightarrow aAD$$

$$A \rightarrow aB \mid bAB$$

$$B \rightarrow b$$

$$D \rightarrow d$$

Solution:

Step 1: Simplification of grammar

The grammar is already in simplified form with no useless symbols, unit and null productions.

Step 2: Conversion of terminals to variables.

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

∴ The production becomes

$$S \rightarrow C_a AD$$

$$A \rightarrow C_a B \mid C_b AB$$

$$B \rightarrow b$$

$$D \rightarrow d$$

Step 3: Conversion to CNF

$$S \rightarrow C_a \underbrace{AD}_{C_1} \Rightarrow S \rightarrow C_a C_1$$

$$C_1 \quad C_1 \rightarrow AD$$

$$A \rightarrow C_a B$$

$$A \rightarrow C_b \underbrace{AB}_{C_2} \Rightarrow A \rightarrow C_b C_2$$

$$C_2 \quad C_2 \rightarrow AB$$

$$B \rightarrow b$$

$$D \rightarrow d$$

4. Convert the grammar given below to CNF

$$S \rightarrow ABA$$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow bB \mid \epsilon$$

Solution:

Step1: Simplification of grammar

1) Elimination of useless production

There are no non-reachable and non-generating symbols in the given grammar.

2) Elimination of Epsilon production

Since $A \rightarrow \epsilon$, $B \rightarrow \epsilon$, and $S \rightarrow ABA$, the nullable variables are $\{A, B, S\}$

Eliminating the ϵ productions, we get,

$$S \rightarrow ABA | AB | BA | A | B$$

$$A \rightarrow aA | a$$

$$B \rightarrow bB | b$$

3) Eliminating the unit production

There are two unit productions $S \rightarrow A$ and $S \rightarrow B$.

Eliminating the unit production, we get

$$S \rightarrow ABA | AB | BA | aA | a | bB | b | AA$$

$$A \rightarrow aA | a$$

$$B \rightarrow bB | b$$

Step 2: Addition of two productions to define the terminals.

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

Step 3: Conversion to CNF

$$\begin{array}{l} S \rightarrow ABA \\ \quad \quad \quad \overbrace{\quad \quad \quad}^{} \\ \quad \quad \quad C_1 \end{array}$$

$$S \rightarrow A C_1$$

$$C_1 \rightarrow BA$$

$$S \rightarrow AB$$

$$S \rightarrow BA$$

$$S \rightarrow C_a A; \text{ where } C_a \rightarrow a$$

$$S \rightarrow a$$

$$S \rightarrow C_b B; \text{ where } C_b \rightarrow b$$

$$S \rightarrow b$$

$$S \rightarrow AA$$

$$A \rightarrow C_a A$$

$$A \rightarrow a$$

$$B \rightarrow C_b B$$

$$B \rightarrow b$$

5. Convert the following grammar to CNF

$$S \rightarrow Pba$$

$$S \rightarrow aab$$

$$Q \rightarrow Pc$$

Solution:

Step 1: Simplification of the grammar

There is no unit or epsilon production there is a non-reachable symbol, B.

The symbol can be deleted, and resulting production is

$$S \rightarrow Pba \mid aab$$

Step 2: Adding productions to handle terminals

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

∴

$$S \rightarrow P C_b C_a \mid C_a C_a C_b$$

Step 3: Conversion to CNF

$$S \rightarrow P \underbrace{C_b C_a}_{C_1}$$

$$S \rightarrow C_a \underbrace{C_a C_b}_{C_2}$$

$$S \rightarrow P C_1$$

$$S \rightarrow C_a C_2$$

$$C_1 \rightarrow C_b C_a$$

$$C_2 \rightarrow C_a C_b$$

6. Convert the following CFG to CNF

$$S \rightarrow AACD$$

$$A \rightarrow 0A1 \mid \epsilon$$

$$C \rightarrow 0C0 \mid 1D1$$

$$A \rightarrow 0D0 \mid 1D1 \mid \epsilon$$

Solution:

Step 1: Simplification of grammar

(1) **Removing epsilon production**

Here $A \rightarrow \epsilon$ is a nullable production.

Elimination of A:

$$S \rightarrow AACD \mid ACD \mid CD$$

$$A \rightarrow 0A1 \mid 01$$

$$C \rightarrow 0C \mid 0$$

$$D \rightarrow 0D0 \mid 1D1$$

(2) Removing non-generating symbol

Here the symbol S and D are non-generating.

Since the start symbol is a non-generating symbol, the grammar becomes invalid.

7. Convert the grammar given to CNF

(AU – Dec 2004)

$$S \rightarrow 0A0 \mid 1B1 \mid BB$$

$$A \rightarrow C$$

$$B \rightarrow S \mid A$$

$$C \rightarrow S \mid \epsilon$$

Solution:

Elimination of ϵ -production

Here $C \rightarrow \epsilon$ is the epsilon production.

Since $A \rightarrow C$, $B \rightarrow A \rightarrow C$, $S \rightarrow A \rightarrow C$, then A, B, S, C are all nullable symbols.

Eliminating the epsilon productions, we get

$$S \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \mid B$$

$$A \rightarrow C$$

$$B \rightarrow S \mid A$$

$$C \rightarrow S$$

Elimination of unit production

The unit productions are $S \rightarrow B$, $A \rightarrow C$, $B \rightarrow S$, $C \rightarrow S$, $B \rightarrow A$.

After eliminating the unit productions, we get

$$S \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$$

$$A \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \quad [\text{since } A \rightarrow C \rightarrow S]$$

$$B \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \quad [\text{since } B \rightarrow S]$$

$$C \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \quad [\text{since } C \rightarrow S]$$

Eliminating of useless symbols

The production „C“ is non-reachable since $S \rightarrow A$ and $S \rightarrow B$.

After eliminating C we get,

$$S \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$$

$$A \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$$

$$B \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB$$

Since S, A and B productions are identical we shall eliminate A and B.

$$S \rightarrow 0A0 \mid 00 \mid 1B1 \mid 11 \mid BB \text{ which can be written as } S \rightarrow 0S0 \mid 00 \mid 1S1 \mid 11 \mid SS$$

Conversion to CNF

Add two productions to the terminals {0, 1}

$$C_1 \rightarrow 0$$

$$C_2 \rightarrow 1$$

Substituting C_1 and C_2 in S, we get

$$S \rightarrow C_1 S C_1 \mid C_1 C_1 \mid C_2 S C_2 \mid C_2 C_2 \mid SS$$

The above grammar is now converted to CNF as

$$S \rightarrow C_1 \underbrace{S C_1}_{C_3} \Rightarrow S \rightarrow C_1 C_3$$

$$C_3 \quad C_3 \rightarrow S C_1$$

$$S \rightarrow C_1 C_1$$

$$S \rightarrow C_2 \underbrace{S C_2}_{C_4} \Rightarrow S \rightarrow C_2 C_4$$

$$C_4 \quad C_4 \rightarrow S C_2$$

$$S \rightarrow C_2 C_2$$

$$S \rightarrow SS$$

$$C_1 \rightarrow 0$$

$$C_1 \rightarrow 1$$

8. Convert the grammar $S \rightarrow AB \mid aB, A \rightarrow aab \mid \epsilon, B \rightarrow bbA$ into CNF.

(AU – June 2009)

Solution:

Elimination of ϵ -production

From the given grammar, $A \rightarrow \epsilon$, which is the nullable variable.

After eliminating the epsilon production, we get

$$S \rightarrow AB \mid B \mid aB$$

$$A \rightarrow aab$$

$$B \rightarrow bbA \mid bb$$

Elimination of unit production

The unit production is $S \rightarrow B$.

After eliminating the unit production the resulting grammar becomes,

$$S \rightarrow AB \mid bbA \mid bb \mid aB$$

$$A \rightarrow aab$$

$$B \rightarrow bbA \mid bb$$

There is no useless symbol in the grammar.

Conversion to CNF

Adding productions to terminals,

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

Rewriting the grammar by substituting C_a and C_b , we get

$$S \rightarrow AB \mid C_b C_b A \mid C_b C_b \mid C_a B$$

$$A \rightarrow C_a C_a C_b$$

$$B \rightarrow C_b C_b A \mid C_b C_b$$

Conversion the grammar to CNF gives,

$$S \rightarrow AB$$

$$S \rightarrow C_b C_b A \Rightarrow S \rightarrow C_b C_1$$

$$C_1 \quad C_1 \rightarrow C_b A$$

$$S \rightarrow C_b C_b$$

$$S \rightarrow C_a B$$

$$A \rightarrow C_a C_a C_b \Rightarrow A \rightarrow C_a C_2$$

$$C_2 \quad C_2 \rightarrow C_a C_b$$

$$B \rightarrow C_b C_b A \Rightarrow B \rightarrow C_b C_1$$

$$C_1 \quad C_1 \rightarrow C_b A$$

$$B \rightarrow C_b C_b$$

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

- 9. Simplify the following grammar to its equivalent CNF. $P \Rightarrow \{ S \rightarrow AB \mid CA, A \rightarrow a, B \rightarrow BC \mid AB, C \rightarrow aB \mid b \}$** (AU – Dec 2008, Dec 2007)

Solution:

Step 1: Simplification of the grammar

There are no \in - production in the grammar.

There are no unit productions.

The non generating symbols in G is {B} since $B \rightarrow BC \mid AB$ generates a derivations, that will never end.

Hence after removing B, the grammar becomes,

$$S \rightarrow CA$$

$$A \rightarrow a$$

$$C \rightarrow b$$

Thus the grammar is already in CNF.

- 10. Convert the grammar into CNF.**

(AU – Apr / May 2008)

$$A \rightarrow bAB \mid \epsilon$$

$$B \rightarrow BAa \mid \epsilon$$

Solution :

- 1) Elimination of \in -productions

Here $A \rightarrow \in$, $B \rightarrow \in$ are the \in productions. Hence the nullable variables are {A, B}

After removing $A \rightarrow \in$, $B \rightarrow \in$, the production becomes,

$$A \rightarrow bAB \mid bB \mid bA \mid b$$

$$B \rightarrow BAa \mid Ba \mid Aa \mid a$$

- 2) Conversion to CNF

$A \rightarrow b$ are already in CNF

$$B \rightarrow a$$

$$A \rightarrow bAB \Rightarrow A \rightarrow C_b \underbrace{C_1}_{C_1} \text{ where } C_1 \rightarrow AB, C_b \rightarrow b$$

$$A \rightarrow C_b B, \text{ where } C_b \rightarrow b$$

$$A \rightarrow C_b A, \text{ where } C_b \rightarrow b$$

$$B \rightarrow BBa \Rightarrow B \rightarrow B \underbrace{C_2}_{C_2} \text{ where } C_2 \rightarrow B C_a, C_a \rightarrow a$$

$$B \rightarrow B C_a, \text{ where } C_a \rightarrow a$$

$$B \rightarrow A C_a, \text{ where } C_a \rightarrow a$$

11. Construct the following grammar in CNF. (AU – May / June 2013)

$$A \rightarrow BCD \mid b$$

$$B \rightarrow YC \mid d$$

$$C \rightarrow gA \mid c$$

$$D \rightarrow dB \mid a, Y \rightarrow f$$

Solution:

There is no useless symbols, unit and null productions in the grammar.

First, the production rules for terminal symbols is constructed as

$$C_1 \rightarrow a \quad C_4 \rightarrow d$$

$$C_2 \rightarrow b \quad C_5 \rightarrow f$$

$$C_3 \rightarrow c \quad C_6 \rightarrow g$$

After substitution of the above productions in the given grammar, we get

$$A \rightarrow BCD \mid b$$

$$B \rightarrow YC_3 \mid d$$

$$C \rightarrow C_6A \mid c$$

$$D \rightarrow C_4 B \mid a$$

$$Y \rightarrow f$$

The grammar above is converted to CNF as,

$$A \rightarrow B C D \Rightarrow A \rightarrow B C_a$$

$$C_a \quad C_a \rightarrow CD$$

$A \rightarrow b$	$C_1 \rightarrow a$
$B \rightarrow YC_3$	$C_2 \rightarrow b$
$B \rightarrow d$	$C_3 \rightarrow c$
$C \rightarrow C_6A$	$C_4 \rightarrow d$
$C \rightarrow c$	$C_5 \rightarrow f$
$D \rightarrow C_4 B$	$C_6 \rightarrow g$
$D \rightarrow a$	
$Y \rightarrow f$	

PUMPING LEMMA FOR CFG

Let G be a context free grammar. Then there exists a constant „n“ such that any string, $\omega \in L(G)$ with $|\omega| \geq n$ can be rewritten as $\omega = uvxyz$, subject to the following conditions.

- (i) $|vxy| \leq n$
- (ii) $vy \neq \epsilon$
- (iii) for all $i \geq 0$, $uv^i xy^i z$ is in L

PUMPING LEMMA FOR CFL – THEOREM

(AU – May/June 2013, N Nov/Dec 2012, Nov/Dec 2011)

Theorem

Let L be a CFL. Then there exist a constant „n“ such that if „ ω “ is any string in L such that $|\omega| \geq n$, then ω can be written as $\omega = uvxyz$ subject to,

- i) $|vxy| \leq n$
- ii) $vy \neq \epsilon$ or $|vy| \neq 0$
- iii) for all $i \geq 0$, $uv^i xy^i z$ is in L

Proof

The initial step is to find a Chomsky normal grammar, G equivalent to L .

Let grammar $G = (V, T, P, S)$ such that $L(G) = L - \epsilon$.

Consider that the grammar has m variable with $n = 2^M$

Assume that „ ω “ is in L of length, $|\omega| \geq n$

By theorem

- o Any parse tree of length = M for the longest path must have a yield of length = 2^{M-1} is $n/2$ or less.

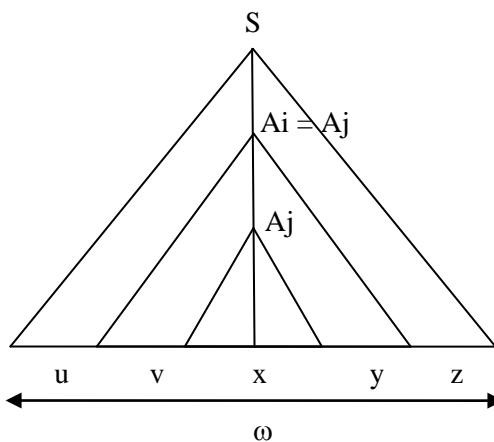
- Such a parse tree cannot have yield = z, since z is too large.

Thus a parse tree with yield ω has a path of length $\geq M + 1$.

This parse tree will have the vertices $\geq M + 2$, since there must be two vertices on an edge.

The tree is divided with the root = A_j of the subtree for the string ω as,

be \in .



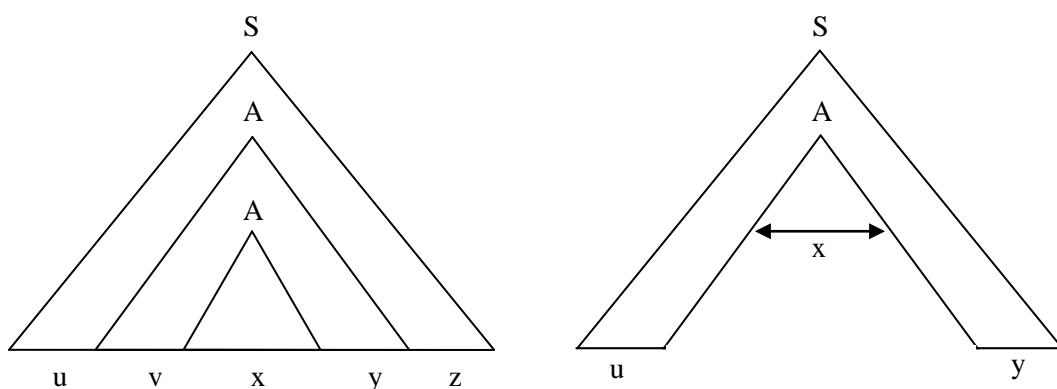
For the yield „x“ at „ A_j “ the strings „v“ and „y“ are to the left and right of „x“.

Since there are no unit productions in the chomsky's grammar, both „v“ and „y“ cannot

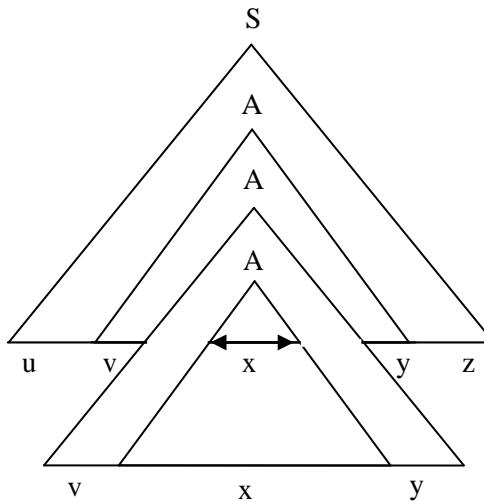
At A_i , strings „u“ and „z“ are to its left and right of vxy.

When $A_i = A_j$, the A_i and A_j are replaced by A has yield = x.

The resultant tree is given as,



The tree has a yield uxy and corresponds to the case, $i=0$ in $uv^i xy^i z$ [as given above]



When $i=2$, $uv^i xy^i z$ becomes

$uv^2 xy^2 \Rightarrow$ as given above, by expanding the tree with $x = vxy$ as another subtree.

The tree can be expanded in a similar way for any number of times as „ i “ grows, yielding the parse tree of the form $uv^i xy^i z$.

Here $|vxy| \leq n$ with A_i as the base of the tree with $k - i \leq M$.

Thus the longest path in of the subtree with root = A_i is no greater than $M+1$.

\therefore yield length is no greater than $2^M = n$.

Thus Pumping Lemma is proved.

Applications

(i) Used to check if a language is context-free or not.

PROBLEMS

1. Prove that $L = \{a^i b^i c^i \mid i \geq 1\}$ is not CFL.

(AU – May 2004, Dec 2004, May 2005, June 2006, June 2007, Dec 2007, Dec 2008, Dec 2009, N/D 2012, M/J 2012)

Solution:

(1) Let us assume that L is a CFL

(2) Let us pick up, $\omega = a^n b^n c^n$, where n is a constant.

(3) ω is rewritten as $uvxyz$, where

- (i) $|vxy| \leq n$
- (ii) $vy \neq \epsilon$
- (iii) for all i , $uv^i xy^i z \in L$.

If vy contains all three symbols a, b, c.

In this case either $v = ab / bc / ac$ or $y = ab / bc / ac$. The exact ordering of a, b, c will be broken.

If $i = 2$, $uv^i xy^i z \Rightarrow uv^2 xy^2 z$ becomes

Case (i) If $v = ab$ and $y = c$

$$uv^2 xy^2 z = (ab)^2 c \Rightarrow uv^i xy^i z \notin L$$

Case (ii) If $v = a$ and $y = bc$

$$uv^2 xy^2 z = a(bc)^2 \Rightarrow uv^i xy^i z \notin L$$

Hence L is not a CFL.

2. Prove that $L = \{a^i b^i c^j \mid j > i\}$ is not a CFL

Solution:

Let us assume that L is CFL

Let $\omega = a^i b^i c^j$, where i, j is a constant.

ω can be rewritten as $uvxyz$, where

- (i) $|vxy| \leq n$
- (ii) $vy \neq \epsilon$
- (iii) $uv^i xy^i z$ is in $L(G)$ for $i = 0, 1, 2$.

Let vy contains all three symbols a, b, c.

(i) If $v = ab$ and $y = c$

$$\text{Then } uv^2 xy^2 z = (ab)^2 c^2$$

Power of c should be greater than or equal to powers of ab. But its is not so here.

(ii) If $v = a$ and $y = bc$

$$uv^2 xy^2 z = a^2 (bc)^2. \text{ Here } j \geq i \text{ is not true.}$$

j is power of c

4.4 Theory of Computation

i is power of a,b

Thus there are unequal no of a"s and b"s and the count of either a or b can be increased from the count of c.

Hence L is not a CFL.

3. $A = \{a^{n^2} | n \geq 1\}$ is context free. If so, enumerate some member of the equivalent CFL.
(AU – May 2008).

Solution:

Let L a CFL. Let $L = a^{n^2}$ where n is a constant.

Let $\omega = a^{n^2}$

We can be written as uvxyz, where

- (i) $|vxy| \leq n$
- (ii) $vy \neq \epsilon$
- (iii) If $i = 0, 1, 2, \dots$ $uv^i xy^i z \in L$

Here

$$\begin{aligned} |uvxyz| &= |a^{n^2}| = n^2 \\ uv^2 xy^2 z &= |uvxyz| + |vy| > n^2 \quad \text{--- (1)} \\ uv^2 xy^2 z &\leq n^2 + n \quad [\text{as } |vy| \leq n \text{ from (i)}] \\ &\leq n^2 + 2n + 1 \\ &\leq (n+1)^2 \quad \text{--- (2)} \end{aligned}$$

From (1) and (2) $n^2 < |uv^2 xy^2 z| \leq (n+1)^2$

But there is no square between n^2 and $(n+1)^2$.

Hence L is not a CFL.

4. Prove that $L = \{a^n b^m c^p | 0 \leq n < m < p\}$ is not CFL.

Solution:

- (i) Let $L = \{a^n b^m c^p | 0 \leq n < m < p\}$
- (ii) Let us pick up a word $\omega = a^n b^{n+1} c^{n+2}$, where n is a constant and [since $n < m < p$]
- (iii) ω can be rewritten as uvxyz, where
 - i) $|vxy| \leq n$
 - ii) $vy \neq \epsilon$

iii) If $i = 0, 1, 2, \dots$, $uv^i xy^i z \in L$

Let vy contains all the 3 symbols a, b, c

(i) If $v = ab$ and $y = c$

$$i = 2 \Rightarrow uv^i xy^i z \Rightarrow uv^2 xy^2 z = (ab)^2 c \dots \dots \dots (1)$$

(ii) If $v = a$ and $y = bc$

$$\text{At } i = 2, uv^2 xy^2 z = a^2(bc)^2 \dots \dots \dots (2)$$

From (1) and (2) $uv^i xy^i z \notin L$ since the powers of abc does not match with the L

$\therefore L$ is not a CFL.

5. Prove that $L = \{\omega\omega \mid \omega \text{ is in } (0+1)^*\}$ is not a CFL

Solution:

Let us assume that L is a CFL.

Let $\omega = 0^n 1^n 0^n 1^n$, where n is a constant and $L = \{\omega\omega \mid \omega \text{ is in } (0+1)^*\}$

ω can be rewritten as $uvxyz$, where

i) $|vxy| \leq n$

ii) $vy \neq \epsilon$

iii) If $i = 0, 1, 2, \dots$, $uv^i xy^i z \in L$

Let vy takes all the values of 0 and 1.

