Theory of Computation

(TOC)

Bachelor of Engineering

Er. Urbara Bhandari
Suraj Kumar Ojha

Publisher

G. L. Book House Pvt. Ltd.

Infront of Thapathali Engineering Campus, Maitighar, Kathmandu

Table of Content

Syllabus

Chapter – 1: Introduction

1.1 Set, relation, function, proof, techniques
1.2 Alphabets, language, regular expression
1.3 Solved out examination problems

Chapter – 2: Finite Automata

2.1 Deterministic finite automata
2.2 Non-deterministic finite automata
2.3 Equivalence of finite automata and regular language
2.4 Regular language, properties of regular language
2.5 Pumping lemma for regular language
2.6 Solved out examination problems

Chapter – 3: Context Free Grammar

3.1 Introduction to context free grammar
3.2 Derivation trees, simplification of context free grammar
3.3 Chomsky normal form
3.4 Greibach normal form
3.5 Pumping lemma for context free language
3.6 Properties of context free language
3.7 Pushdown automata
3.8 Equivalence of context free language and push down automata
3.9 Solved out examination problems

Chapter – 4: Turing Machine

4.1 Definition of Turing machine
4.2 Computing with Turing machine
4.3 Combination of Turing machine
4.4 Extensions of Turing machine
4.5 Unrestricted grammar
4.6 Recursive function theory
4.7 Solved out Examination problems

Chapter – 5: Undecidability

5.1 Church Turing thesis
5.2 Universal Turing machine
5.3 Halting problem
5.4 Undecidable problems about Turing machines, grammars
5.5 Recursive and recursively enumerable language and its properties
5.6 Solved out examination problems

Chapter – 6: Computational Complexity

6.1 Introduction to computational complexity
6.2 Class P, NP problems NP hard and NP complete problems
6.3 Solved out examination problems

Chapter - 1
Introduction

In this chapter, we shall discuss about

- Set
- Function
- Alphabets
- Regular Expression
- Relation
- Proof Techniques
- Language

1. Set

As we already know, set is a collection of well-defined objects. For example,

S = {5, 10, 15}; V = {a, e, i, o, u}

Each object of set is called element.

Types of set

(a) Empty set: A set having no element. It is denoted by Greek letter ϕ. For example,

A = {x : x is a male student in a girl's campus} = ϕ.

(b) Finite set: A set having definite number of elements. For example,

A = {x : x is vowels in English language}
= {a, e, i, o, u}

(c) Infinite sets: A set which is not finite. For example,

A = {x : x are natural numbers}
= {1, 2, 3, 4, 5, 6, ...}

(d) Singleton set: A set containing only one element. For example,

A = {f : x is divisor of prime number other than prime number itself}
= {1}

Relation between sets

Two sets A and B (say) may have common elements between them and there may exist relation between them defined as:

(a) **Subset:** A set A is said to be the subset of B if all element of A belongs to set B. It is written as $A \subseteq B$ For example,

$A = \{x : x \text{ is a letter in English alphabet}\}$

$B = \{x : x \text{ is a vowel}\}$

Thus, B is a subset of A i.e. $B \subset A$.

(b) **Equal sets:** Two sets A and B are said to be equal sets if they have same elements in any order. For example,

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

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

Then $A = B$

(c) **Intersecting sets:** Two sets A and B are intersecting sets if they have at least one element common in them.

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

$B = \{a, b, c, d, e\}$

(d) **Disjoint sets:** Two sets A and B are disjoint set if they have no element in common. For example,

$A = \{a, b, c\}$

$B = \{x, y, z\}$

(e) **Equivalent sets:** Two sets A and B said to be equivalent sets if they have same number of elements. For example,

$A = \{0, 1, 2, 3\}$

$B = \{a, b, c, d\}$

Here, $n(A) = n(B)$

(f) **Power set:** It is the set of any possible subset of any set. For example,

$A = \{0, 1\}$

Then, power set of $A = 2^{\wedge} = 2^2 = \{\{0\}, \{1\}, \{0, 1\}, \phi\}$

Other examples of type of sets

(a) **Universal set:** A fixed state is universal set if it contains all the subjects under discussion. For example,

For the sets of people of different countries, the set of people in the world is universal set.