1) If $v = 0^1$ and $y = 0^{n-1} 1^n 0^n 1^n$

$$\text{At } i = 2, uv^2 xy^2 z = (0)^2 (0^{n-1} 1^n 0^n 1^n) \dots \dots \dots (1)$$

2) If $v = 0^n 1^n 0^n 1^{n-1}$; $y = 1$

$$\text{At } i = 2, uv^2 xy^2 z = (0^n 1^n 0^n 1^{n-1})^2 (1)^2 \dots \dots \dots (2)$$

From (1) and (2); $uv^i xy^i z \notin L$ since the powers of 0 and 1 and $L = \omega\omega$ does not match.

6. Prove that $L = \{0^i 1^j 2^i 3^j \mid i \geq 1 \text{ and } j \geq 1\}$ is not context – free.

(AU – Dec 2003, Dec 2005)

Solution:

Let us assume that L is CFL.

Let $\omega = 0^n 1^n 2^n 3^n$, where n is a constant.

4.4 Theory of Computation

Let ω is rewritten as $uvxyz$ where

- i) $|vxy| \leq n$
- ii) $vy \neq \epsilon$
- iii) $uv^i xy^i z \in L$ for $i = 0, 1, 2, \dots$

Let vy takes the symbols 012 or 123

- (i) if $v = 01$ and $y = 2$
At $i = 2$, $uv^2 xy^2 z = (01)^2 (2)^2 \epsilon$ ----- (1)
- (ii) if $v = 12$ and $y = 3$
At $i = 2$, $uv^2 xy^2 z = (12)^2 (3)^2 \emptyset$ ----- (2)

From (1) and (2) there are unequal number of 0 1 2 and 3 in (1) and unequal numbers of 1 2 3 and 0 in (2).

Hence L is not a CFL.

PROPERTIES OF CFL

(AU – Nov/Dec 2013, Nov/Dec 2010, May/June 2013)

The properties of CFL are classified into 2 groups.

- Closure properties
- Algorithm / Decision properties

Closure properties

A context-free language is closed under the following operations.

- Union
- Concatenation
- Kleene star

Context free language is not closed under

- Intersection
- Complementation
- Reversal

CFL is closed under union

(AU – Dec 2007, Dec 2009)

Theorem

If L_1 and L_2 are CFL, the $L_1 \cup L_2$ is also CFL.

Proof

Let L_1 be a CFL generated by a CFG, $G_1 = (V_1, T_1, P_1, S_1)$

Similarly, L_2 is another CFL generated by a CFG, $G_2 = (V_2, T_2, P_2, S_2)$

We can combine, the two grammar G_1 and G_2 into one grammar, G that will generate the union of the two languages.

- A new start symbol, S is added to G .
- Two new productions are added to G

$$S \rightarrow S_1$$

$$S \rightarrow S_2$$

- The grammar G can be written as

$$G = (V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \rightarrow S_1 | S_2\}, S)$$

- S can generate a string of terminals by selecting start symbol S_1 of G_1 or start symbol S_2 of G_2 .
- Thus, S can generate a string from L_1 or from L_2

$$L(G) = L_1 \cup L_2$$

CFL is closed under concatenation

Theorem

If L_1 and L_2 are context-free language, then $L_1 L_2$ is a context-free language.

Proof

Let L_1 be a CFL with the grammar $G_1 = (V_1, T_1, P_1, S_1)$

Let L_2 be a CFL with the grammar $G_2 = (V_2, T_2, P_2, S_2)$

A new language L is constructed by combining the two grammars G_1 and G_2 into one grammar G that will generate the concatenation of the two languages.

- A new start symbol, S is added to G .
- A new production is added to G

$$S \rightarrow S_1 S_2$$

The start symbol S will generate a string, ω of the form:

$$\omega = \omega_1 \omega_2$$

Where $\omega_1 \in L_1 \omega_2 \in L_2$

The grammar G can be written as

$$G = (V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}, S)$$

4.4 Theory of Computation

CFL is closed under kleene star

Theorem

If L is a context free language, then L^* is a context free language.

Proof

Let L_1 be a CFL with the grammar, $G_1 = (V_1, T_1, P_1, S_1)$.

A new language L is constructed from L_1 , which is L_1^* .

$$L = L_1^*$$

A new start symbol, S is added to grammar, G of L .

Two new productions are added to G .

$$S \rightarrow S S_1$$

$$S \rightarrow \epsilon$$

The production $S \rightarrow SS_1 | \epsilon$ will generate a string ω^* where $\omega \in L_1$.

The grammar G can be written as

$$G = (V_1, T_1, P \cup \{S \rightarrow S S_1 | \epsilon\}, S)$$

CFL is not closed under intersection

Theorem

Context free language are not closed under intersection.

Proof

Let us consider two context free languages L_1 and L_2 .

Where

$$L_1 = \{a^n b^n c^m \mid n, m \geq 0\}$$

$$L_2 = \{a^m b^n c^n \mid n, m \geq 0\}$$

The language L_1 is a CFL with set of productions.

$$S \rightarrow AB$$

$$A \rightarrow aAb \mid \epsilon$$

$$B \rightarrow CB \mid \epsilon$$

The language L_2 is a CFL with set of productions.

$$S \rightarrow AB$$

$$A \rightarrow aA \mid \epsilon$$

$$B \rightarrow bBC \mid \epsilon$$

A string $\omega_1 \in L_1$ contains equal number of a"s and b"s. A string $\omega_2 \in L_2$ contains equal number of b"s and c"s.

A string $\omega \in L_1 \cap L_2$ will contain equal number of a"s and b"s and equal number of b"s and c"s.

The $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$. From pumping lemma for CFL, a string of the form $a^n b^n c^n$ cannot be generated by a CFG.

\therefore The class of CFL is not closed under intersection.

CFL is not closed under complementation

Theorem

The set of CFL is not closed under complementation.

Proof

This theorem can be proved through contradiction.

Let us assume me that CFL is closed under complementation.

If L_1 is context-free, then L_1^\complement is also context-free.

If L_2 is context-free, then L_2^\complement is also context-free.

Now $L_1 \cap L_2$ can be written $(L_1^\complement \cup L_2^\complement)$, which should also be a context-free.

Since, $L_1 \cap L_2$ is not generated to be context-free our assumption that CFL is closed under complementation is wrong.

CFL is closed under reversal

Theorem

If L is a CFL, then so is L^R .

Proof

Let us assume that $L = L(G)$ for some CFG, $G = (V, T, P, S)$.

A grammar generating reverse L is given by $G^R = (V, T, P^R, S)$

$P^R \rightarrow$ can be obtained from P by reversing the right hand side of the production.

If $A \rightarrow \alpha$ is a production in P , then

$A \rightarrow \alpha^R$ is a production in P^R

Algorithm / decision properties

These are algorithms for testing.

4.4 Theory of Computation

The method of finding whether a string ω belongs to $L(G)$ is known as parsing. There are two types of parsing.

- Top-down parsing
- Bottom-up parsing

TURING MACHINE

A Turing machine is an „automatic machine“ that manipulates the input strings according to the transition rules.

It was invented by Alan Turing in 1936.

Turing machine is also called an „Intelligent machinery“ is a hypothetical device.

Alan Turing, named Turing machine as „a-machine“, was stimulated by his lecturer Newman’s works and words on „mechanical process“, made him invented this when he was doing master degree in UK.

Turing machine is the base for inventing various other machines such as

- The Enigma {secret code encryption machine}
- Automatic computing engine [AEC]
- EDVAC

Turing’s „a-machine“ is the base of even today’s digital computer.

Due to its usage in several ways, Turing machine became the most powerful general model of computation.

Further, TM is the mathematical model of partial recursive functions and in computing undecidability.

Definitions of Turing machines

Description of a TM/Informal definition

Turing machine is composed of

1. Input tape with tape head
 2. Finite set of transition rules
 3. Output tape with tape head
- The input tape contains the input symbols. The tape is divided into a number of cells, each cell storing one symbol.
 - The input tape can also serve as output tape, that contains the resultant value of the operation done by TM.

- The tape head is used to read a symbol from the input tape or write a symbol on the output tape.
- The tape is capable of reading/writing one symbol at a time.
- Several heads can be used to perform parallel processes/operations on the head.
- The transition rules are responsible for performing the required operations using its rules, and based on the input read.
- It performs state transitions based on the processing done.

Nature of TM

The Turing machines are computing machines that are more powerful than the FA and PDA.

It is given as

$$\text{Finite automata} \subseteq \text{Determination PDA} \subseteq \text{NPDA} \subseteq \text{TM}$$

FA, DPDA, NPDA are less powerful than TM since these machines have no control over the input and cannot modify their own inputs.

But TM can control and modify their inputs on the tape.

Formal definition of TM

A Turing machine, M is a 7-tuple given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where,

$Q \rightarrow$ Finite set of states

$\Sigma \rightarrow$ Finite set of inputs symbols on input tape

$\Gamma \rightarrow$ Finite set of tape symbols $[\Sigma \cup B]$ [where $B \rightarrow$ blank symbol]

$\delta \rightarrow$ Transition function given by

$(q, a) = (q', b, M)$ [$M \rightarrow$ movement \rightarrow left, right, no movement]

$q_0 \rightarrow$ Initial state $[q_0 \in Q]$

$B \rightarrow$ Blank symbol representing empty cell $[B \in \Gamma]$

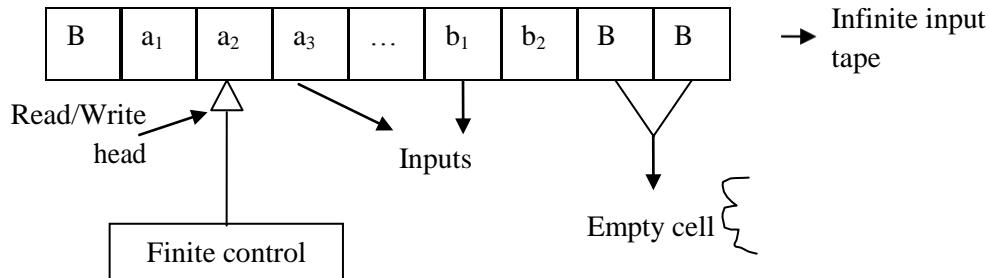
$F \rightarrow$ Set of final states $[F \subseteq Q]$

Model of Turing machine

- Turing machine shall be thought of as a finite state machine with a read/write head.
- The input is stored on an input tape that is divided into a number of individual cells.

4.5 Theory of Computation

- The cells contains blank symbol, „B“ when there is no input in it.
- Each cell can store one symbol in it.



- The read/write head of the system reads from and writes data to the input tape.
- The head performs its operation on one cell at a time.
- The movement of the head can be
 - Left (moving backward)
 - Right (moving forward)
 - None (no movement)
- The head is capable of
 - Reading a symbol
 - Modifying a symbol
 - Move Previous(L) / Move Next(R)
 - Halt

INSTANTANEOUS DESCRIPTION FOR TM

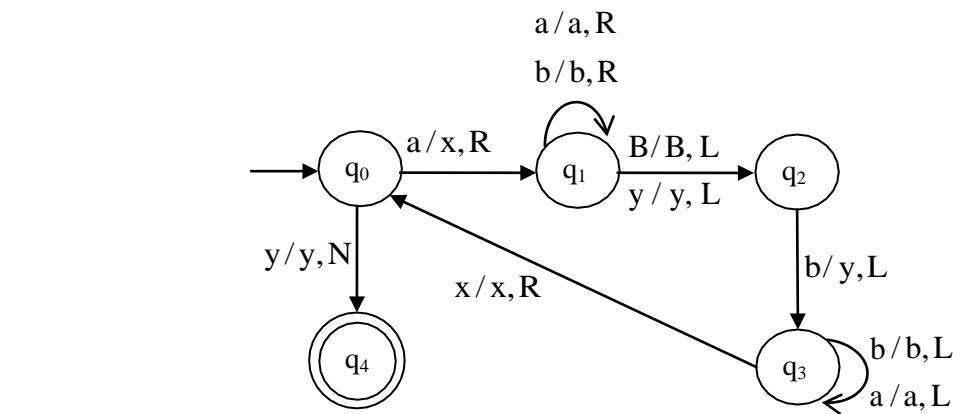
Instantaneous description for TM is the snapshot of how the input string is processed by the Turing machine.

It describes,

- The input string
- Position of the head
- State of the machine

Example

For the Turing machine given below, provide the instantaneous description for the string aabb.



Instantaneous description

$aabbB \xrightarrow{q_0} xabbB \xrightarrow{q_1} xabbB \xrightarrow{q_1} |xabbB \xrightarrow{q_1} \|abbB$
 $\uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow$
 $q_0 \quad q_1 \quad q_1 \quad q_1 \quad q_1$
 $\vdash x\|bbB \vdash xabyB \vdash xabyB \vdash xabyB \vdash xabyB$
 $\uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow$
 $q_1 \quad q_3 \quad q_3 \quad q_3 \quad q_0$
 $\vdash xxbyB \vdash xxbyB \vdash xxbyB \vdash xxyyB \vdash xxyyB$
 $\uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow$
 $q_1 \quad q_1 \quad q_2 \quad q_3 \quad q_0$
 $\vdash xxyyB$ [String Accepted]
 \uparrow
 q_4

PROBLEMS

1. Design a Turing machine that accepts all strings of the form $a^n b^n$ for $n \geq 1$ and rejects all other string.

AU – JUN 2006, JUN 2009, DEC 2009

Solution:

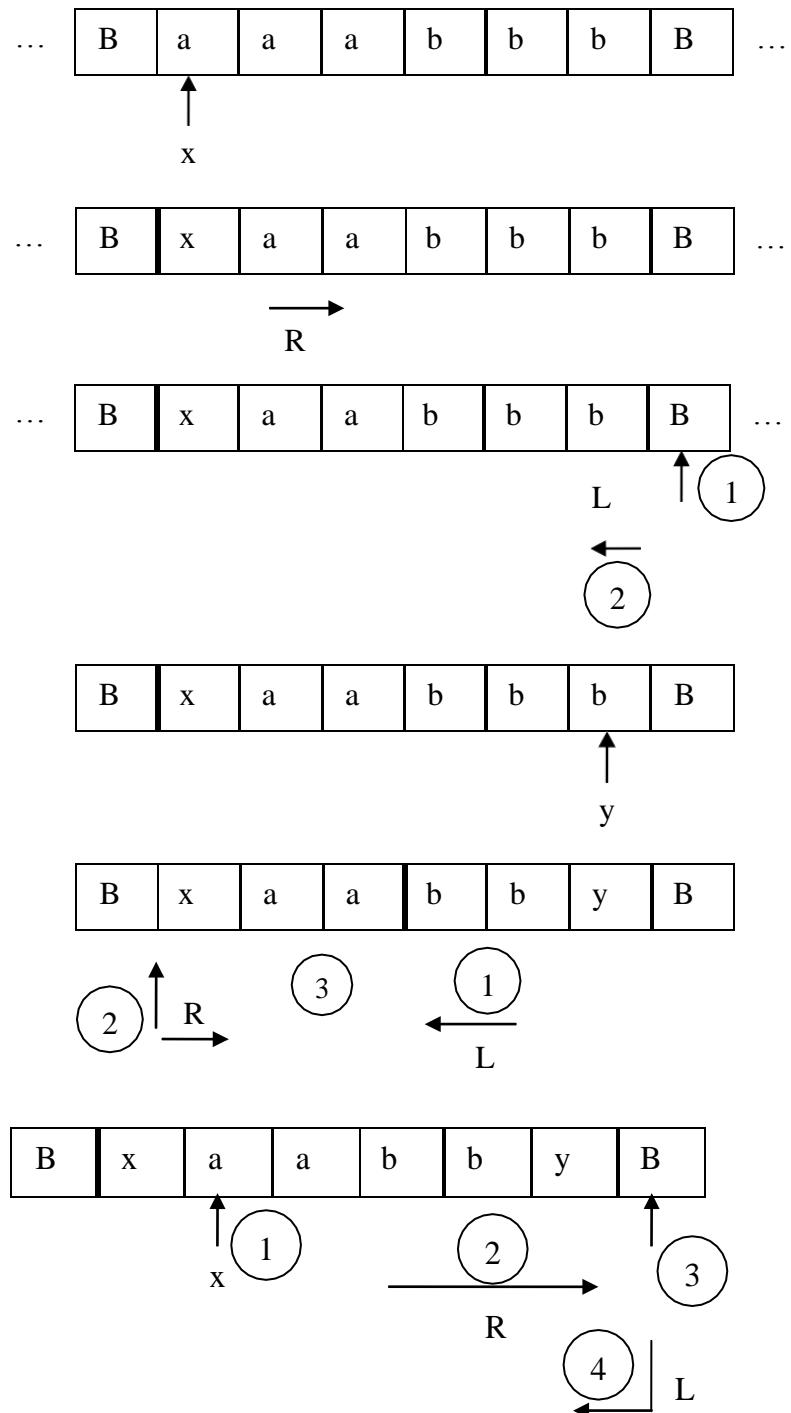
Algorithm

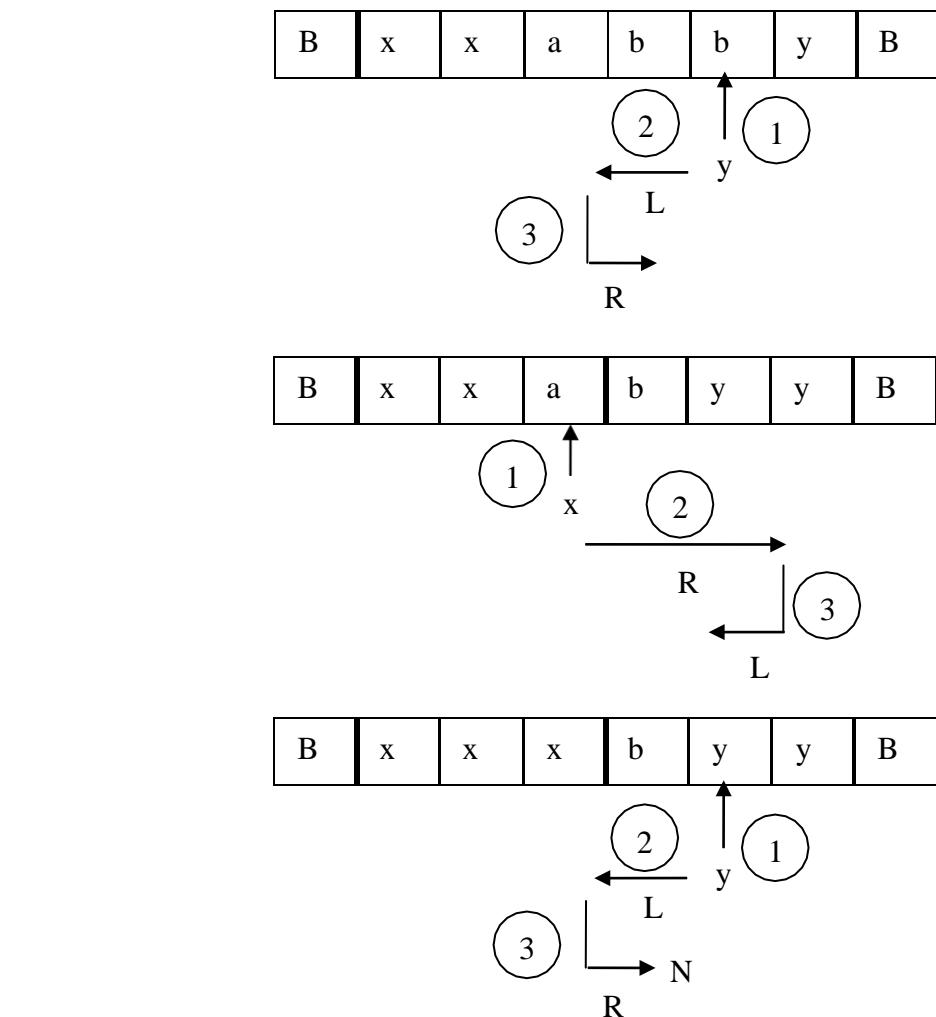
1. Change the leftmost a to x
2. Move forward right side and change the rightmost b to y .
3. Move left and change the leftmost a , which is at position immediate right of x , to x .
4. Move right and change the rightmost b , which is at immediate left of y , to y .
5. Repeat step 3 and 4 until there are no more b 's equal to a 's.
6. Halt the TM.

4.5 Theory of Computation

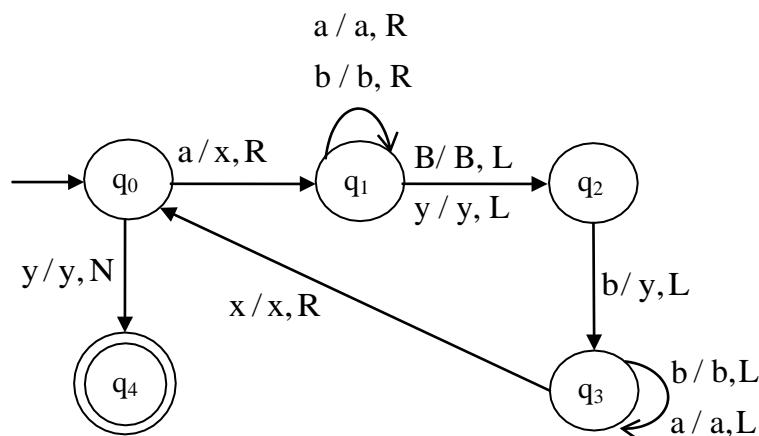
Instantaneous description

Let $n = 3 \Rightarrow \omega = a^3 b^3 = aaabbb$





Turing machine



Description of the TM

The Turing machine, M is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where,

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

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

$$\Gamma = \{a, b, B, x, y\}$$

$$q_0 = \{q_0\}$$

$$B = \{B\}$$

$$F = \{q_4\}$$

δ is given by

$Q \setminus \Sigma \cup \Gamma$	a	b	x	y	B
$\rightarrow q_0$	(q_1, x, R)	-	-	(q_4, y, N)	-
q_1	(q_1, a, R)	(q_1, b, R)	-	(q_2, y, L)	(q_2, B, L)
q_2	-	(q_3, y, R)	-	-	-
q_3	(q_3, a, L)	(q_3, b, L)	(q_0, x, R)	-	-
* q_4	ϕ	ϕ	ϕ	ϕ	ϕ

Here, $q_0 \rightarrow$ replaces leftmost a by x.