(b) **Uncountable set:** A set is uncountable if it contains so many elements that they can't be put one to one correspondence with the set of natural numbers. In other words, it is opposite to that of countably infinite set. For example,

The set of real numbers in {0, 1}

(c) **Countably infinite set:** A set is said to be countably infinite set if its elements can be put one-one correspondence with the set of natural numbers. For example,

$Z = \{x : x \text{ is a element of integers}\}$

$= \{-3, -2, -1, 0, 1, 2, 3, ...\}$

Here, we can't never count the cardinality of sets but if we arrange elements is such a way that

$Z = \{0, -1, 2, -2, 2, -3, 3, ...\}$

And then if we are asked to count number of elements up to -3, we can do that. So, the set of integers is the countably infinite set.

Operation on sets

(a) **Union:** Union of two sets A and B is the set of all elements belonging to at least one of two sets or both. It is denoted by $(A \cup B)$.

$A \cup B = \{x : x \in A \text{ or } x \in B\}$

(b) **Intersection:** Intersection of two sets A and B is the set of all elements belonging to both of the sets. It is denoted by $(A \cap B)$.

$A \cap B = \{x : x \in A \text{ and } x \in B\}$

(c) **Difference:** The difference of two sets A and B denoted by A – B is the set of all elements of A a that aren't elements of B.

$A - B = \{x : x \in A \text{ and } x \notin B\}$

(d) **Symmetric difference:** Symmetric difference of two sets A and B is the set of all elements which are in either of the sets and not in their intersection. The symmetric difference is denoted by

$A \Delta B = (A - B) \cup (B - A)$

Properties of set operation

(a) Indempotency → $A \cup A = A$

→ $A \cap A = A$

(b) Commutativity → $A \cup B = B \cup A$

→ $A \cap B = B \cap A$

(c) Associativity → $A \cup (B \cup C) = (A \cup B) \cup C$
→ $A \cap (B \cap C) = (A \cap B) \cap C$

(d) Diustribuity → $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$
→ $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$

(e) Absorption → $(A \cup B) \cap A = A$
→ $(A \cap B) \cup A = A$

(f) De-Morgan's law → $A - (B \cup C) = (A - B) \cap (A - C)$
→ $A - (B \cap C) = (A - B) \cup (A - C)$

Ordered pairs

The pair in which first element belongs to first set and second element belongs to second set is called ordered pairs. For ordered pairs (a, b)

→ $a \in A$ and $b \in B$.

Cartesian product: The Cartesian product of two sets A and B denoted by $A \times B$ is the set of all ordered pairs (a, b) with $a \in A$ and $b \in B$. Note that $A \times B \neq B \times A$

i.e. $A \times B = \{(a, b) : a \in A, b \in B\}$

$B \times A = \{(c, d) : c \in B, d \in A\}$

For example,

Let $A = \{a, b\}$ and $B = \{c, d\}$ be the two sets

Then $A \times B = \{(a, c), (a, d), (b, c), (b, d)\}$

$B \times A = \{(c, a), (c, b), (d, a), (d, b)\}$

∴ $A \times B \; B \times A$.

2. Relations

Relation between sets is the every possible subset of Cartesian products between them. Generally, it is denoted by R i.e. if A and B are sets as defined above then one of the relation.

$R_1 = \{(1, 3), (2, 4)\}$

$R_2 = \{(1, 4), (2, 5)\}$

If A has m elements and B has n elements then the possible number of relation between them is 2^{mn}.

Binary relation: If relation is formed from subsets of Cartesian product of two sets then it is binary relation.

Types of relation

(a) **Reflexive relation:** A binary relation R over a set A is reflexive if $(a, a) \in A$ i.e. every element of X is related to itself. In mathematics "is equal to" relation between real number is reflexive.

(b) **Symmetric relation:** A binary relation over a set A i.e. $R \subseteq A \times A$ is symmetric if $(b, a) \in R$ wherever $(a, b) \in R$.

For example, $A = \{a, b, c\}$

Symmetric relation $R = \{(a, b), (b, a), (b, c), (c, a), (a, c), (c, b)\}$

Asymmetric relation: $(b, a) \notin R$ whenever (a, b)

In mathematics "is equal to" relation between real numbers is symmetric whereas "is less than" is asymmetric.

(c) **Transitive relation:** A binary relation on set A is transitive relation if $a, b, c \in A$ and a is related to b and b is related to c then a is related to C. In mathematics "is greater than" or "is less than" relation between number is transitive relation.

(d) **Equivalence relation:** A relation that is reflexive, symmetric and transitive is equivalent relation.

3. Function

A special type of relation which associates each element of set A with a unique element of B is called function. Let $A = \{1, 2\}$ and $B = \{3, 4\}$.

For example, let relation $R_1 = \{(1, 3), (2, 4)\}$

Here, in relation R_1 for every 1st element there corresponds a unique second element which is greater than first element. And this incremented by 2 is a function.

Types of functions

(a) **One-one or injective function:** A function from set A to set B i.e. $f : A \to B$ is one-one function if distinct element of A is associated with distinct element of B.

For set A and B defined above

$R_1 \subset A \times B = \{(1, 3), (2, 4)\}$ is one-one function

(b) **Onto function:** A function from set A to set B i.e. $A : A \to B$ is onto function if every element of B is associated with at least one element of A.

For set A and set B defined above

$R_2 \subset A \times B = \{(1, 3), (2, 4)\}$ is onto function,

Here, no element of B is left-over.

(c) **Bijection function:** Any function R satisfying both one-one and onto function i.e. R_1 and R_2 is bijection function. For example,

$R \subset A \times B = \{(1, 3), (2, 4)\}$

4. Proof Techniques

The three fundamental proof techniques are described below:

(a) **The principle of mathematical induction:** It states that if a statement P(n) is:

(i) P(0) or P(1) is true

(ii) P(n + 1) is true when P(n) is assumed to be true.

Then P(n) is true for $n \in N$

where N = set of natural numbers

Working rules:

- Denote the given statement (i.e. the result to be proved) by P(m).
- Show that the given statement is true for n = 1 i.e. P(m) is true.
- Assume that the given statement is true for n = m i.e. assume P(n) is true.
- Show that the statement is true for n = m + 1 when P(n) is true.
- The statement is true for all $n \in N$.

For example, Prove by the principle of mathematical induction that:

$$0 + 1 + 2 + 3 + \ldots = \frac{n(n+1)}{2}$$

Step – I: Let P(n) be the statement

$$P(n) = 0 + 1 + 2 + \ldots = \frac{n(n+1)}{2}$$

when n = 0;

From LHS;

P(0) = 0

From RHS;

$$P(0) = \frac{0(0+1)}{0} = 0$$

∴ This statement is true for P(0).

Step – II: Let n = m be true then P(m)

$$P(m) = 0 + 1 + 2 + \ldots + m = \frac{m(m+1)}{2} \quad \ldots\ldots\ldots (i)$$

Step – III: For n = m + 1, we have to show that P(m + 1) is true when P(m) is true

For P(m + 1); LHS = 0 + 1 + 2 + ... m + (m + 1)

$$\Rightarrow \frac{m(m+1)}{2} + (m+1) \quad \text{[From (i)]}$$

$$\Rightarrow \frac{m(m+1) + 2(m+1)}{2}$$

$$\Rightarrow \frac{(m+1)(m+2)}{2}$$

$$\Rightarrow \frac{(m+1)\{(m+1)+1\}}{2}$$

= RHS

∴ Since, LHS = RHS. This statement is true for P(m + 1) when P(m) is true. Hence, by principle of mathematical induction, this statement is true for all $n \in N$.

4. Proof Techniques

(a) **Pegion hole principle:** If A and B are finite sets and |A| > |B| then there is no one-one function from A to B. If an attempt is made to pair, off the element of set A (pigeon) with elements of set B (pigeon holes). Sooner or later we will have to put more than one pigeon in a pigeon hole.