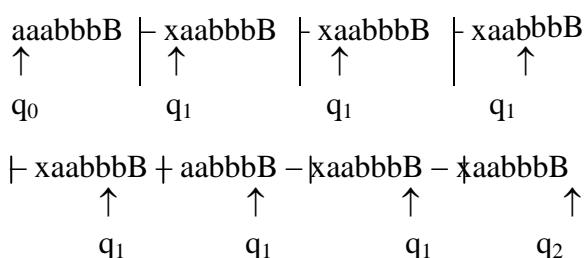
$q_1 \rightarrow$ skips a, b, and moves right until first B or first y.

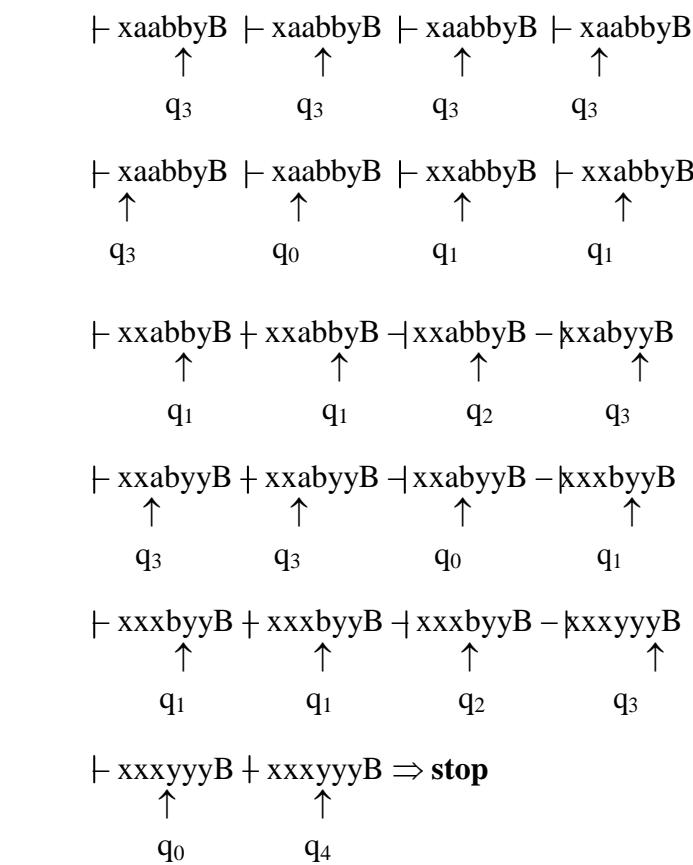
$q_2 \rightarrow$ replaces rightmost b by y.

$q_3 \rightarrow$ skips a, b, and moves left till first x is replaced and enters q_0 .

$q_4 \rightarrow$ halts if a valid $a^n b^n$ string is processed.

Instantaneous description for the string, $\omega = "aaabbb"$





2. Design a TM that perform right shift over $\Sigma = \{0, 1\}$.

Solution:

To perform: Shift the input string right by one place.

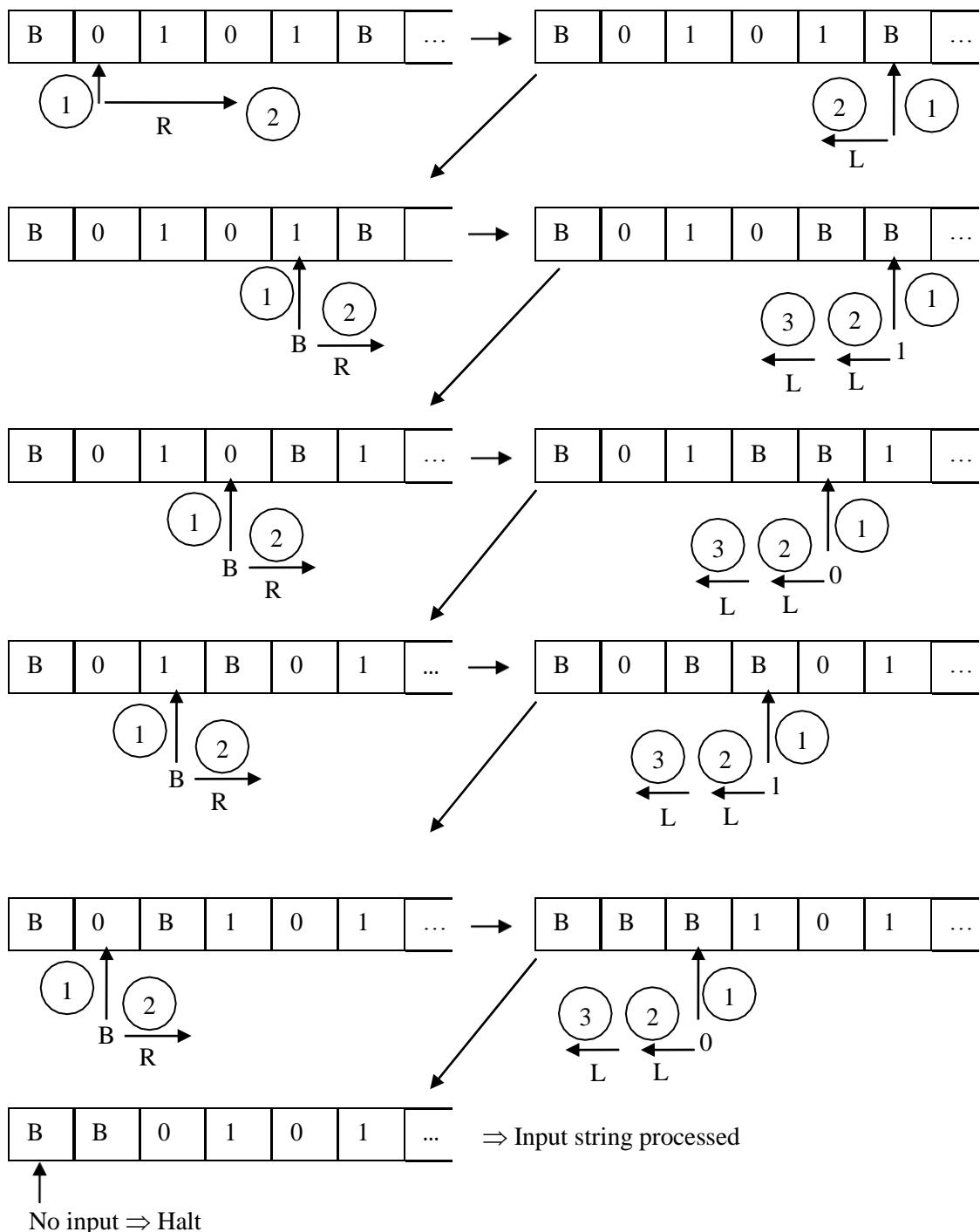
Algorithm

1. Move right side to the last character from the initial character.
2. If the character is „0“ replace it with „B“ and move one step right to replace the immediate „B“ to „0“.
3. If the character is „1“, replace it with „B“ and move one step right to replace the immediate „B“ to „1“.
4. After processing as above, move left one step to perform step 2 or 3 on the next right most unprocessed character.
5. Perform step – 4 until all the characters are processed.

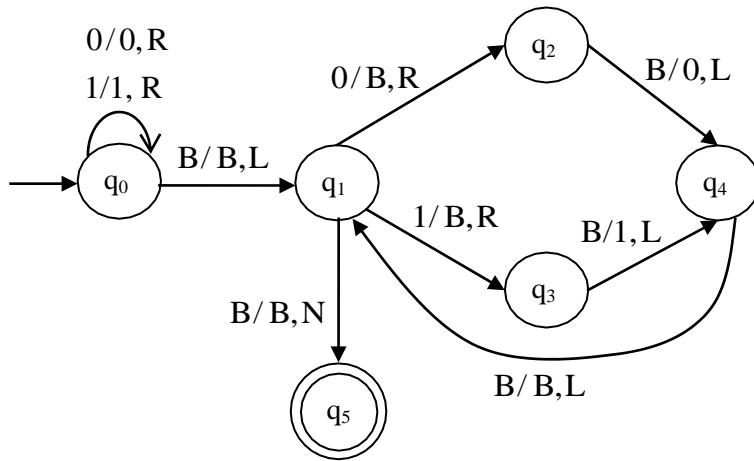
4.5 Theory of Computation

Instantaneous description

Let $\omega = 0101$



Turing Machine



Description of Turing Machine

The turing machine is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where,

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

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

$$\Gamma = \{0, 1, B\}$$

$$q_0 = \{q_0\}$$

$$B = \{B\}$$

$$F = \{q_5\}$$

δ is given by

$Q \setminus \Sigma \cup \Gamma$	0	1	B
$\rightarrow q_0$	$(q_1, 0, R)$	$(q_0, 1, R)$	(q_1, B, L)
q_1	(q_2, B, R)	(q_3, B, R)	(q_5, B, N)
q_2	-	-	$(q_4, 0, L)$
q_3	-	-	$(q_4, 1, L)$
q_4	-	-	(q_1, B, L)
$* q_5$	ϕ	ϕ	ϕ

Here,

$q_0 \rightarrow$ skips the initial 0's and 1's and moves the head of the TM to the last character.

$q_1 \rightarrow$ the unprocessed rightmost character is replaced by B.

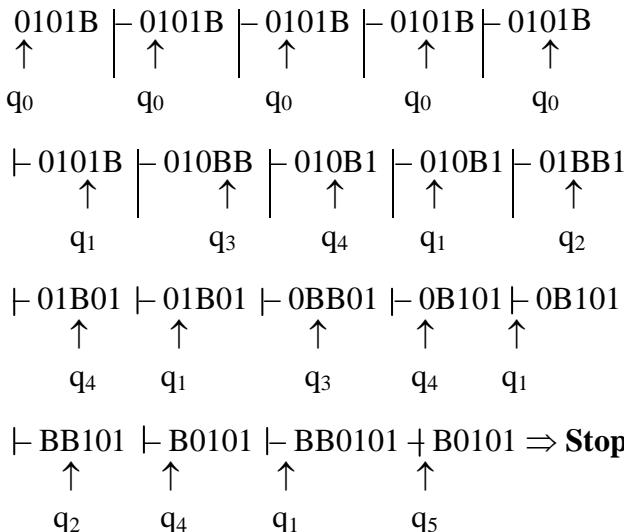
$q_2 \rightarrow$ on processing the input „0“, q_2 replaces immediate the „B“ as „0“ and takes left.

$q_3 \rightarrow$ on processing „1“, q_3 replaces immediate the „B“ as „1“ and moves left.

$q_4 \rightarrow$ takes a left position to process the next character

$q_5 \rightarrow$ halts if no more inputs characters are available.

Instantaneous description for $\omega = "0101"$



2. Construct a Turing machine to make a copy of a string over $\Sigma = \{0, 1\}$

Solution:

Algorithm

Given

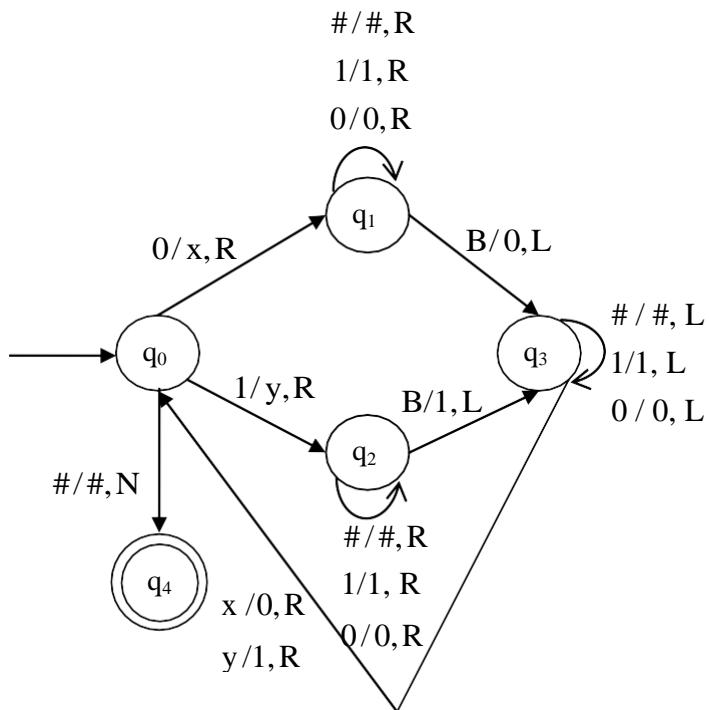
A sequence of input symbols followed by „#“ at the end.

Steps

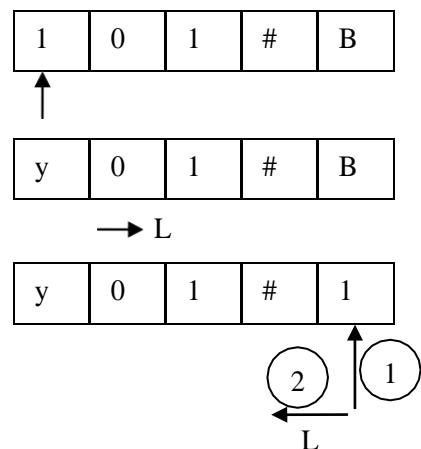
1. If the symbol is „0“, move right until „B“ is reached. Mark „0“ as „x“ to indicate that it is processed.
2. Replace the symbol „B“ by „0“. Move left until „x“ is reached. Mark x as „0“ and move right.

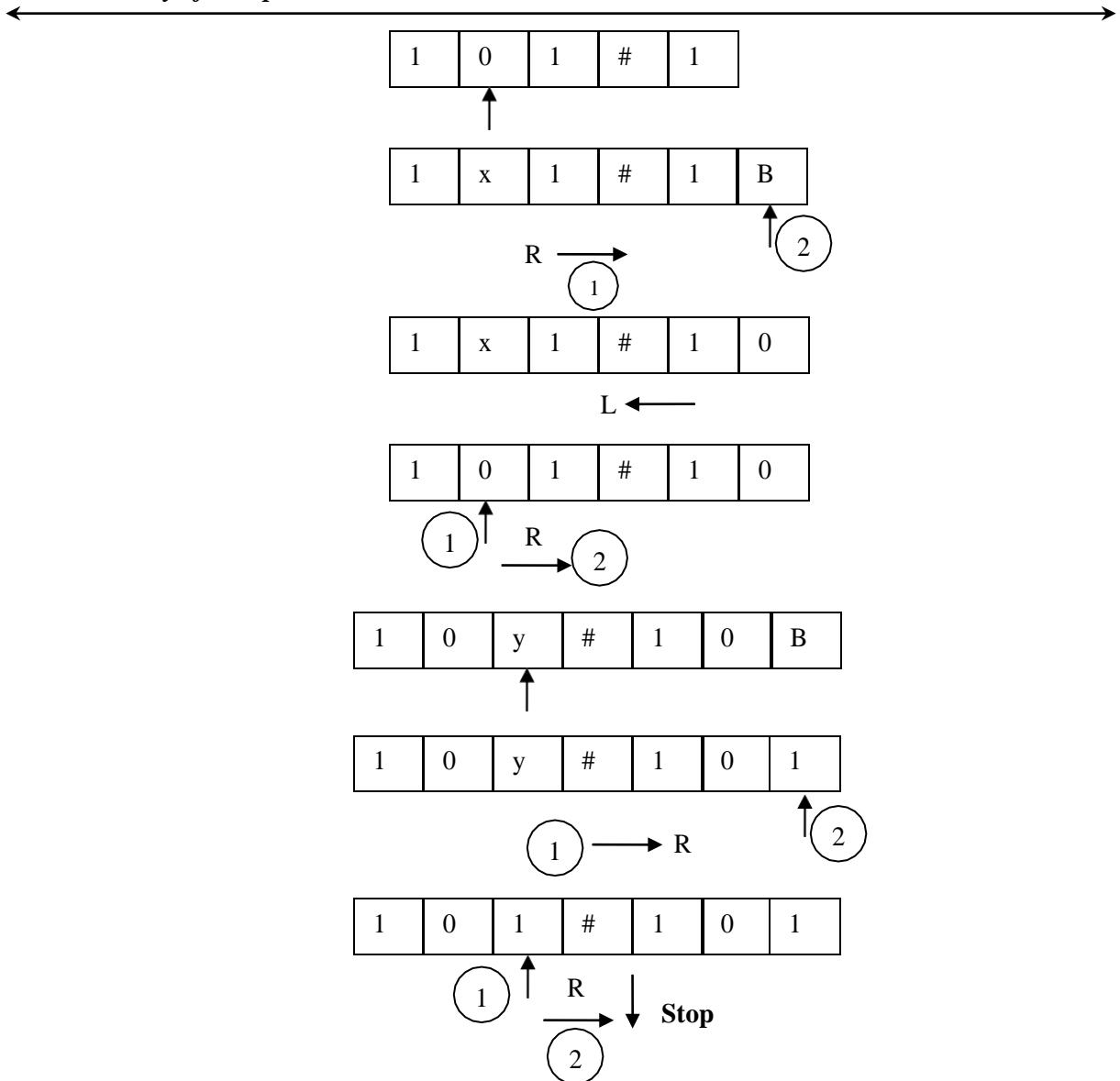
3. If the symbol is „1“, move right until „B“ is reached. Mark „1“ as „y“ to indicate that it is processed.
4. Replace „B“ by 1 and move left until „y“ is reached. Mark „y“ as 1 and move right.
5. Process step 1 and 2 if „0“ is the input symbol; else perform step 4 and 5 to process „y“.
6. Stop and reach final state if „#“ is reached.

Turing machine



Representation of „101“





Description of Turing machine

The Turing machine, M is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where,

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

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

$$\Gamma \equiv \{0, 1, x, y, \#, B\}$$

$$\mathbf{q}_0 \equiv \{\mathbf{q}_0\}$$

$$B = \{B\}$$

$$F = \{q_4\}$$



δ is given by

$\Gamma \backslash Q$	0	1	#	B	x	y
$\rightarrow q_0$	(q_1, x, R)	(q_2, y, R)	($q_4, \#, N$)	-	-	-
q_1	($q_1, 0, R$)	($q_1, 1, R$)	($q_1, \#, R$)	($q_3, 0, L$)	-	-
q_2	($q_2, 0, R$)	($q_2, 1, R$)	($q_2, \#, R$)	($q_3, 1, L$)	-	-
q_3	($q_3, 0, L$)	($q_3, 1, L$)	($q_3, \#, L$)	-	($q_0, 0, R$)	($q_0, 1, R$)
* q_4	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ

Instantaneous description for $\omega = 101$

$$\begin{array}{ccccccccc}
 & 101\#B + y01\#B - y01\#B - |y01\#B - |y01\#B \\
 & \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \\
 & q_0 \quad q_2 \quad q_2 \quad q_2 \quad q_2 \\
 \\
 & \vdash y01\#1 \vdash y01\#B \vdash y01\#B \vdash y01\#B \vdash 101\#1 \\
 & \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \\
 & q_3 \quad q_3 \quad q_3 \quad q_3 \quad q_0 \\
 \\
 & \vdash 1x1\#1 \vdash 1x1\#1 \vdash 1x1\#1B \vdash 1x1\#1B \vdash 1x1\#10 \\
 & \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \\
 & q_1 \quad q_1 \quad q_1 \quad q_3 \quad q_3 \\
 \\
 & \vdash 1x1\#10 + 1x1\#10 \vdash 101\#10 - |10y\#10 \\
 & \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \\
 & q_3 \quad q_3 \quad q_0 \quad q_2 \\
 \\
 & \vdash 10y\#10 + 10y\#10B \vdash 10y\#10B - |10y\#10 \\
 & \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \\
 & q_2 \quad q_2 \quad q_2 \quad q_3 \\
 \\
 & \vdash 10y\#101 + 10y\#101 \vdash 10y\#101 - |101\#101 \\
 & \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \\
 & q_3 \quad q_3 \quad q_3 \quad q_0 \\
 \\
 & \vdash 101\#101 \\
 & \uparrow \quad \Rightarrow \text{stop} \\
 & q_4
 \end{array}$$

3. Design a TM to reserve a string over $\Sigma = \{a, b\}$.

Solution:

Algorithm

1. Move to the last symbol, replace x for a or x for b and move right to convert the corresponding B to „a“ or „b“ accordingly.

- ↔
2. Move left until the symbol left to x is reached.
 3. Perform step – 1 and step – 2 until „B“ is reached while traversing left.
 4. Replace every x to B to make the cells empty since the reverse of the string is performed by the previous steps.

Instantaneous description using the below given TM

Let $\omega = abb$

$abbB + abbB \vdash abbB -|abbB -\#bbB$
 ↑ ↑ ↑ ↑ ↑
 q₀ q₀ q₀ q₀ q₁

$\vdash abxB + abxbB \vdash abxbB \vdash axxbB$
 ↑ ↑ ↑ ↑
 q₃ q₄ q₁ q₃

$\vdash axxbB \vdash axxbB \vdash axxbbB \vdash axxbbB$
 ↑ ↑ ↑ ↑
 q₃ q₃ q₄ q₄

$\vdash axxbbB + xxxbbB \dashv xxxbbB$
 ↑ ↑ ↑
 q₁ q₂ q₂

$\vdash xxxbbB \vdash xxxbbB \vdash xxxbbB + xxxbbaB$
 ↑ ↑ ↑ ↑
 q₂ q₂ q₂ q₄

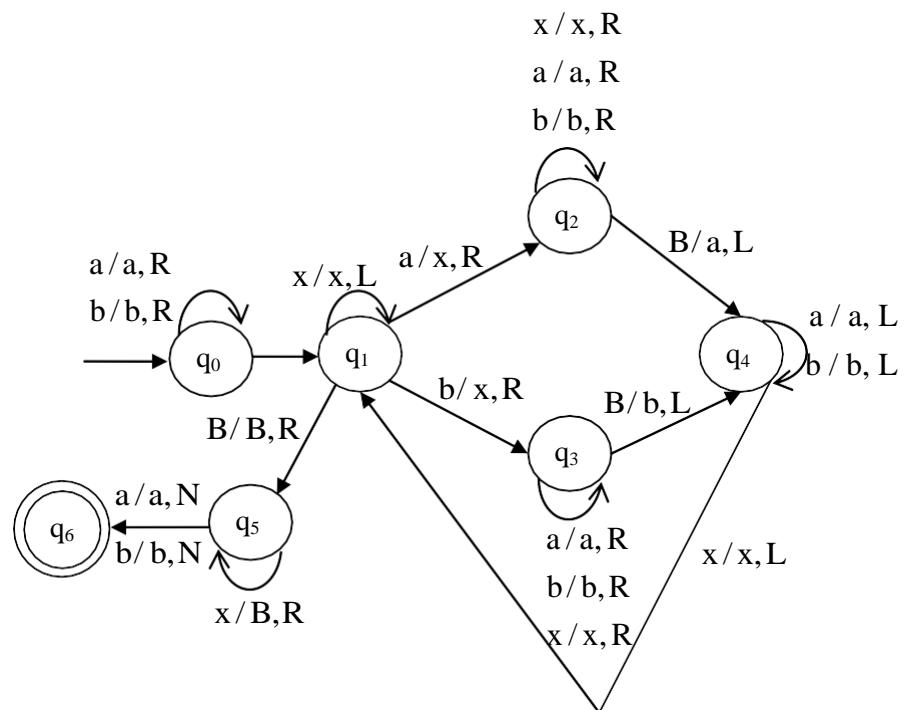
$\vdash xxxbbaB \vdash xxxbbaB \vdash xxxbbaB$
 ↑ ↑ ↑
 q₄ q₄ q₁

$\vdash BxxxbbaB \vdash BxxxbbaB \vdash BxxxbbaB$
 ↑ ↑ ↑
 q₁ q₁ q₅

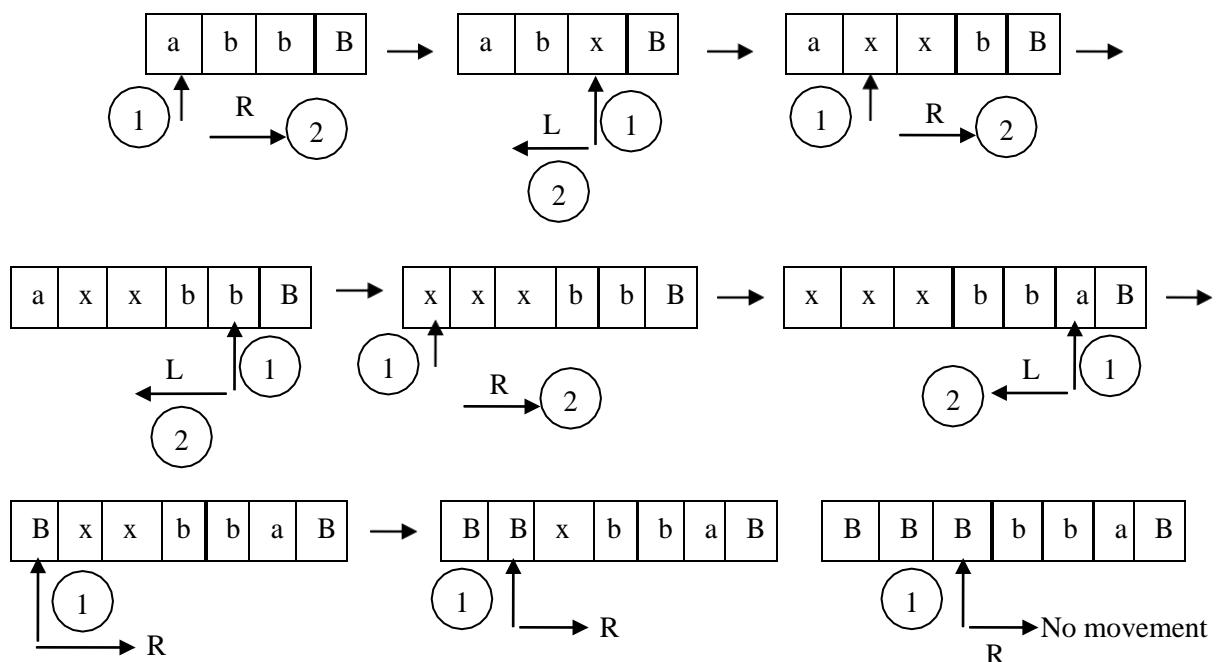
$\vdash BBxxxbbaB \vdash BBBxbbaB \vdash BBBBbbaB$
 ↑ ↑ ↑
 q₅ q₅ q₅

$\vdash BBBBbbaB$
 ↑ ⇒ Halt
 q₆

Turing Machine



Representation for $\omega = abb$



4.6 Theory of Computation

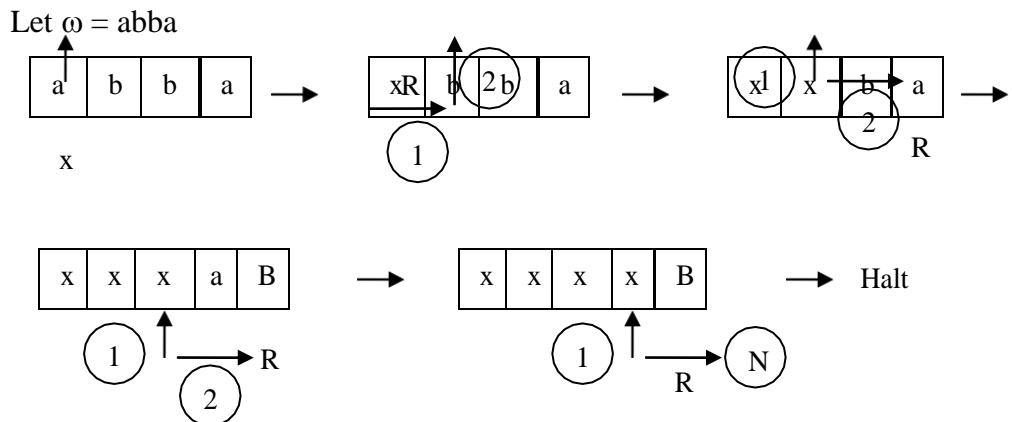
4. Design a Turing machine to check if there are equal number of a's and b's over {a, b}

Solution:

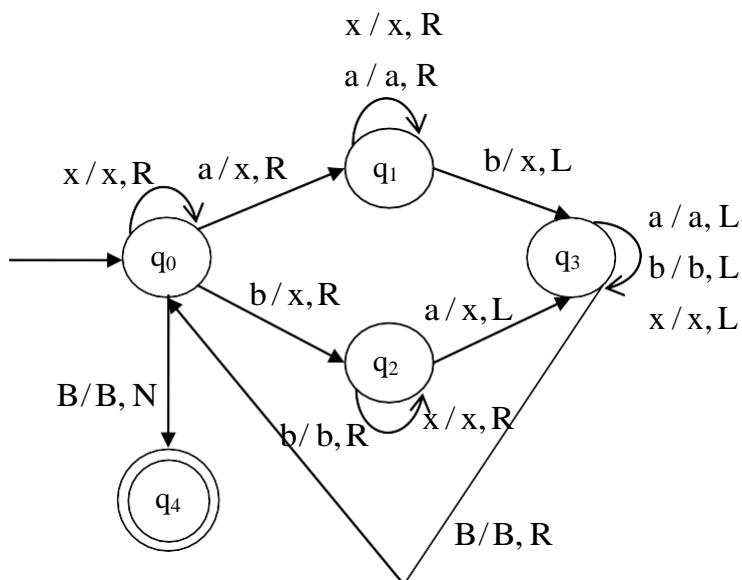
Algorithm

1. Process the first symbol
2. If it is „a“, move right to locate the first „b“, and replace both byx.
3. If „b“ occurs, replace it by x and move forward to reach the first „a“ replace „a“ by x.
4. Perform steps 2 and 3 until all inputs are processed.

Representation



Turing Machine



Instantaneous description for $\omega = abba$

$$\vdash \text{BabbaB} + \text{BxbbaB} -| \text{BxxbaB} -| \text{BxxbaB}$$

↑ ↑ ↑ ↑
q₀ q₁ q₃ q₃

$$\vdash \text{BxxbaB} + \text{BxxbaB} + \text{BxxbaB} -| \text{Bxxxab}$$

↑ ↑ ↑ ↑
q₀ q₀ q₀ q₂

$$\vdash \text{BxxxxB} + \text{BxxxxB} -| \text{BxxxxB} -| \text{BxxxxB}$$

↑ ↑ ↑ ↑
q₃ q₃ q₃ q₃

$$\vdash \text{BxxxxB} + \text{BxxxxB} -| \text{BxxxxB} -| \text{BxxxxB}$$

↑ ↑ ↑ ↑
q₀ q₀ q₀ q₀

$$\vdash \text{BxxxxB} + \text{BxxxxB} \Rightarrow \text{Halt.}$$

↑ ↑
q₀ q₄

The Turing machine, M is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where,

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

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

$$\Gamma = \{a, b, x, B\}$$

$$q_0 = \{q_0\}$$

$$B = \{B\}$$

$$F = \{q_0\}$$

$\delta \Rightarrow$ TM as mentioned above.

δ is given by,

Q	a	B	x	B
$\rightarrow q_0$	(q ₁ , x, R)	(q ₂ , x, R)	(q ₀ , x, R)	(q ₄ , B, N)
q ₁	(q ₁ , a, R)	(q ₃ , x, L)	(q ₁ , x, R)	-
q ₂	(q ₃ , x, L)	(q ₂ , b, R)	(q ₂ , x, R)	-
q ₃	(q ₃ , a, L)	(q ₃ , b, L)	(q ₃ , x, L)	(q ₀ , B, R)
* q ₄	ϕ	ϕ	ϕ	ϕ

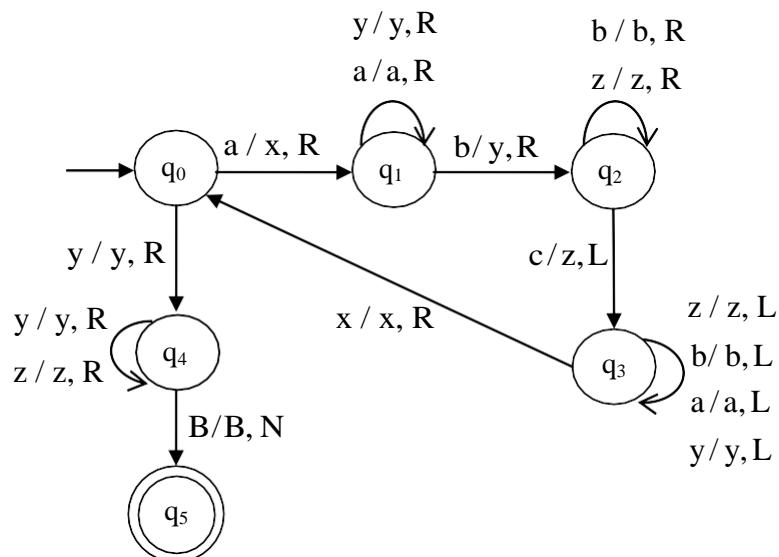
5. Design a TM that recognizes strings of the form $a^n b^n c^n \mid n \geq 1$ over $\Sigma = \{a, b, c\}$.

Solution:

Algorithm

1. Process the leftmost „a“ and replace it by „x“.
2. Move right until the leftmost „b“ is reached. Replace it by „y“.
3. Move right until the leftmost „c“ is reached. Replace it by „z“.
4. Move left to reach the leftmost „a“ and perform steps 1, 2 and 3 ($n - 1$) times.
5. Halt if there are „n“ number of x, y, z.

Turing Machine



The Turing machine, M is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

$$\Sigma = \{a, b, c\}$$

$$\Gamma = \{a, b, c, x, y, z, B\}$$

$\delta \Rightarrow$ Given by the transition diagram of the TM

$$q_0 = \{q_0\}$$

$$B = \{B\}$$

$$F = \{q_5\}$$

← →

δ is given by

Q	Γ	a	b	c	x	y	z	B
$\rightarrow q_0$	(q_1, x, R)	-	-	-	(q_4, y, R)	-	-	-
q_1	(q_1, a, R)	(q_2, y, R)	-	-	(q_1, y, R)	-	-	-
q_2	-	(q_2, b, R)	(q_3, z, R)	-	-	(q_2, z, R)	-	-
q_3	(q_3, a, L)	(q_3, b, L)	-	(q_0, x, R)	(q_3, y, L)	(q_3, z, L)	-	-
q_4	-	-	-	-	(q_4, y, R)	(q_4, z, R)	(q_5, B, N)	
* q_5	ϕ	ϕ						

Instantaneous description for $\omega = a^2 b^2 c^2$

$\omega = aabbcc$

$$\vdash BaabbccB + BxabbccB \dashv BxabbccB$$

$$\begin{array}{ccc} \uparrow & \uparrow & \uparrow \\ q_0 & q_1 & q_1 \end{array}$$

$$\vdash BxaybccB + BxaybccB \dashv BxaybzcB$$

$$\begin{array}{ccc} \uparrow & \uparrow & \uparrow \\ q_2 & q_2 & q_3 \end{array}$$

$$\vdash BxaybzcB + BxaybzcB \dashv BxaybzcB$$

$$\begin{array}{ccc} \uparrow & \uparrow & \uparrow \\ q_3 & q_3 & q_3 \end{array}$$

$$\vdash BxaybzcB + BxxybzcB \dashv BxxybzcB$$

$$\begin{array}{ccc} \uparrow & \uparrow & \uparrow \\ q_0 & q_1 & q_1 \end{array}$$

$$\vdash BxxyyzcB + BxxyyzcB \dashv BxxyyzzB$$

$$\begin{array}{ccc} \uparrow & \uparrow & \uparrow \\ q_2 & q_2 & q_3 \end{array}$$

$$\vdash BxxyyzzB + BxxyyzzB \dashv BxxyyzzB$$

$$\begin{array}{ccc} \uparrow & \uparrow & \uparrow \\ q_3 & q_3 & q_3 \end{array}$$

$$\vdash BxxyyzzB + BxxyyzzB \dashv BxxyyzzB$$

$$\begin{array}{ccc} \uparrow & \uparrow & \uparrow \\ q_0 & q_4 & q_4 \end{array}$$

$$\vdash BxxyyzzB + BxxyyzzB \dashv BxxyyzzB \Rightarrow \text{Halt}$$

$$\begin{array}{ccc} \uparrow & \uparrow & \uparrow \\ q_4 & q_4 & q_5 \end{array}$$

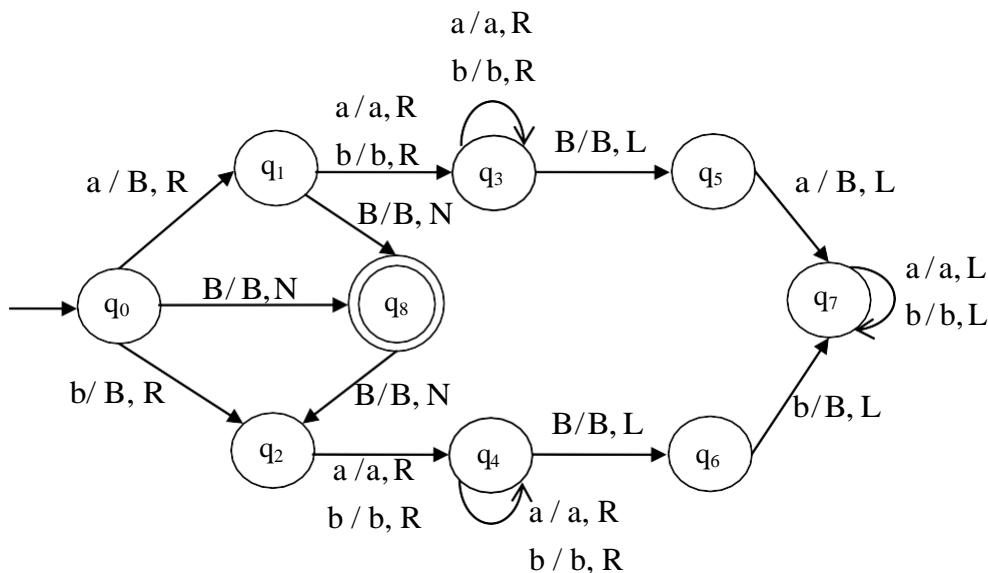
6. Design a TM which recognizes palindromes over $\Sigma = \{a, b\}$.

Solution:

Algorithm

1. If there is no input, reach the final state and halt.
2. If the input = „a“, then traverse forward to process the last symbol = „a“. convert both a“s to „B“.
3. Move left to read the next symbol.
4. If the input = „b“, replace it by B and move right to process its equivalent „B“ at the rightmost end.
5. Convert the last „b“ to „B“.
6. Move left and process step 2 – 5 until there are no more inputs to process.
7. If the machine reaches the final state after processing the entire input string, then the string is a palindrome halt the machine.

Turing Machine



The Turing machine, M is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$

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

$$\Gamma = \{a, b, B\}$$

$\delta \Rightarrow$ Given by the above mentioned transition diagram

$$q_0 = \{q_0\}$$

$$B = \{B\}$$

$$F = \{q_8\}$$

Instantaneous Description

Let $\omega = aabbaa$

$$\vdash BaabbaaB + BBabbaaB \dashv BBabbaaB - |BBabbaaB$$



$$\vdash BBabbaaB + BBabbaaB \dashv BBabbaaB - |BBabbaaB$$



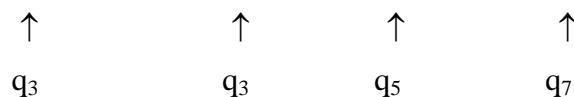
$$\vdash BBabbaBB + BBabbaBB \dashv BBabbaBB - |BBabbaBB$$



$$\vdash BBabbaBB + BBabbaBB \dashv BBBbbaBB - |BBBbbaBB$$



$$\vdash BBBbbaBB + BBBbbaBB \dashv BBBbbaBB - |BBBbbBBB$$



$$\vdash BBBbbBBB \dashv BBBbbBBB \dashv BBBbbBBB \dashv BBBBbBBB$$



$$\vdash BBBBbBBB \dashv BBBBbBBB \dashv BBBBbBBB \dashv BBBBbBBB$$



$$\vdash BBBBbBBB \Rightarrow \text{Halt}$$

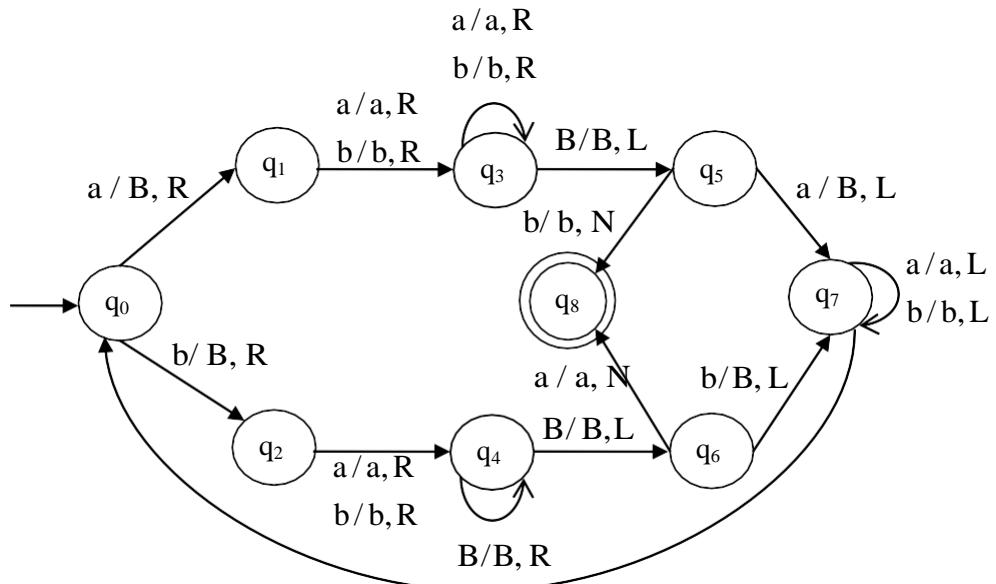


Hence the given string aabbaa is a palindrome.

7. Design a TM that recognizes the language of all non-palindromes over $\{a, b\}$.

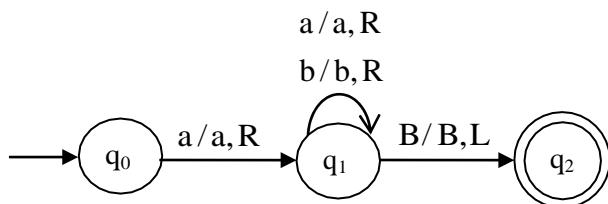
Solution:

Turing Machine



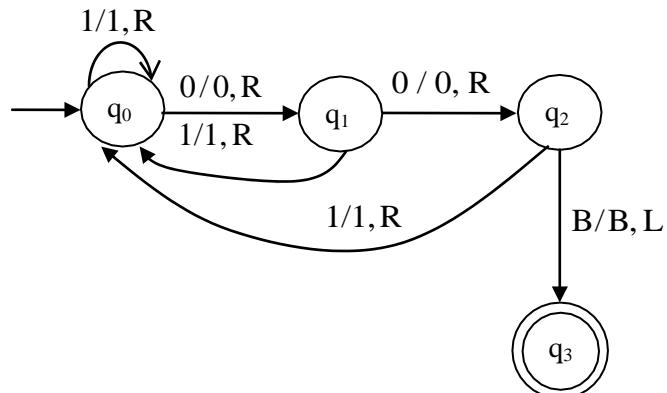
8. Design a TM that accepts $a(a+b)^*$ over $\Sigma = \{a, b\}$

Solution:



9. Construct a TM for $L = \{x \in \{0, 1\}^* \mid x \text{ ends in } 00\}$

Solution:



COMPUTABLE LANGUAGES AND FUNCTIONS

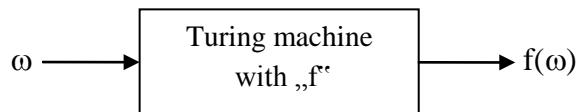
Turing machine is capable of performing several operations/functions.

It is capable of performing any sort of computations such as

- Addition
- Subtraction
- Multiplication
- Division
- 2''s complementation
- 1''s complementation
- Comparing two numbers
- Squaring a number
- GCD of numbers
- Finding binary equivalents.

A Turing machine M can compute a function, f from the subset of \sum^* .

The machine starts the computation corresponding to the input string, ω in the domain of „f“ and halts with the output string $f(\omega)$.



When „ ω “ is an input string that cannot be computed by the Turing machine, M with function f, then the TM should not accept ω .

The Turing machine, M can handle a function containing several variables.

Representation:

Computation of a function „f“ is represented as

$$f : \sum_1^* \rightarrow \sum_2^*$$

The function f is Turing computable by the machine,

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

With the transition function δ of the form,

$$(q_0, \omega) \xrightarrow[M]{*} (q_f, f(\omega))$$

where,

$q_0 \rightarrow$ Initial state ($q_0 \in Q$)

$\omega \rightarrow$ Input string, where $\omega \in \Sigma^*$.