(b) **Diagonalization principle:** The diagonalization principle is based m and simple observation. Its principle are as follows:

(i) Assume condition for contradiction.

(ii) Find the reversed diagonal and check whether it is different from each row in table or not.

(iii) If it is different from each row in table, it contradict the assumed condition proving the theorem.

Explanation

Let A be a finite set A = {a, b, c, d} and R be a binary relation on set A such that

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

Now, we can represent R in a square table, rows and columns representing the elements and cell having 1S if there is a link between the corresponding elements.

	a	b	c	d
a	0	1	1	0
b	0	1	0	1
c	0	1	0	1
d	1	0	0	0

The diagonal element is 0, 1, 0, 0 and taking the complement of diagonal (replacing 1S with OS and vice-versa) is different from each row. The reversed diagonal element is 1, 0, 1, 1 and you can see that it is different from each row.

And if the reversed diagonal is different from each row, it contradicts assumptions with what table is made upon. And in the way, the theorem can be proved.

5. Alphabets

If is a finite, non-empty set of symbols denoted by 'Σ'

Σ = {0, 1} are binary alphabets.

= {*, #, π} are character alphabets.

= {A, B, C, D} are uppercase alphabets.

= {a, b, c, d} are lowercase alphabets.

Strings: Strings are finite sequence of symbols taken from alphabets not separated by commas, denoted by 'w'

(a) **Length of string:** Total number of alphabets in a string is referred as length of string. It is denoted by |w| for example, |w| = |001| = 3.

(b) **Empty string:** The string that has zero occurrence of symbol. i.e. Σ or e not Σ)

Σ* = set of all strings including empty strings over an alphabet Σ is denoted by Σ*. For example

(i) Σ = {0, 1}

Σ* = {∈ 0, 1, 01, 001, 10, ...}

(ii) Σ = {a}

Σ* = {a, aa, aaa, ∈, ...}

(c) **Concatenation of string:** Assume a string x and y then concatenation of x and y is xy.

(d) **Substring:** A string V is said to be valid substring of a string w if w = x Vy where x and y are also substring that could by ∈. (empty string)

(e) **Reversal of string:** Let w be any string defined as;

$w = w_1, w_2, w_3, \ldots w_n$, w = abc

then $w^R = w_n, w_{n-1}, w_{n-2}, \ldots w_1$, w^R = cba

6. Language

A set of strings over an alphabet ∈ is called a language. It is set of all strings including the empty string or an alphabet.

- L = {w ∈ {a, b}*, w has odd number of a's}

  = {a, ab, aaba, ...}

  Note: ∈ → belongs to (in this case)

- L = {w ∈ {a, b}*, w has equal number of a's and b's}

  = {ab, abab, abba, ...}

(a) **Concatenation of language:** Given languages L_1 and L_2 we define their concatenation to be the language $L_1 \circ L_2 = \{xy \mid x \in L_1, y \in L_2\}$

Example - 1: L_1 = {hello} and L_2 = {world}

then $L_1 \circ L_2$ = {helloworld}

Example - 2: If L_1 = {001, 10, 111}, L_2 = {∈, 001}

then $L_1 \circ L_2$ = {001, 10, 111, 001001, 10001, 111001}

(b) **Kleene star:** Another language operation is kleene star, denoted by L^*. It is also called closure of language

The positive closure of langue L is L^* i.e.

$L^+ = L^* - \in$

L^+ is infinite union, Ui ≥ Li when $l > 0$

(c) **Union of two language:** The union of two languages L_1 and L_2 denoted by

$L_1 \cup L_2$ is defined as;

L_1 = {001, 10, 111}

L_2 = {∈, 001}

∴ $L_1 \cup L_2$ = {10, 001, 111}

Regular language: The language that can be constructed from the set operation union, concatenation and kleene star. We would discuss more about it in latter chapter.

7. Regular Expression

It is designed to represent regular language with mathematical tool to build set of strings combining string of symbol from some alphabet ∈ and operator +, . and *.