$q_f \rightarrow$ Final state ($q_f \in F$)

$f(\omega) \rightarrow$ Output string after computation of „ ω “ by f ($f(\omega) \in \Sigma^*$).

Here, the inputs other than $\omega \in \Sigma^*$ will not be accepted by the TM.

Representation of a number

For simplicity, the unary numbers are represented by a single symbol let „0“.

The decimal numbers are represented as,

No input $\rightarrow B$

1 $\rightarrow 0$

2 $\rightarrow 00$

3 $\rightarrow 000$

4 $\rightarrow 0000$

5 $\rightarrow 00000$

.

.

.

$n \rightarrow$ „ n “ number of zeros [0ⁿ]

Computing numerical functions

Turing machines can also compute numerical functions.

Numerical functions computes string which are represented as above

It is defined as,

$$T = (Q, \{0\}, \Gamma, \delta, q_0, B, F)$$

With „ f “ being the numerical function defined by a set of natural numbers, N.

„ f “ is defined as

$$(q_0, 0^n) \xrightarrow[M]{*} (q_f, 0^{f(n)})$$

where,

$q_0 \rightarrow$ initial state

$0^n \rightarrow$ input string

$q_f \rightarrow$ final state

$0^{f(n)} \rightarrow$ output of M, after computing 0^n in „f“

PROBLEMS

- 1. Design a Turing machine to perform 2's complement of a number over $\Sigma = \{0, 1\}$ (Binary number)**

Solution:

Algorithm

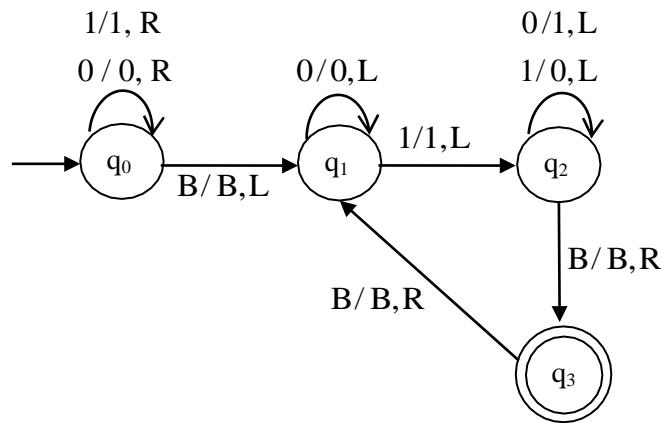
1. Traverse right and locate the rightmost bit
2. If the bit = „0“, perform no replacement and move left
3. If the bit = „1“ perform no change and move left
4. If the next bit symbol = „0“, replace it by „1“ and move left.
5. Else if the next bit symbol = „1“, replace it by „0“ and move left.
6. Perform steps 4 and 5 until all the input symbols are processed (from right to left)
7. Halt the machine.

Note:

2"s complement computation

- Do not change the bits from the right towards left until the first „1“ has been processed.
- Perform complementation to the rest of the bits from right to left (after first 1 is processed)
- Example:

$$\begin{array}{c}
 01011010000 \\
 \downarrow \downarrow \downarrow \downarrow \downarrow \text{No change} \\
 101001 \\
 \Rightarrow 10100110000
 \end{array}$$



The Turing machine, M is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where,

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

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

$$\Gamma = \{0, 1, B\}$$

$\delta \Rightarrow$ Given by the above transition diagram

$$q_0 = \{q_0\}$$

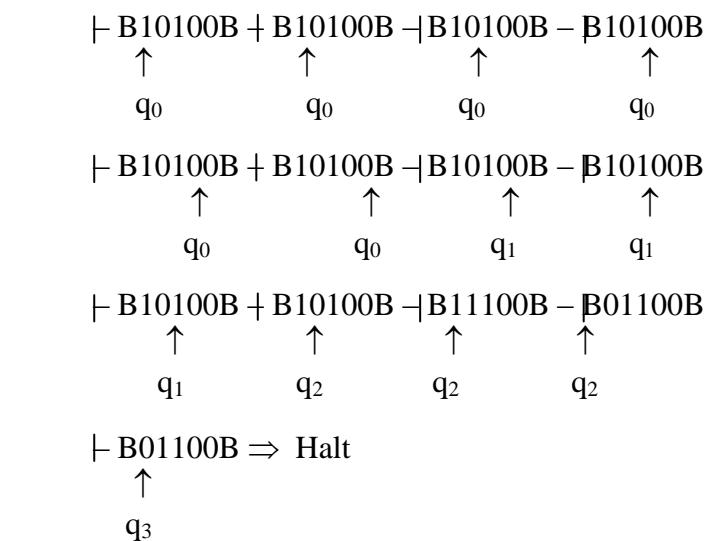
$$B = \{B\}$$

$$F = \{q_3\}$$

Transition table, δ

$Q \backslash \Gamma$	0	1	B
q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	(q_1, B, L)
q_1	$(q_1, 0, L)$	$(q_2, 1, L)$	(q_3, B, R)
q_2	$(q_2, 1, L)$	$(q_2, 0, L)$	(q_3, B, R)
* q_3	ϕ	ϕ	ϕ

Instantaneous description for $\omega = 10100$



2 Design a TM to find 1's complement of binary inputs.

Solution:

1's complement computation

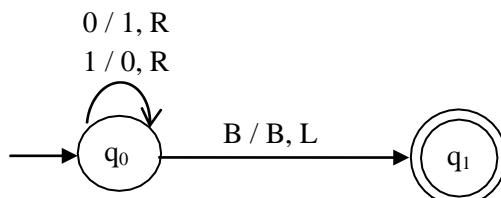
- Complement every bit in the string from right towards left.
- Eg: 0 1 0 1

$$\begin{array}{r} \downarrow \downarrow \downarrow \downarrow \\ 1 0 1 0 \end{array} \Rightarrow 1010$$

Algorithm

1. On reading the input, if the symbol is „0“, replace it by 1 and move right.
2. If the symbol read = „1“ replace it by „0“ and move right.
3. Halt the TM if all the string symbols are processed by step 1 and 2.

Turing Machine



The Turing machine, M is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where, $Q = \{q_0, q_1\}$

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

$$\Gamma = \{0, 1, B\}$$

$\delta \Rightarrow$ Given by the above Turing machine transition diagram

$$q_0 = \{q_0\}$$

$$B = \{B\}$$

$$F = \{q_1\}$$

Transition table, δ

$\Gamma \backslash Q$	0	1	B
$\rightarrow q_0$	$(q_0, 1, R)$	$(q_0, 0, R)$	(q_1, B, L)
* q_1	ϕ	ϕ	ϕ

Instantaneous description for $\omega = 0101$

$$\begin{array}{cccc} \vdash B0101B \dashv B1101B - B1001B - B1011B \\ \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \\ q_0 \quad q_0 \quad q_0 \quad q_0 \end{array}$$

$$\begin{array}{cc} \vdash B10100B \dashv B10100B \Rightarrow \text{Halt} \\ \uparrow \quad \uparrow \\ q_0 \quad q_1 \end{array}$$

3. Design a TM to compute addition of two unary numbers.

Solution:

The unary input number „n“ is represented with a symbol „0“ n – times.

Example: $4 \rightarrow 0000$

$$1 \rightarrow 0$$

$$5 \rightarrow 00000$$

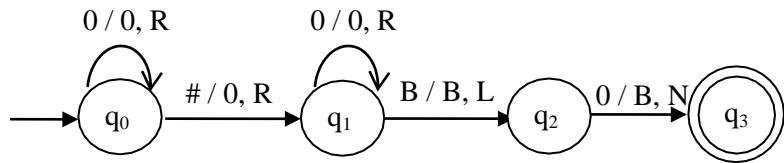
The separation symbol, „#“ (any other special character) shall be used to distinguish between two or more inputs.

Example: 5, 2 are the inputs represented by 00000 # 00.

Algorithm

1. Read the symbols of the first input with no replacements and move right.
2. When the symbol = „#“ replace it by „0“ and move right.
3. Traverse right side until the rightmost „0“ (left to „B“ – lastsymbol)
4. Replace the rightmost „0“ by „B“
5. Stop the machine.

Turing Machine



The Turing machine, M is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where,

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

$$\Sigma = \{0, \#\}$$

$$\Gamma = \{0, \#, B\}$$

$\delta \Rightarrow$ Given by the above transition diagram

$$q_0 = \{q_0\}$$

$$B = \{B\}$$

$$F = \{q_3\}$$

Transition table, δ

$\Gamma \backslash Q$	0	#	B
$\rightarrow q_0$	($q_0, 0, R$)	($q_0, 0, R$)	-
q_1	($q_1, 0, R$)	-	(q_2, B, R)
q_2	(q_3, B, N)	-	-
* q_3	ϕ	ϕ	ϕ

Instantaneous description for $\omega = 2, 3$

$$\vdash B00\#000B + B00\#000B \dashv B00\#000B$$

\uparrow \uparrow \uparrow
 q_0 q_0 q_0

$$\vdash B000000B + B000000B \dashv B000000B$$

\uparrow \uparrow \uparrow
 q_1 q_1 q_1

$$\vdash B000000B + B000000B \dashv B00000BB \Rightarrow \text{Halt}$$

\uparrow \uparrow \uparrow
 q_1 q_1 q_1

4 Design a TM to compute subtraction of two unary numbers given by

AU DEC 2003, DEC 2005

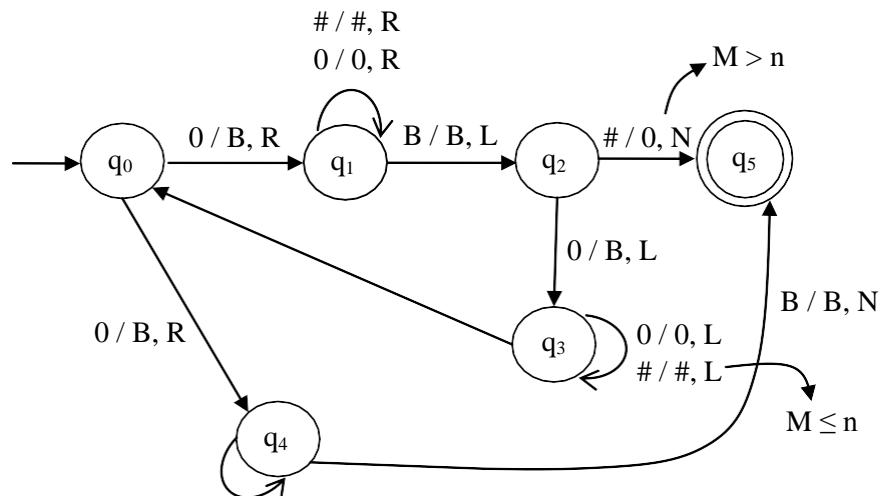
$$f(m, n) = \begin{cases} m - n, & \text{if } m > n \\ 0, & \text{otherwise} \end{cases}$$

Solution:

Algorithm

1. Replace the leftmost „0“ by B and move right.
 2. Replace the rightmost „0“ by B and move left.
 3. Perform steps 1 and 2, $(n - 1)$ times.
 4. Halt the machine since remaining „0“ by B and halt.

Turing Machine



The Turing machine, M is given by

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
where,

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

$$\Sigma = \{0, \#\}$$

$$\Gamma = \{0, \#, B\}$$

$$q_0 = \{q_0\}$$

$$B = \{B\}$$

$$F = \{q_5\}$$

δ is given by

$Q \setminus \Gamma$	0	#	B
$\rightarrow q_0$	(q_1, B, R)	(q_4, B, R)	-
q_1	($q_1, 0, R$)	($q_1, \#, R$)	(q_2, B, L)
q_2	(q_3, B, L)	($q_5, 0, N$)	-
q_3	($q_3, 0, L$)	($q_3, \#, L$)	(q_0, B, R)
q_4	(q_4, B, R)	-	(q_5, B, N)
* q_5	ϕ	ϕ	ϕ

Instantaneous description for $\omega = 000\#0 [3 - 1]$

$$\vdash B000\#0B \dashv BB00\#0B - BB00\#0B$$

\uparrow
 q_1 \uparrow
 q_1 \uparrow
 q_1

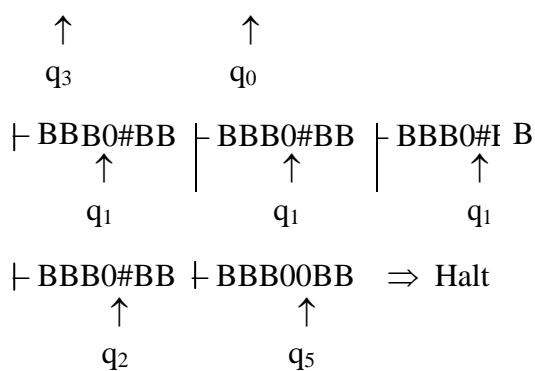
$$\vdash BB00\#0B \dashv BB00\#0B - BB00\#0B$$

\uparrow
 q_1 \uparrow
 q_1 \uparrow
 q_1

$$\vdash BB00\#0B \dashv BB00\#BB - BB00\#BB$$

\uparrow
 q_2 \uparrow
 q_3 \uparrow
 q_3

$$\vdash BB00\#BB \dashv BB00\#BB$$



5. Compute a TM that performs multiplication of two unary numbers.

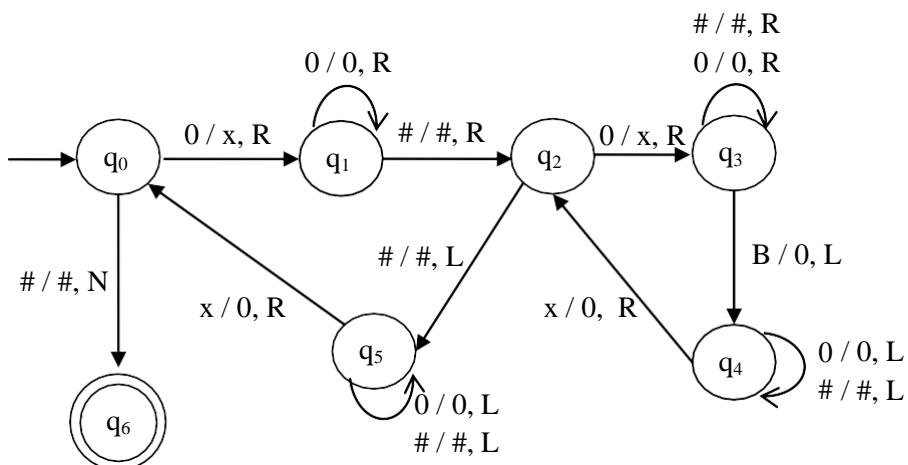
AU MAY 2005, DEC 2007

Solution:

Algorithm

1. Read the leftmost „0“ replace it by „x“ and move right to process the immediate symbol after „#“.
2. Replace the symbol „0“ by x and move right reach the first „B“ after „#“
3. Replace „B“ by „0“ and move left until the nearest „x“ is reached
4. Replace the „x“ by 0 and move right to process the next symbol of the multiplicand.
5. Perform steps 2, 3 and 4 until all the symbols of the multiplicand are processed.
6. Move left to replace the symbol of the multiplier, „x“ by „0“.
7. Perform steps 1 to 6 until all the symbols of the multiplier are processed.

Turing Machine



The turing machine, M is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where,

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$$

$$\Sigma = \{0, \#\}$$

$$\Gamma = \{0, \#, x, B\}$$

$$q_0 = \{q_0\}$$

$$B = \{B\}$$

$$F = \{q_6\}$$

$\delta \Rightarrow$ given by the above transition diagram.

Instantaneous description

Let $\omega = 2, 2 \Rightarrow 2 \times 2 = 4 \Rightarrow B00\#00\#BB$

$$\vdash B00\#00\#B \dashv Bx0\#00\#B \dashv Bx0\#00\#B$$

\uparrow \uparrow \uparrow
 q_0 q_1 q_1

$$\vdash Bx0\#00\#B \dashv Bx0\#x0\#B \dashv Bx0\#x0\#B$$

\uparrow \uparrow \uparrow
 q_2 q_3 q_3

$$\vdash Bx0\#x0\#B \dashv Bx0\#x0\#0B \dashv Bx0\#x0\#0B$$

\uparrow \uparrow \uparrow
 q_3 q_4 q_4

$$\vdash Bx0\#x0\#0B \dashv Bx0\#00\#0B \dashv Bx0\#0x\#0B$$

\uparrow \uparrow \uparrow
 q_4 q_2 q_3

$$\vdash Bx0\#0x\#0B \dashv Bx0\#0x\#0B \dashv Bx0\#0x\#00B$$

\uparrow \uparrow \uparrow
 q_3 q_3 q_4

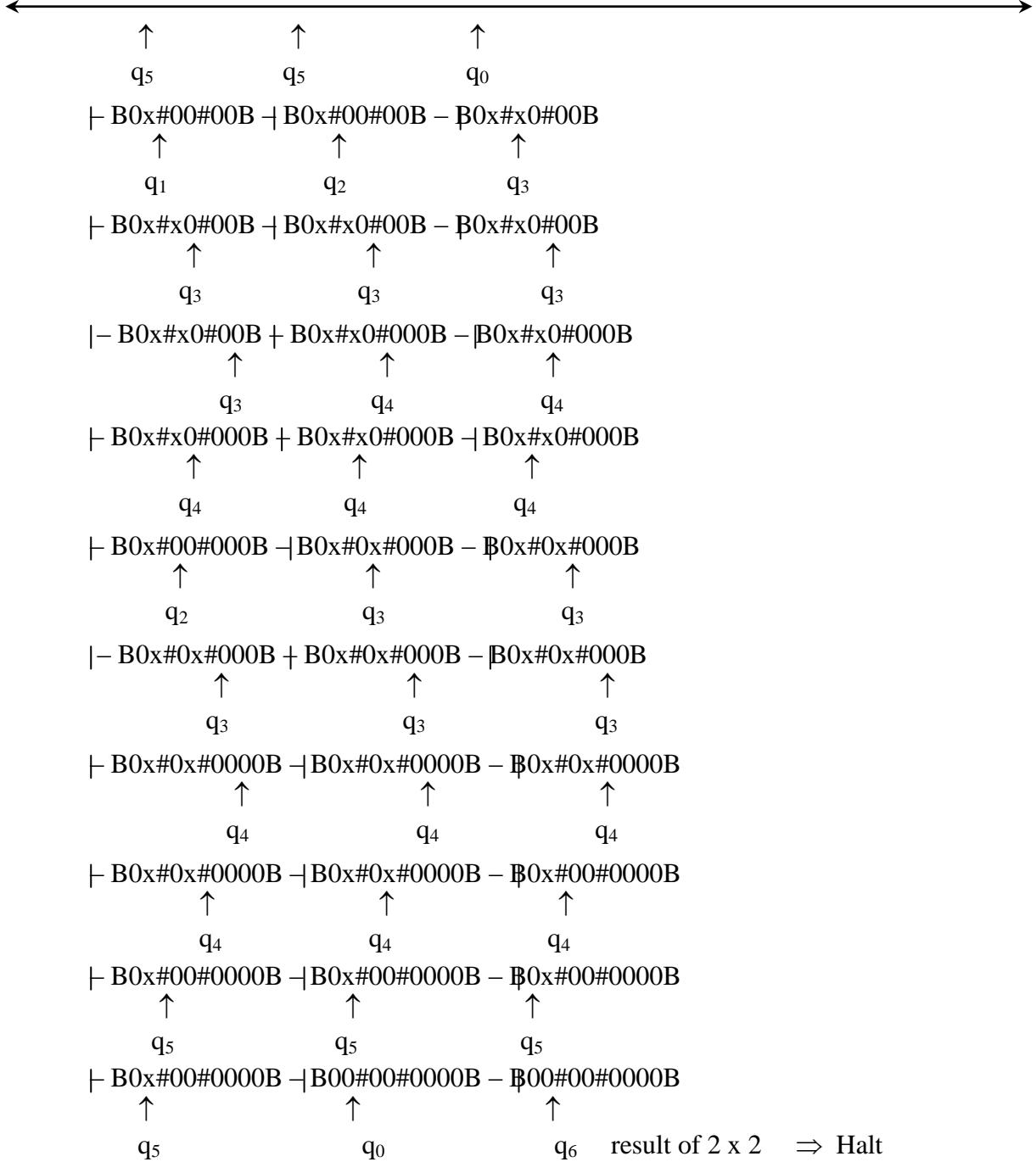
$$\vdash Bx0\#0x\#00B \dashv Bx0\#0x\#00B \dashv Bx0\#00\#00B$$

\uparrow \uparrow \uparrow
 q_4 q_4 q_2

$$\vdash Bx0\#00\#00B \dashv Bx0\#00\#00B \dashv Bx0\#00\#00B$$

\uparrow \uparrow \uparrow
 q_5 q_5 q_5

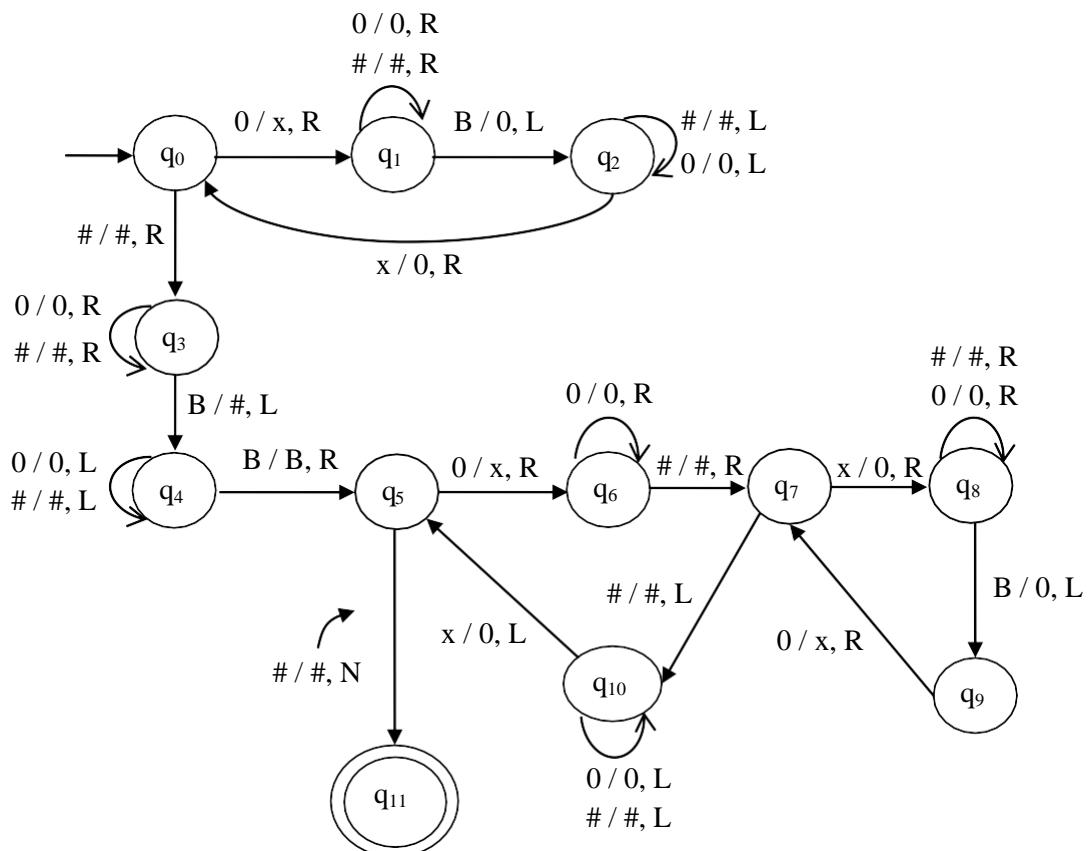
$$\vdash Bx0\#00\#00B \dashv Bx0\#00\#00B \dashv B00\#00\#00B$$



6. Design a TM to perform n^2 for $n \geq 0$.

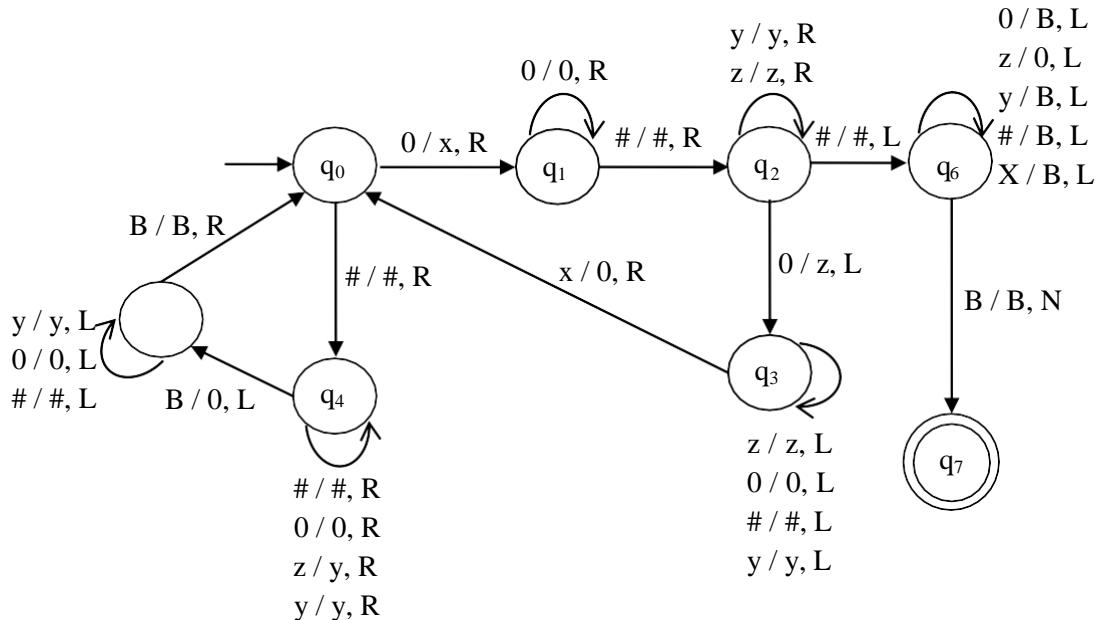
Solution:

- n^2 can be computed by multiplying „n“ by „n“ times.
- $n^2 = n \times n$
- Example: $3^2 = 3 \times 3 = 9$

Turing Machine

7. Design a TM to compute the quotient and remainder of division of two unary numbers.

Solution:



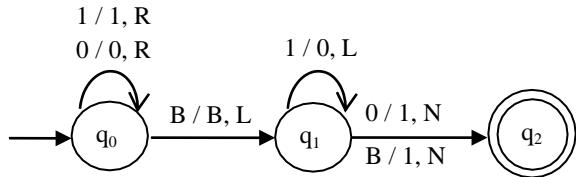
8. Design a TM that increments a binary number by 1.

Solution:

To perform incrementation

- Replace the trailing 1's to 0.
- Replace the first „0“ from the right end by 1.

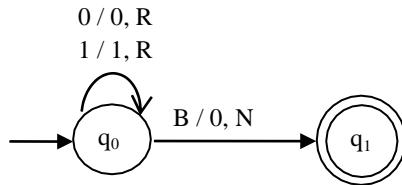
Turing Machine



9. The binary equivalent of a positive integer is stored in a tape. Write the transitions to multiply the integer by 2. AU Dec 2006

Solution:

Append „0“ to any binary number to get the solution of multiplication of the number by 2.



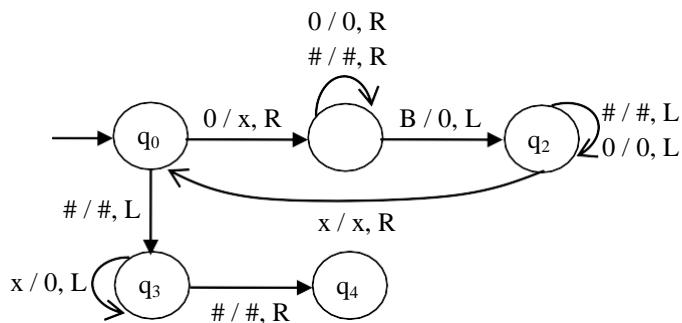
10. Design a TM that perform multiplication operation using “copy” subroutine. AU May 2004, June 2006

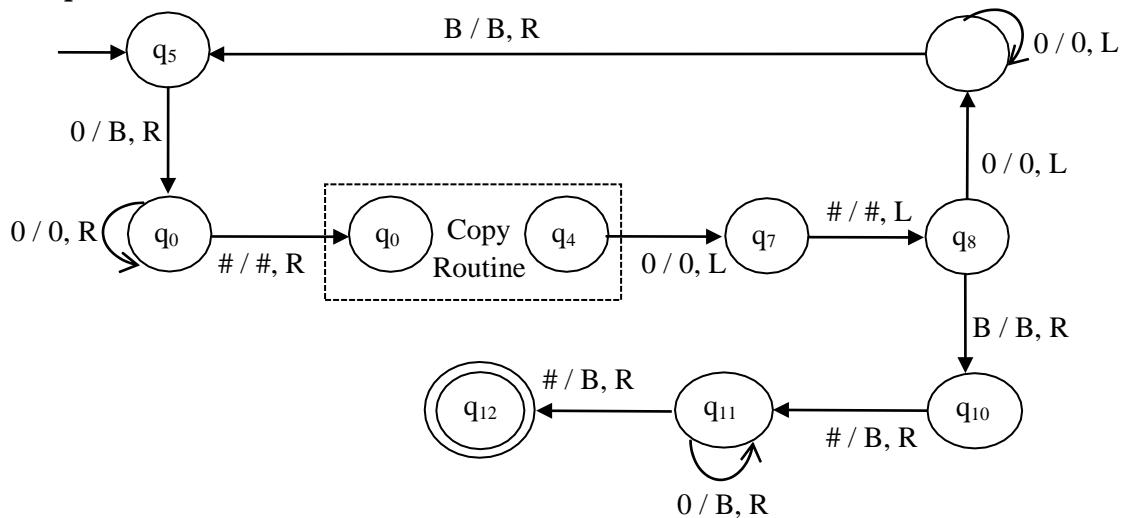
Solution:

- Let the inputs be 0^x and 0^y
- The inputs are stored as $B0^x \# 0^y \# BB$.

Turing Machine

Subroutine for copy operation



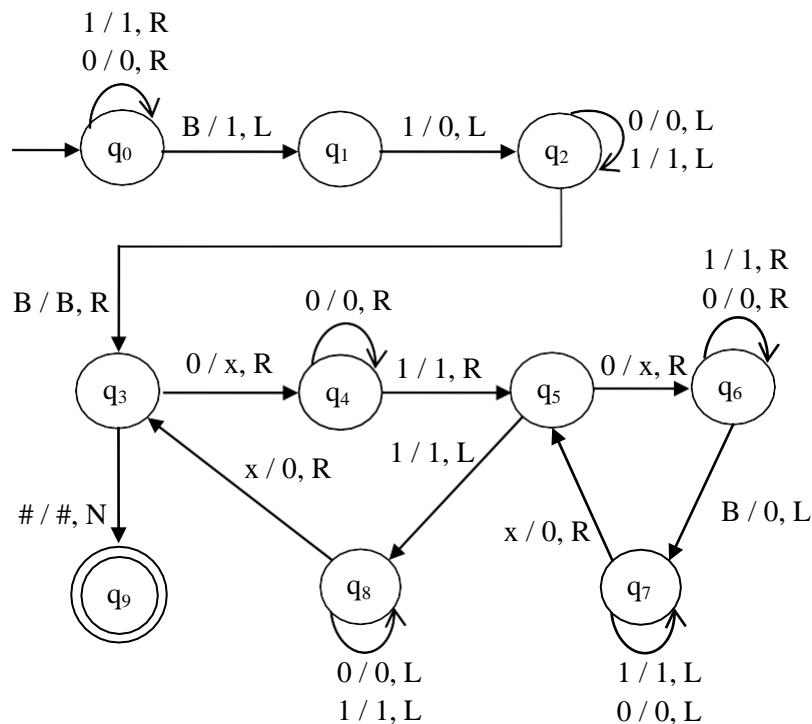
TM for multiplication

11. Design a TM for $f(x, y) = x * y + x$, where x and y are stored in the tape of the form, $0^x | 0^y$.
AU Dec 2008

Solution:

$$\begin{aligned}
 f(x, y) &= x * y + x \\
 &= x(y + 1) \\
 &= \text{Increment } y \text{ value by 1} \\
 &= \text{Multiply } x \text{ with } (y + 1)
 \end{aligned}$$

Turing machine:



TECHNIQUES FOR TURING MACHINE CONSTRUCTION

- Every Turing machine is defined by its transition function, $\delta: Q \times \Gamma \times \{L, R, S\}$ for any (q, ω) .
- The transition function is a partial function since it is not defined for all (q, ω) .
- The transition is based on the current state and input symbol.
- The process results in a transition/resulting state, tape symbol replacement by right / left / stop (No change).
- The higher level of formal description of a Turing machine is called high level description / implementation description.
- There are some high-level conceptual tools to construct Turing machine easier.
- They are,
 - Turing machine with stationary head
 - Storage in the state
 - Multiple track Turing machine
 - Subroutines
 - Two-way infinite tape.

Turing Machine with Stationary Head

- Every Turing machine has a finite state machine that controls the input tape using its head (movable).
- The head is capable of moving left (L) right (R) or remain stationary (N) with no movement.
- For any transition can be defined as, $\delta(q, a) = (q', b, M)$ where M can be L / R / N.
- Thus the head moves left / right / remain unchanged after reading the input symbol, „a“.
- Consider the transition,

$$\delta(q, a) = (q', b, N)$$

- The Turing machine after reading „a“, changes the state from q to q' and replace „a“ by „b“.
- The head remains in the same cell since the movement is given as „N“.
- The above transition can also be simulated by the below given transitions.

$$\delta(q, a) = (q'', b, R)$$

$$\delta(q'', B) = (q', b, L), \text{ where } B \in \Sigma \cup \Gamma$$

- Thus the turing machine is modeled with a transition given by

$$\delta(q, a) = (q'b, M)$$

where,

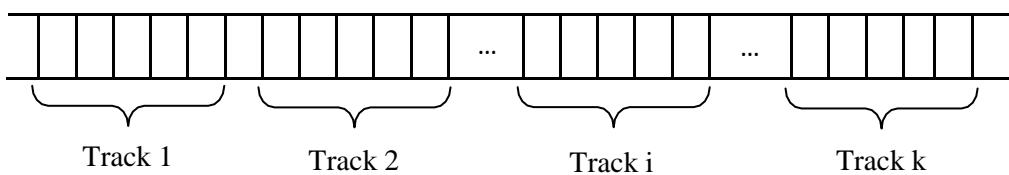
$$M = L / R / N$$

Storage in the State

- The states of turing machine are to remember/store a symbol. This is done by the finite control.
- The pair (q, a) defines the state and the tape symbol being stored.

Multiple tracks Turing Machine

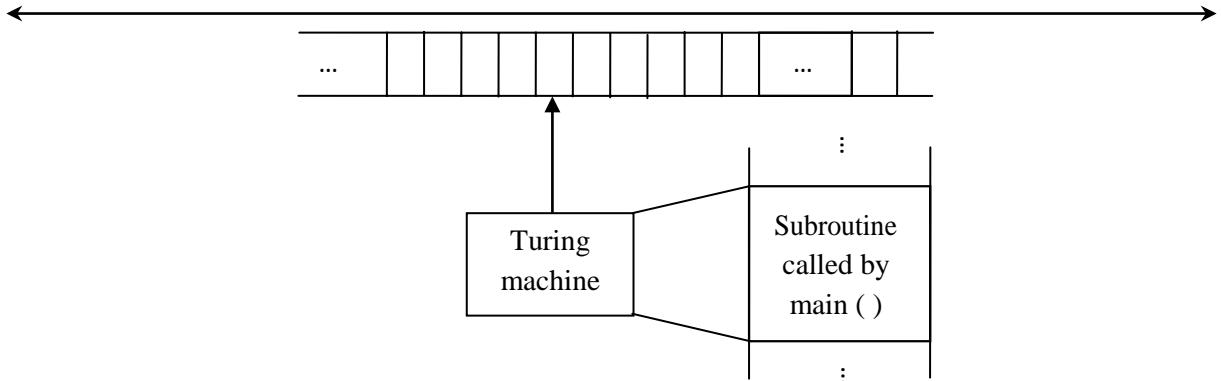
- The input tape of a Turing machine is an infinite single tape divided into individual cells.
- A multiple track Turing machine is a single tape divided into multiple tracks.
- This is also called as k-dimension tape, where k is the number of „tracks“.
- Standard Turing machine contains the tape symbol as the elements of Γ .
- Multitrack Turing machine contain multiple tracks (k) each with Γ set of elements leading to Γ^k symbols on the tape.
- The transitions are same in both the types.



Subroutine

- Subroutines are sub functions that can be used to execute repeated tasks for any number of times depending on the application.
- In such case, the Turing machine has to be designed that handles subroutines.

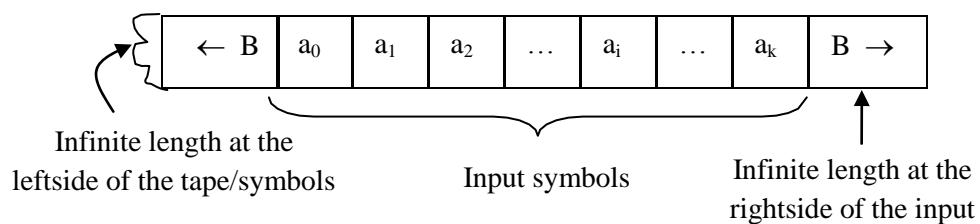
4.8 Theory of Computation



- The Turing machine is designed to process the subroutine of a program.
- The subroutine has two states
 - Initial state
 - Return state
- When the main function is executed, the subroutine is called.
- The TM reaches the initial state and follows a series of executions using the transition rules of the subroutine.
- After processing, the TM reaches the return state that returns back to the main function after a temporary halt.

Two – way infinite tape TM

- A turing machine, $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is said to be a two-way infinite tape TM if the input tape is infinite to the left and right.



- It can be viewed as a finite sequence of input symbols with infinite sequence of blank symbols to the left and right of the input.
- As with the standard TM, there is a fixed left end, the two - way infinite tape TM has no left end. Hence it can move as far as possible towards left as well as right.
- This is no “fall off” of TM towards left.

Multi-head and multi-tape Turing machine

- There are several variants of standard Turing machine.
- Standard TM has a finite state machine controlling a single input tape.
- The transition function of which is given as,

$$\delta (q, a) = (q', b, M)$$

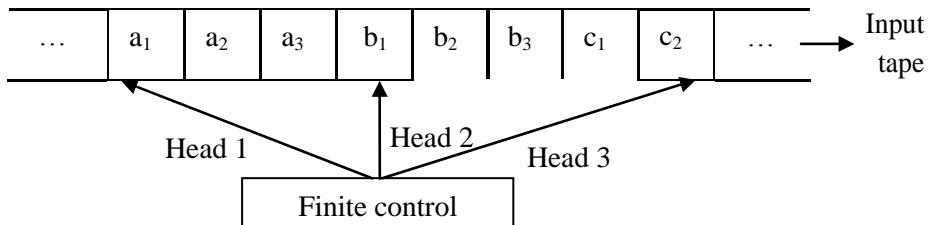
where,

$$M = R \text{ or } L \text{ or } N$$

- The variants of TM include,
 - Multi-head turing machine
 - Multi-tape turing machine

Multi-head Turing machine

- A Turing machine with a single tape can have any number of heads to provide simultaneous accesses over the tape.



- The finite control possesses two or more tape heads to access the input tape for performing multiple reads/writes in a simultaneous and independent manner.
- The transition behavior of a 2 headed tape is given as

$$\delta (q', H_1(a), H_2(a)) = (q'', H_1(b), M_1), H_2(b), M_2))$$

Where

$q', q'' \rightarrow$ states of Q

$H_1(a) \rightarrow$ symbol to be processed (read) by H_1

$H_2(a) \rightarrow$ symbol to be processed (read) by H_2

$H_1(b) \rightarrow$ symbol to be replaced by H_1

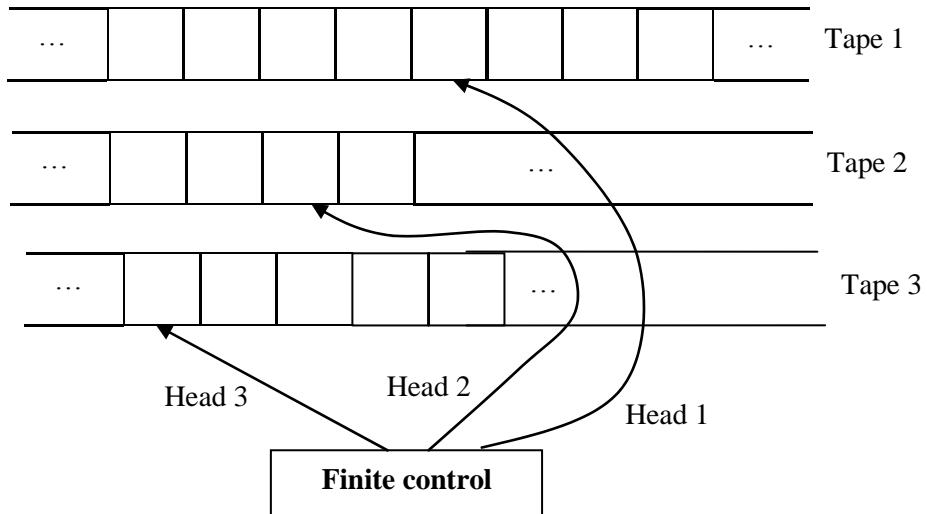
$H_2(b) \rightarrow$ symbol to be replaced by H_2

$M_1 \rightarrow$ movement of H_1 (L / R / N)

$M_2 \rightarrow$ movement of H_2 (L / R / N)

Multi-tape Turing machine

- A multitape TM is finite control with more than one tape for reading/writing symbols, storing states etc.
- Each tape has „n“ number of individual cells.
- The first tape is mostly used to store the input symbols of the string, ω .
- The second and third tapes are normally used to store the states and state-changes.



The transition rules of a two tape TM is given as,

$$\delta (q, a_1, a_2) = (q', H_1(b), M_1), H_2(b), M_2)$$

where,

$q \rightarrow$ current state to be processed

$q' \rightarrow$ next state to be reached

$a_1 \rightarrow$ symbol read by tape 1

$a_2 \rightarrow$ symbol read by tape 2

$H_1(b) \rightarrow$ symbol to be written by tape 1

$H_2(b) \rightarrow$ symbol to be written by tape 2

$M_1 \rightarrow$ Movement by tape 1 (R / L / N)

$M_2 \rightarrow$ Movement by tape 2 (R / L / N)

- Each move in a multitape TM depends on the current state and the symbol scanned by the tape heads.

- Thus, on each move,
 - State change
 - Replace new symbols on each cells being processed on the tapes.
 - Tape heads take right/left movement or remain unchanged.

THEOREM: Equivalent of multitape and single tape TM

If a language, L is accepted by a multitape TM, then it is accepted by a single tape TM.

Proof

Let M_1 , be a multitape turing machine with k-tapes that accepts the language, L

Let M_2 be a single-tape, multi – track turing machine simulated by M_1 .

The number of tracks on M_2 is double the number of tapes of M_1 [2 x k tracks]

Thus M_2 consists of two tracks for every tape in M_1 .

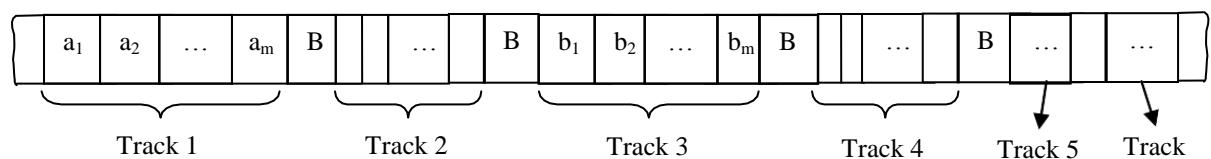
At $k = 1$, M_2 has 2 tracks

- The first track records the contents of M_1
- The second track (initially kept as blank, B) holds the symbol processed by M_1 .

This set-up can be further extended to k-tape TM as,

Head 1		X				
Tape 1	a_1	a_2	a_m
Head 2				X		
Tape 2	b_1	b_2	b_m
:						:
Head k	X					
Tape k	ℓ_1	ℓ_2	ℓ_m

M_1 k : k – tapes TM



M_2 : 2k tracks TM

To prove that $L(M_1) = L(M_2)$, let us consider $k = 2$ (lower most possible n-tape TM) and prove the above

Further k shall be extended to get the general case.

Let

$$M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, B, F_1)$$

$$M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, B, F_2)$$

such that

$$\Gamma_1 \subseteq \Gamma_2 \text{ and } \Sigma \subseteq \Gamma_1 \cup \Gamma_2$$

The tape symbol of m_2 (M_2) includes,

$$\Gamma = \Gamma_1 \cup B \Rightarrow \text{ordinary symbols}$$

$$\text{Elements of } \Gamma \times \Gamma$$

NON DETERMINISTIC TURING MACHINE

- Non determinism is a powerful feature of Turing machine.
- These NTM machines are easy to design and are equivalent to deterministic TM.
- A NTM accepts a string, ω if there exists at least one sequence of moves from the initial state to final state.

Definition

A NTM is defined as

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where,

$Q \rightarrow$ Set of states including initial, having rejecting states.

$\Sigma \rightarrow$ Finite set of input alphabets

$\Gamma \rightarrow$ Finite set of tape symbols

$\delta \rightarrow$ Transition function defined by

$$\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R, N\}) \text{ where } P \rightarrow \text{power set}$$

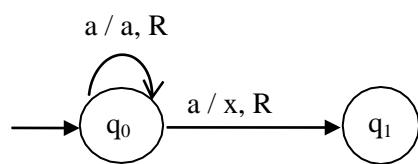
$q_0 \rightarrow$ initial state

$B \rightarrow$ blank symbol

$F \rightarrow$ set of final states ($F \subseteq Q$)

The transition function δ takes on the states tape symbols and head movement.

Example



The above transition takes on two paths for the same input, a

The transition of „a“ at q_0 is defined as,

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

UNIT - 5

UNDECIDABILITY

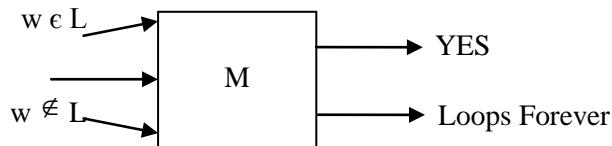
RECURSIVE AND RECURSIVELY ENUMERABLE LANGUAGES

Recursively Enumerable Language

A language $L \subseteq \Sigma^*$ is recursively enumerable if there exist a Turing machine, M that accepts every string, $w \in L$ and does not accept strings that are not in L.

If the input string is accepted, M halts with the answer, "YES".

If the string is not an element of L, then M may not halt and enters into infinite loop.

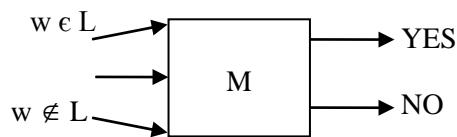


The language, L is Turing Acceptable.

Recursive Language

A language is said to be recursive if there exists of Turing machine, M that accepts every string, $w \in L$ and rejects those strings that are not in L.

If the input is accepted, M halts with the answer, "YES"



If the Turing machine doesn't accept the string.

If $w \notin L$, then M halts with answer, "NO". This is also called as Turing Decidable language.

PROPERTIES OF RECURSIVE AND RE LANGUAGES

1. The Union of two recursively enumerable languages is recursively enumerable.

2. The language L and its complement \bar{L} are recursively enumerable, then L and \bar{L} are recursive.
3. The complement of a recursive language is recursive.
4. The union of two recursive language is recursive.
5. The intersection of two recursive language is recursive.
6. The intersection of two recursively enumerable language is recursively enumerable

Proofs on the Properties

Property-1

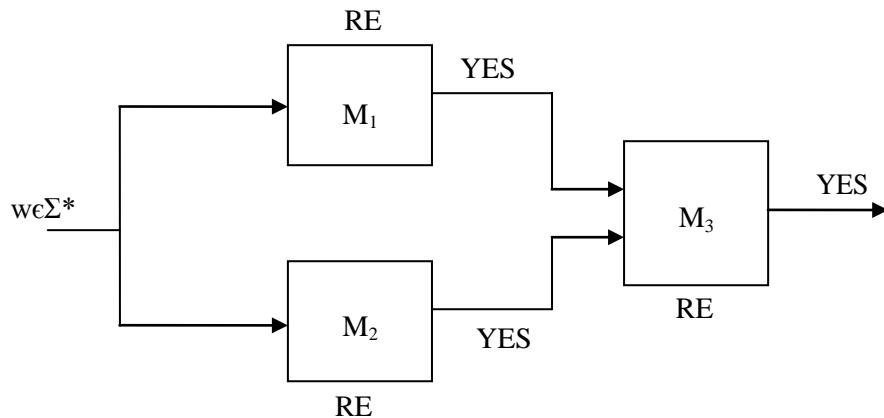
The union of two recursively enumerable languages is recursively enumerable.

Proof:

Let L_1 and L_2 be two recursively enumerable languages accepted by the Turing machines M_1 and M_2 .

If a string $w \in L_1$ then M_1 returns “YES”, accepting the input string: Else loops forever. Similarly if a string $w \in L_2$ then M_2 returns “YES”, else loops forever.

The Turing machine M_3 that performs the union of L_1 and L_2 is given as



Here the output of M_1 and M_2 are written on the input tape of M_3 . The turning machine, M_3 returns “YES”, if at least one of the outputs of M_1 and M_2 is “YES”. The M_3 decides on $L_1 \cup L_2$ that halts with the answer, “YES” if $w \in L_1$ or $w \in L_2$. Else M_3 loops forever if both M_1 and M_2 loop forever.

Hence the union of two recursively enumerable languages is also recursively enumerable.

Property - 2

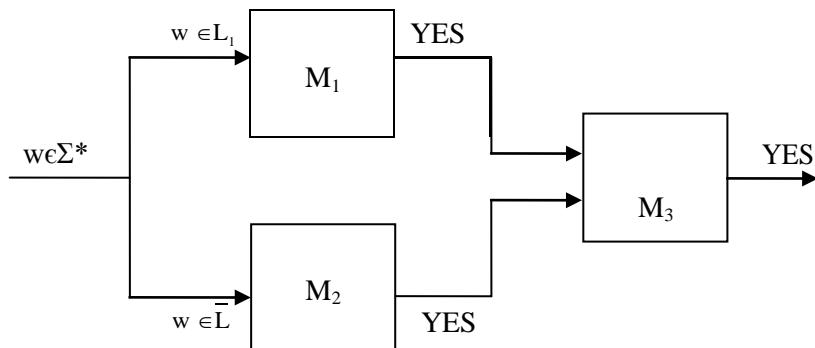
A language is recursive if and only if both it and its complement are recursively enumerable.

Proof

Let L and \bar{L} be two recursively enumerable languages accepted by the Turing machines M_1 and M_2 . If a string, $w \in L$, it is accepted by M_1 and M_1 halts with answer "YES". Else M_1 enters into infinite loop.

If a string, $w \in \bar{L}$ [$w \notin L$], then it is accepted by M_2 and M_2 halts with answer "YES". Otherwise M_2 loops forever.

The Turing machine, M_3 that simulates M_1 and M_2 simultaneously is given as



From the above design of TM, if $w \in L$, if $w \in \bar{L}$, then M_1 accepts w and halts with "YES".

If $w \notin L$, then M_2 accepts w [$w \in \bar{L}$] and halts with "YES".

Since M_1 and M_2 are accepting the complements of each other, one of them is guaranteed to halt for every input, $w \in \Sigma^*$.

Hence M_3 is a Turing machine that halts for all strings.

Thus if the language and its complement are recursively enumerable, then they are recursive.

Property - 3

The complement of a recursive language is recursive.

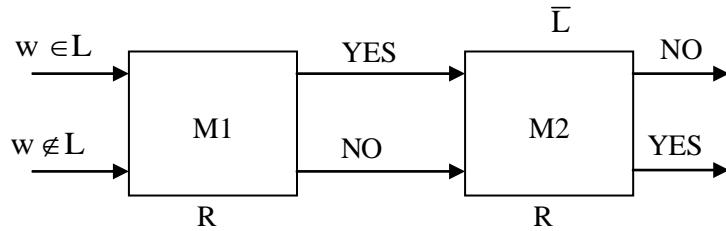
Proof

Let L be a recursive language accepted by the turning machine, M_1 .

Let \bar{L} be a recursive language accepted by the Turing machine M_2 .

5.4 Theory of Computation

The construction of M_1 and M_2 are given as,



Let $w \in L$, then M_1 accepts w and halts with “YES”.

M_1 rejects w if $w \notin L$ and halts with “NO”

M_2 is activated once M_1 halts.

M_2 works on \bar{L} and hence if M_1 returns “YES”, M_2 halts with “NO”.

If M_1 returns “NO”, then M_2 halts with “YES”

Thus for all w , where $w \in L$ or $w \notin L$, M_2 halts with either “YES” or “NO”

Hence the complement of a recursive language is also recursive.

Property – 4

The union of two recursive language is recursive.

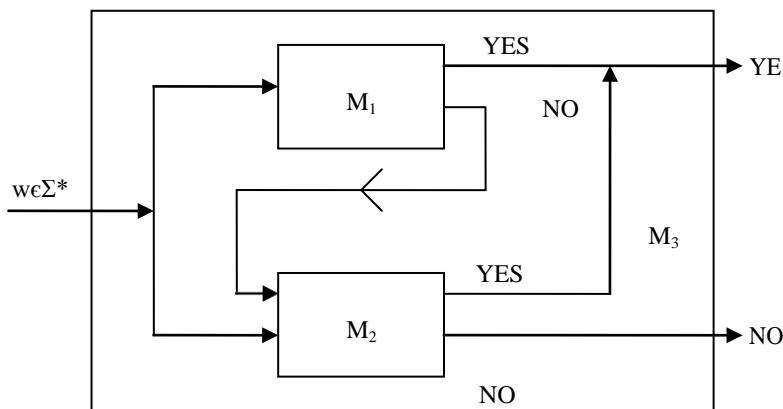
Proof:-

Let L_1 and L_2 be two recursive languages that are accepted by the Turing machines M_1 and M_2 , given by

$$L(M_1) = L_1$$

$$L(M_2) = L_2$$

Let M_3 be the Turing machine constructed by the union of M_1 and M_2 . M_3 is constructed as follows.



The Turing machine M_3 first simulates M_1 with the input string, w .

If $w \in L_1$, then M_1 accepts and thus M_3 also accepts since $L(M_3) = L(M_1) \cup L(M_2)$.

If M_1 rejects string $[w \notin L_1]$, then M_3 simulates M_2 . M_3 halts with “YES” if M_2 accepts „ w “, else returns “NO”.

Hence M_3 , M_2 , M_1 halt with either YES or NO on all possible inputs.

Thus the union of two recursive languages is also recursive.

Property – 5

The intersection of two recursive language is recursive.

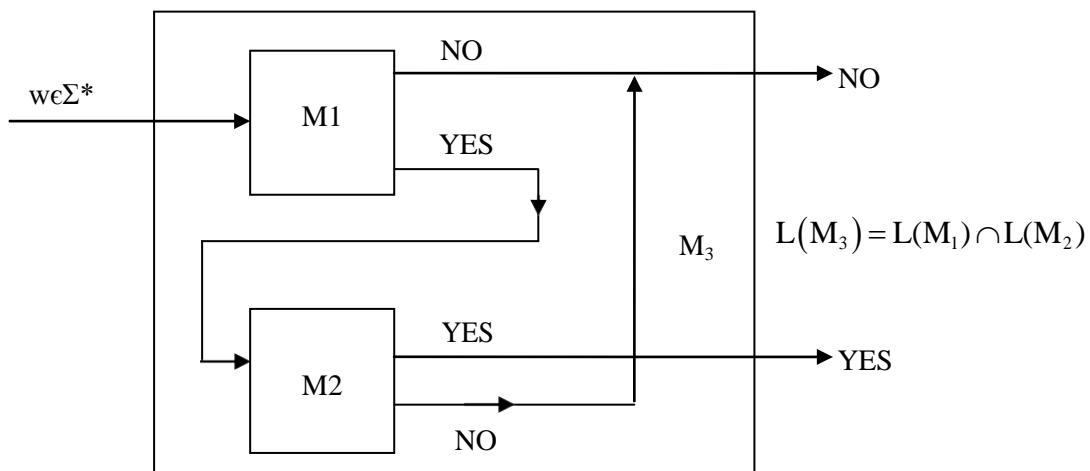
Proof:-

Let L_1 and L_2 be two recursive languages accepted by M_1 and M_2 where

$$L(M_1) = L_1$$

$$L(M_2) = L_2$$

Let M_3 be the Turing machine that is constructed by the intersection of M_1 and M_2 , M_3 is constructed as follows.



The Turing machine M_3 simulates M_1 with the input string, w .

If $w \notin L_1$, then M_1 halts along with M_3 with answer “NO”, since $L(M_3) = L(M_1) \cap L(M_2)$. If then M_1 accepts with the answer “YES” and M_3 simulates M_2 .

If M_2 accepts the string, then the answer of M_2 and M_3 are “YES” and halts. Else, M_2 and M_3 halts with answer “NO”.

Thus, the intersection of two recursive languages is recursive.

Property – 6

Intersection of two recursively enumerable languages is recursively enumerable.

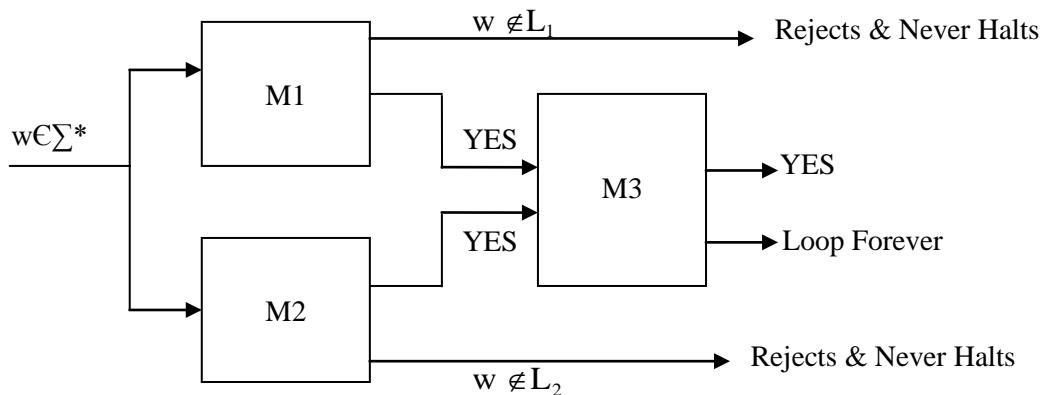
Proof:-

Let L_1 and L_2 be two recursively enumerable languages accepted by the Turing machine M_1 and M_2 .

If a string $w \in L_1$ then M_1 returns “YES” accepting the input. Else will not halt after rejecting $w \notin L_1$.

Similarly if a string, $w \in L_2$, then M_2 returns “YES” else rejects „ w “ and loop forever.

The Turing machine, $M_3 = M_1 \cap M_2$ is given as



Here the output of M_1 and M_2 are written the input tape of M_3 . The machine, M_3 returns “YES” if both the outputs of M_1 and M_2 is “YES”.

If at least one of M_1 or M_2 is NO it rejects „ w “ and never halts.

Thus M_3 decides on $L_1 \cap L_2$ that halts if and only if $w \in L_1$ and $w \in L_2$. Else M_3 loops forever along with M_1 or M_2 or both

Hence the intersection of two recursively enumerable languages is recursively enumerable.

THE HALTING PROBLEM

- The halting problem is the problem of finding if the program/machine halts or loop forever.
- The halting problem is un-decidable over Turing machines.

Description

- Consider the Turing machine, M and a given string ω , the problem is to determine whether M halts by either accepting or rejecting ω , or run forever.
- Example**

```
while (1)
{
    printf("Halting problem");
}
```

- The above code goes to an infinite loop since the argument of while loop is true forever.
- Thus it doesn't halt.
- Hence Turing problem is the example for undecidability.
- This concept of solving the halting problem being proved as undecidable was done by Turing in 1936.
- The undecidability can be proved by reduction technique.

Representation of the halting set

The halting set is represented as,

$$h(M, \omega) = \begin{cases} 1 & \text{if } M \text{ halts on input } \omega \\ 0 & \text{otherwise} \end{cases}$$

where,

$M \rightarrow$ Turing machine

$\omega \rightarrow$ Input string

Theorem

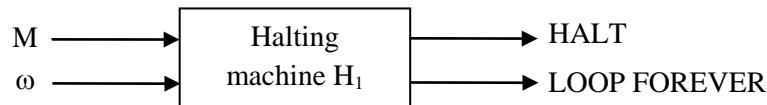
Halting problem of Turing machine is unsolvable / undecidable.

Proof

The theorem is proved by the method of proof by contradiction.

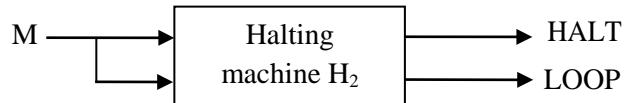
Let us assume that Turing machine is solvable / decidable.

Construction of H_1



- Consider, a string describing M and input string, ω for M.
- Let H_1 generates “halt” if H_1 determines that the turing machine, M stops after accepting the input, ω .
- Otherwise H_1 loops forever when, M doesn’t stop on processing ω .

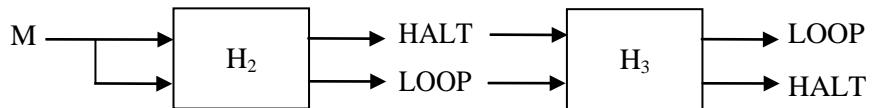
Construction of H_2



H_2 is constructed with both the inputs being M.

H_2 determines M and halts if M halts otherwise loops forever.

Construction of H_3

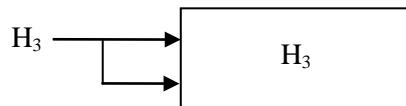


Let H_3 be constructed from the outputs of H_2 .

If the outputs of H_2 are HALT, then H_3 loops forever.

Else, if the output of H_2 is loop forever, then H_3 halts.

Thus H_3 acts contractor to that of H_2 .



- Let the output of H_3 be given as input to itself.
- If the input is loop forever, then H_3 acts contradictory to it, hence halts.
- And if the input is halt, then H_3 loops by the construction.
- Since the result is incorrect in both the cases, H_3 does not exist.
- Thus H_2 does not exist because of H_3 .
- Similarly H_1 does not exist, because of H_2 .

Thus halting problem is undecidable.

PARTIAL SOLVABILITY

Problem types

There are basically three types of problems namely

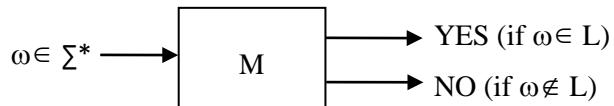
- Decidable / solvable / recursive
- Undecidable / unsolvable
- Semi decidable / partial solvable / recursively enumerable

Decidable / solvable problems

A problem, P is said to be decidable if there exists a turing machine, TM that decides P.

Thus P is said to be recursive.

Consider a Turing machine, M that halts with either „yes“ or „no“ after computing the input.



The machine finally terminates after processing

It is given by the function,

$$F_p(\omega) = \begin{cases} 1 & \text{if } p(\omega) \\ 0 & \text{if } \neg p(\omega) \end{cases}$$

The machine that applies $F_p(\omega)$ is said to be turing computable.

Undecidable problem

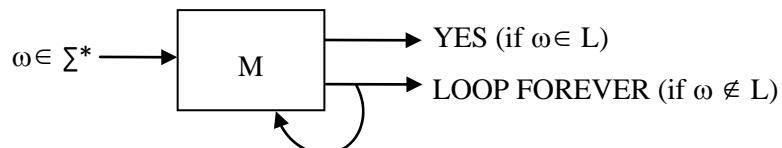
A problem, P is said to be undecidable if there is a Turing machine, TM that doesn't decides P.

Semi decidable / partial solvable / recursively enumerable

A problem, P is said to be semi decidable, if P is recursively enumerate.

A problem is RE if M terminates with „YES“ if it accepts $\omega \in L$; and doesn't halt if $\omega \notin L$.

Then the problem is said to be partial solvable / Turing acceptable.



5.10 Theory of Computation

Partial solvability of a machine is defined as,

$$F(\omega) = \begin{cases} 1 & \text{if } p(\omega) \\ p & \text{undefined if } \neg p(\omega) \end{cases}$$

Enumerating a language

Consider a k-tape turing machine. Then the machine M enumerates the language L (such that $L \subseteq \Sigma^*$) if

- The tape head never moves to the left on the first tape.
- No blank symbol (B) on the first tape is erased or modified.
- For all $\omega \in L$, where there exists a transition rule, δ_i on tape 1 with contents

$$\omega_1 \# \omega_2 \# \omega_3 \# \dots \# \omega_n \# \omega \# \quad (\text{for } n \geq 0)$$

Where $\omega_1, \omega_2, \omega_3, \dots, \omega_n, \omega$ are distinct elements on L.

If L is finite, then nothing is printed after the # of the left symbol

That is,

- If L is a finite language then the TM, M either
 - Halts normally after all the elements appear on the first tape (elements are processed)
or
 - Continue to process and make moves and state changes without scanning/printing other string on the first tape.

If the language, L is finite, the Turing machine runs forever.

Theorem

A language $L \subseteq \Sigma^*$ is recursively enumerable if and only if L can be enumerated by some TM.

Proof

Let M_1 be a Turing machine that enumerates L.

And let M_2 accept L. M_2 can be constructed as a k-tape Turing machine [$k(M_2) > k(M_1)$].

M_2 simulates M_1 and M_1 pauses whenever M_2 scans the „#“ symbol.

M_2 compares its input symbols to that of the symbols before „#“ while, M_1 is in pause.

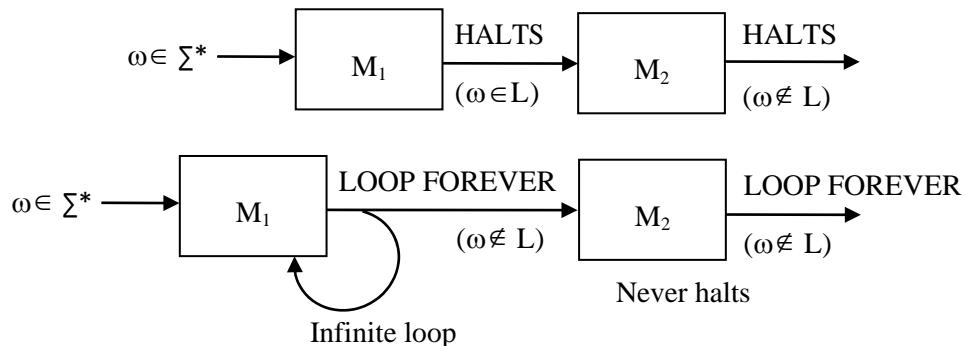
If the comparison finds a match of the string, M_2 accepts L .

Here M_2 is a semi acceptor TM for L

- Scans the input string, ω
- Runs the transition rules of M_1
- If M_1 outputs ω , then ω is accepted and M_1 halts

If $\omega \in L$, M_1 will output ω and M_2 will eventually accept „ ω “ and halts.

If $\omega \notin L$, then M_1 will never provide an output ω and so M_2 will never halt.



Thus M_2 is partially solvable / Turing acceptable for L .

POST CORRESPONDENCE PROBLEM

Post correspondence problem, known as PCP is an unsolvable combinatorial problem. This Undecidable problem was formulated by Emil Post in 1946.

Since PCP is simpler than halting problem, this is used in proofs of undecidability / un-solvable problems.

PCP definition

The input instance of the problem has two lists of non-null strings $\{\alpha_1, \alpha_2, \dots, \alpha_n\}$ and $\{\beta_1, \beta_2, \dots, \beta_n\}$ over an alphabets, Σ . [Σ having at most two symbols].

The task is to find a sequence of integers i_1, i_2, \dots, i_k [i_k for $1 \leq k \leq N$] such that

$$\alpha_{i_1}, \alpha_{i_2}, \dots, \alpha_{i_k} = \beta_{i_1}, \beta_{i_2}, \dots, \beta_{i_k}$$

The PCP is a decision problem that whether the above mentioned sequence exists or not.

EXAMPLE

- For $\Sigma = \{a, b\}$ with $A = \{a, aba^3, ab\}$ and $B = \{a^3, ab, b\}$, Does the PCP with A and B have a solution?

Solution:

The sequence obtained from A and B = (2, 1, 1, 3) as,

A ₂	A ₁	A ₁	A ₃
aba ³	a	a	ab
B ₂	B ₁	B ₁	B ₃
ab	a ³	a ³	b

$$\text{Thus } A_2A_1A_1A_3 = B_2B_1B_1B_3 = aba^3a^3b = aba^6b$$

The PCP given has a solution (2,1,1,3) with the two lists of elements.

- Let $\Sigma = \{0, 1\}$. Let A and B be the lists of three strings defined as

	A	B
I	w _i	x _i
1	1	111
2	10111	10
3	10	0

Solution:

Consider the sequence (2, 1, 1, 3)

$$A_2A_1A_1A_3 \Rightarrow w_2w_1w_1w_3 = 10111110$$

$$B_2B_1B_1B_3 \Rightarrow X_2X_1X_1X_3 = 10111110$$

Thus the PCP has (2, 1, 1, 3) sequences as solution

RICE'S THEOREM

- Rice's theorem is named after the inventor Henry Gordon Rice.
- This theorem is also called as Rice- Myhill – Shapiro theorem.

Theorem

If R is a property of languages that are satisfied by some but not all recursively enumerable languages, then the decision problem,

P_R : Given a TM, M, does $L(M)$ have property, R?
is unsolvable.

Explanation

Consider a language L such that

$$L = \{M \mid L(M) \text{ has some property } P\}$$

where,

P is non-trivial of TM, M

Then the language L is undecidable.

That is, every non-trivial property of the language of Turing machines is Undecidable

Proof

Consider a Turing machine, that recognizes empty language, ϕ that does not have the property, P. If ϕ has the property, P, then complement of P is considered

Thus the language is Undecidable due to complementation.

To arrive at a contradiction, assume that P is decidable

Then there is a Turing machine B that recognizes and halts on processing the descriptions of B that satisfy P

Using the Turing machine, M another Turing machine, A is constructed that accepts the language $\{(M, w) \mid M\}$ is the Turing machine that accepts w.

As M is Undecidable, then B cannot exist and thus the problem P must be Undecidable
Let MP be a Turing machine that satisfies P [P → non trivial] A is computed as

- (i) On the input x, M run on the string, w until it accepts. If it doesn't accept, C(M, w) will run forever.
- (ii) Run MP on x A accepts if and only MP accepts.

Here C(M, w) accepts the same language as Mp if M accepts w. C(M, w) accepts empty language, ϕ if M does not accept w.

If M accepts w, C(M, w) has the property, P, otherwise, C(M, w) is passed to B

If B accepts, C(M, w) accept the input (M, w) If B rejects, C(M, w) reject

UNIVERSAL TURING MACHINE

Motive of UTM

A single Turing machine has a capability of performing a function such as addition, multiplication etc.

For computing another function, other appropriate Turing machine is used. To do so, the machine has to be re-written accordingly.

Hence Turing proposed “Stored Program Computer” concept in 1936 that executes the program/instructions using the inputs, stored in the memory.

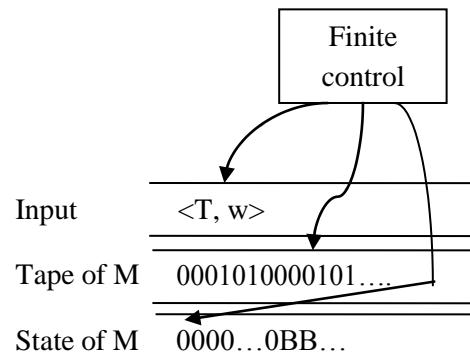
The instructions and inputs are stored on one or more tapes.

Concept of UTM

The universal Turing machine, T_u takes over the program and the input set to process the program.

The program and the inputs are encoded and stored on different tapes of a multi-tape Turing machine.

The T_u thus takes up T, w where T is the special purpose Turing machine that passes the program in the form of binary string, w is the data set that is to be processed by T .



Encoding T and w

The encoding function, “e” is used to encode both T and w to obtain $e(T)$ and $e(w)$.

The encoding is applied such that the reconstruction of $e(T)$ to T and $e(w)$ to w should also be possible without any loss of data.

The encoding function „e“ is given as,

$$S(\varepsilon) = 0$$

$S(a_i) = 0^{i+1}$	$[a_i \in S] [S \rightarrow \text{input symbols / terminals}]$
$S(h_a) = 0$	$[h_a \rightarrow \text{halting state}]$
$S(h_r) = 00$	$[h_r \rightarrow \text{Rejecting / Crash state}]$
$S(q_i) = 0^{i+2}$	$[q_i \rightarrow \text{set of states}]$
$S(S) = 0$	$[S \rightarrow \text{stop}]$
$S(L) = 00$	$[L \rightarrow \text{Left}]$
$S(R) = 000$	$[R \rightarrow \text{Right}]$

We know that the transition move of a Turing machine

$$\delta(q_i, T) = (q_j, T, M) \quad q_i, q_j \in Q$$

$T \rightarrow \text{terminal symbol}$

Which is encoded as,

$M \rightarrow \text{movement of head}$

$$e(m) = 1s(q_i)1s(T)1s(q_j)1s(T)1s(M)1$$

Where $e(m) \rightarrow \text{encoded transition, } m.$

$$e(w) = |1s(w_1)|S(w_2)|S(w_3)|\dots|S(w_n)|$$

where $e(w) \rightarrow \text{encoded input, } w$

For any Turing machine, T ,

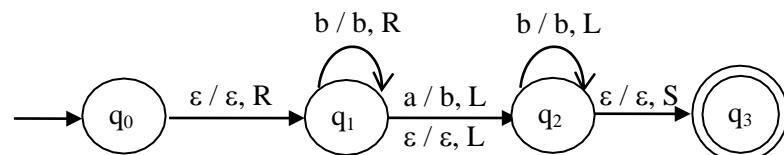
$$e(T) = |s(q)|e(m_1)|e(m_2)|\dots|e(m_k)|$$

where

$q \rightarrow \text{initial state}$

$m \rightarrow \text{transition of TM}$

Example



The initial state = $\{q_0\}$

Final state = $\{q_3\}$

5.16 Theory of Computation

Inputs	= {a,b}
Moves	= {L,R,S}
States	= {q ₀ ,q ₁ ,q ₂ ,q ₃ }

The code for,

q₀- first state


Status	$q_0 \rightarrow 000$ [q since $S(q_i) = 0^{i+2} \Rightarrow S(q_0) = 0^{1+2}$]
	$q_1 \rightarrow 0000$, [$S(q_1) = 0^{2+2}$, since q → second state]
	$q_2 \rightarrow 00000$ [$S(q_2) = 0^{3+2}$, since $q_2 \rightarrow$ third state]
	$q_3 \rightarrow 0$ [since $q_3 \rightarrow$ halt state, h _a]

$$\varepsilon \rightarrow 0 \text{ [since } s(\varepsilon) = 0\text{]}$$

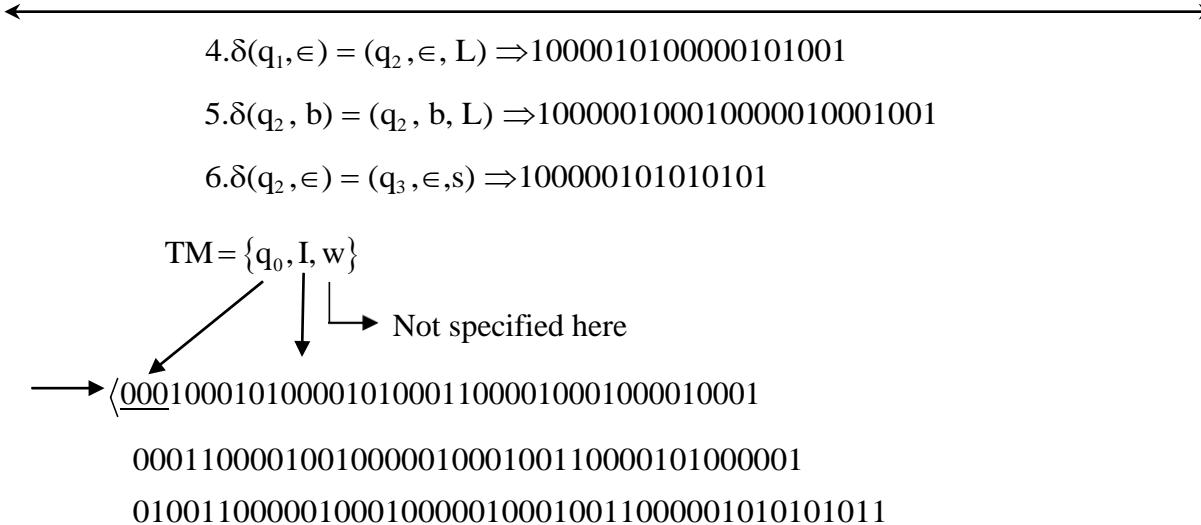
Inputs	a → 00 [since $a(a_i) = 0_{i+1}$; a → first input; so i = 1 ⇒ s(a) = 0 ¹⁺¹]
	b → 000 [$s(b) = 0^{2+1}$, since b → second input]
	s → 0 [since $s(s) = 0$]
Moves	L → 00 [since $s(L) = 00$]
	R → 000 [since $s(R) = 000$]

The transition of the above given Turing machine are,

1. $\delta(q_0, e) = (q_1, \in, R)$
2. $\delta(q_1, b) = (q_1, b, R)$
3. $\delta(q_1, a) = (q_2, b, L)$
4. $\delta(q_1, \in) = (q_2, \in, L)$
5. $\delta(q_2, b) = (q_2, b, L)$
6. $\delta(q_2, \in) = (q_3, \in, s)$

The binary string after applying encoding function

1. $\delta(q_0, \in) = (q_1, \in, R) \Rightarrow 100010100001010001$
2. $\delta(q_1, b) = (q_1, b, R) \Rightarrow 10000100010000100010001$
3. $\delta(q_1, a) = (q_2, b, L) \Rightarrow 1000010010000010001001$



Input to the T_u

The universal Turing machine, T_u is always provided with the code for Transitions, $e(T)$ and code for input, $e(w)$ as

$$TM = e(T)e(w)$$

For example, if the input data, $w = "baa"$, then

$$e(w) = 10001001001$$

This $e(w)$ will be appended to $e(T)$ of T_u .

Construction of T_u

As in the figure for universal Turing machine, there are three tapes controlled by a finite control component through heads for each tape.

Tape -1 \Rightarrow Input tape and also serves as output tape. It contain $e(T)$ $e(w)$.

Tape-2 \Rightarrow Tape of the TM/Working tape during the simulation of TM

Tape -3 \Rightarrow State of the TM, current state of the T in encoded form.

Operation of UTM

- UTM checks the input to verify whether the code for $TM = \langle T, w \rangle$ is a legitimate for some TM.
 - If the input is not accepted, UTM halts with rejecting, w
- Initialize the second tape to have $e(w)$, that is to have the input, w in encoded form.

- Place the code of the initial state on the third tape and move the head of the finite state control on the first cell of second tape.
- To simulate a move of the Turing machine, UTM searches for the transition - $o^i 1 o^j 1 o^k 1 o^l 1 o^m$ on the first tape, with o^i (initial state/current state) on tape -3 and o^j (input symbol to be processed) on tape- 2.
- The state transition is done by changing the tape -3 content as o^k as in the transition.
- Replace o^j by o^l on tape-2 to indicate the input change.
- Depending on o^m [$m=1 \Rightarrow$ stop, $m=2 \Rightarrow$ Left, $m=3 \Rightarrow$ Right], move the head on tape-2 to the position of the next 1 to the left/right/stop accordingly
- If TM has no transition, matching the simulated state and tape symbol, then no transition will be found. This happens when the TM stops also.
- If the TM, T enters h_a (accepting state), then UTM accepts the input, w

Thus for every coded pair $\langle T, w \rangle$, UTM simulates T on w, if and only if T accepts the input string, w.

Definition of Universal Language [L_u]

The universal language, L_u is the set of all binary strings $[\alpha]$, where α represents the ordered pair $\langle T, w \rangle$ where

$$\begin{aligned} T &\rightarrow \text{Turing machine} \\ w &\rightarrow \text{any input string accepted by } T \end{aligned}$$

It can also be represented as $\alpha = e(T) e(w)$.

Theorem

L_u is the recursively enumerable but not recursive .

Proof

From the definition and operations of UTM, we know that L_u is recursively enumerable.

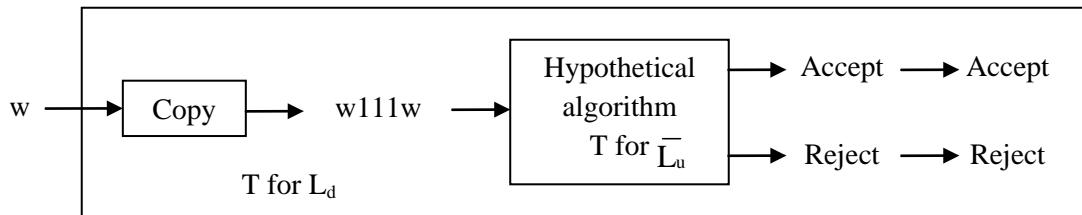
L_u accepts the string w if it is processed by the TM, T. Else, rejects „w“ and the machine doesn't halt forever.

To prove that L_u is not recursive, the proof can be done by contradiction. Let L_u is Turing decidable [recursive], and then by definition L_u (complement of L_u) is Turing acceptable.

We can show that $\overline{L_u}$ is Turing acceptable, that leads to L_d to be Turing acceptable. But we know that L_d is not Turing acceptable.

Hence L_u is not Turing decidable by proof by contradiction.

Proof on $\overline{L_u}$ is during acceptable $\Rightarrow L_d$ is Turing acceptable



Suppose “A” is the algorithm that recognizes L_u .

Then $\overline{L_d}$ is recognized as follows. Given a string $w \in (0,1)^*$ determined easily, the value of I such that $w = w_i$.

Integer value, I in binary is the corresponding code for TM, T_i . Provide $\langle T_i, w_i \rangle$ to the algorithm A and accept, w if and only if T_i accepts w_i .

So the algorithm accepts w if and only if $w = w_i$ which is in $L(T_i)$.

This is the algorithm for L_d . Hence L_u is Recursively Enumerable but not recursive.

TRACTABLE AND INTRACTABLE PROBLEMS

Tractable Problems/Languages

The languages that can be recognized by a Turing machine in finite time and with reasonable space constraint is said to be tractable.

Example: If the language $L_1 \in \text{Time}(f)$, then L is tractable and is less complex in nature

Example: If $L_2 \notin \text{Time}(f)$, L2 is complex and cannot be tractable in limited time.

Tractable problems are those that can be solved in polynomial time period.

Intractable Problems

The languages that cannot be recognized by any Turing machine with reasonable space and time constraint is called intractable problems.

These problems cannot be solved in finite polynomial time. Even problems with moderate input size cannot achieve feasible solution

Terminologies

Polynomial Time, PSet

The set of languages that can be recognized by a Turing machine with polynomial time complexity.

It is given by,

$$P = \bigcup_{k \geq 0} \text{UTime}(Cn^k)$$

$$c > 0, k \geq 0$$

Polynomial Space, PSpace Set

The set of languages that can be recognized by a TM with polynomial space complexity is said to be PSpace Set.

It is given by,

$$\text{PSpace} = \bigcup_{k \geq 0} \text{Space}(Cn^k)$$

$$c > 0, k \geq 0$$

Non-Deterministic Polynomial Time, NP

The set of languages that are recognized by a Nondeterministic TM with polynomial time is said to be NP.

It is given by,

$$\text{NP} = \bigcup_{k \geq 0} \text{NTIME}(Cn^k)$$

$$c > 0, k \geq 0$$

Non-deterministic Polynomial Space, N Space

The set of languages that can be recognized by a NDTH with polynomial Space complexity

It is given by,

$$\text{NP} = \bigcup_{k \geq 0} \text{NSPACE}(Cn^k)$$

$$c > 0, k \geq 0$$

Example: The CNF satisfiability Problem

Satisfiability problem is also called as Boolean Satisfiability /Proportional satisfiability/SAT problem.

The SAT problem is to find if there is any interpretation that satisfies the given Boolean equation.

It is to determine the variables of a given boolean formula can be replaced by „TRUE“ or „FALSE“ in a form that evaluates the formula to „TRUE“.

If „TRUE“ is obtained, the Boolean equation is said to be satisfiable.

If there is no assignment obtained and the evaluation gives out „FALSE“, then the Boolean equation is said to be unsatisfiable.

Example: Let a= TRUE

$$b=\text{FALSE}$$

$$a \text{ AND } b \Rightarrow \text{FALSE}$$

$$a \text{ AND NOT } a \Rightarrow \text{FALSE}$$

$$a \text{ AND NOT } b \Rightarrow \text{TRUE}$$

$$a \text{ OR } b \Rightarrow \text{TRUE}$$

- The operator „AND“ is denoted by conjunction operator (\wedge)
- The OR operator is given by disjunction (\vee).
- NOT operator is denoted by negation (\neg) operator

Example: $a \wedge b \Rightarrow \text{FALSE}$

$$a \wedge (\neg a) \Rightarrow \text{FALSE}$$

$$a \wedge (\neg b) \Rightarrow \text{TRUE}$$

$$a \vee b \Rightarrow \text{TRUE}$$

Terminologies

The variable used in the Boolean expression is called a literal. Literals can be either positive or negative.

The normal variable is called positive literal.

The negation of a variable is called negative literal.

The Boolean clause is said to be Horn clause if it contains at most one positive literal.

A formula is in CNF (Conjunctive normal form) if it contains/formed by a conjunction of single clauses.

Example: $x_1 \Rightarrow$ Positive literal

$\neg x_2 \Rightarrow$ Positive literal

$x_1 \vee \neg x_2 \Rightarrow$ Clause

$\neg x_2 \vee x_2 \vee x_3 \Rightarrow$ Not a horn clause

$\neg x_1 \Rightarrow$ horn clause

$x_1 \vee \neg x_2$

$\neg x_1 \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2) \Rightarrow$ CNF

Thus CNF is a conjunction of clauses to form a formula.

$C_1 \wedge C_2 \wedge C_3 \wedge \dots \wedge C_n \Rightarrow$ CNF

where,

$C_1, C_2, C_3 \wedge \dots \Rightarrow$ Conjuncts / clauses.

K-satisfiability

- Generalized form of CNF with each clause containing upto „k“ literals.

Encoding instances of CNF SAT:-

The instances of CNF SAT can be encoded as,

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee \neg x_1) \wedge (\neg x_4 \vee x_2)$$

$$\Rightarrow \neg x_1 \neg x_1 1 \wedge x_1 1 x_1 1 1 \neg x_1 \wedge \neg x_1 1 1 1 x_1 1$$

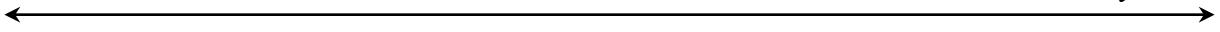
The encoding scheme omits parenthesis and OR (V) notation. The variables are denoted by unary notation of their subscripts.

Thus CNF-Satisfiability is the language over $\Sigma = \{\wedge, 1, x, \neg x\}$ containing encodings for all positive literals of the instances in CNF SAT.

When there are k-literals, c-conjucts, v-distinct literals, and n is the length of the string encoding these instances,

$$n \leq k(V+1) + \leq k^2 + 2k$$

If CNF – satisfiable \in NP , then the decision problem CNF-SAT is in NP.



P AND NP COMPLETENESS

Problems are classified into two, namely P and NP.

P

- „P“ refers to the class of problems that can be solved in polynomial time.
- Example: Searching an element in an ordered list, sorting elements in a lists, Multiplication of integers , finding all- pair –shortest path, finding minimum spanning tree of a graph, etc
- These are also referred as tractable problems
- Polynomial – time algorithms are efficient ones that can be solved more rapidly.

NP

- „NP“ refers to the class of problems that are solved by non-deterministic polynomial time
- These types of NP problems are known as interactable problems.
- Example: Towers of Hanoi, Traveling Salesman problem, Graph colouring problem, Hamiltonian Circuit problem, Satisfiability Problem, etc.
- If a problem, P is solvable in polynomial time by NDTM, then there is no guarantee that there exists a Deterministic TM that can solve P in polynomial time.
- If P is a set of tractable problem, then $P \subseteq NP$
- Every deterministic TM is a special case of case of NDTM.
- NP complete Problem – Types:-
- The NP type class of problems are classified into
 - NP – COMPLETE
 - NP – HARD

NP- Complete Problem

A problem is said to be NP- complete if it belongs to NP class problem and can be solved in polynomial time.

They are also called polynomial -time reducible problems.

A NP-complete problem can be transformed into any other in polynomial time.

NP-Hard Problem

- A problem is said to be NP hard if there exists an algorithm for solving it and it can be translated into one for solving another NP-Problem.
- A Problem P_1 is NP-hard if
 - the problem is an NP class problem
 - For any other problem , P_2 in NP, there is a polynomial time reduction of L_2 to L_1
- Every NP complete problem must be NP- hard problem