0 + 1 represent the set {0, 1}

Then, the regular expression is;

$(0 + 1)^* = \{\in + (0 + 1) + (0 + 1)(0 + 1) + \ldots\}$

$(0 + 1)^+ = \{(0 + 1) + (0 + 1)(0 + 1) + \ldots\}$

Building regular expressions: Assume that Σ = {a, b, c}

(a) Zero or more: → a* means zero or more a's.

(b) One or more: → a.a*

(c) Zero or one: → (a + λ) where λ denotes empty string

(d) Any string at all: (a + b + c)* (could be empty as well)

(e) Any non- empty string: (a + b + c) (a + b + c)*

(f) Any string not containing ...: To describe any string at all that does not contain an 'a' (with Σ = {a, b, c}), you can use (b + c)*

(g) Any string containing exactly one ...: To describe any string that contains exactly one 'a' put "any string containing a" on either side of the 'a' like.

(b + c)* a (b + c)*

Regular expression

Regular language: The regular language are those language that can be constructed from 'big three' set operation viz (a) union '+' (b) concatenation 'o' (c) Kleen star (*).

Definition: d Let Σ be an alphabet. The class of 'regular languages' over Σ is defined inductively as follows:

(a) ϕ is a regular language.

(b) For each σ ∈ Σ, {σ} is a regular language

(c) For any natural number n ≥ 2 if $L_1, L_2, \ldots L_n$ are regular languages then so is $L_1 \cup L_2 \ldots L_n$.

(d) For any natural number n ≥ 2, if $L_1, L_2, \ldots L_n$ are regular languages then so is $L_1 \circ L_2 \circ \ldots L_n$

(e) If L is regular then so is L*

Identities of regular expression

(a) ϕ + R = R
(b) ϕR + Rϕ = ϕ
(c) ∈R = R∈ = R
(d) ∈* = ∈ and ϕ* = ∈
(e) R + R = R
(f) R* R* = R
(g) RR* = R* R
(h) (R*)* = R*
(i) ∈ + RR* = ∈ + R* R = R*
(j) (PQ)* P = P(QP)*
(k) (P + Q)* = (P* Q*)* = (P* + Q*)*
(l) (P + Q)R = PR + QR and R(P + Q) = RP + RQ

1. Write a regular expression that generate3 strings that

(a) Start with ab
ab (a + b)*

(b) End with ab
(a + b)* ab

(c) Contains a substring aab
(a + b)* aab (a +b)*

(d) Start and ends with a
a + a (a + b)* a

(e) Starts and ends with same symbol
a + a (a + b)* a + b + b (a + b)* b + ∈

(f) Start and end with different symbol
(a (a + b)* b)+ + (b (a + b)* a)+

(g) |S| = 3 (i.e. length of any string s is 3)
(a + b) (a + b) (a + b)

(h) |S| ≤ 3 = ∈ + (a + b) + (a + b) (a + b) + (a + b) (a + b) (a + b)
or, (a + b + ∈) (a + b + ∈) (a + b + ∈)

(i) |S| ≥ 3
$(a + b)^3 (a + b)^*$
or, (a + b) (a + b) (a + b) (a + b)*

(j) number of a's = 2 i.e. |w| a = 2
b* ab* ab*

(k) |w|a ≥ 2
(a + b)* a (a + b)* a (a + b)*

(l) 3rd symbol from left end is b
(a + b) (a + b) b (a + b)*
$(a + b)^2 b (a + b)^*$

(m) $|S| = 0 \pmod 3$

$\{(a + b)(a + b)(a + b)\}^*$

$\{(a + b)^3\}^*$

(n) $|w| = 2 \pmod 3$

$(a + b)^2 \{(a + b)^3\}^*$

(o) $|w|_b = 0 \pmod 2$

$(a^* ba^* ba^*)^*$

(p) $|w|_a = 1 \pmod 3$

$b^* ab^* (b^* ab^* ab^* ab^*)^*$

(q) $|w|_b = 2 \pmod 3$

$a^* ba^* ba^* (a^* ba^* ba^* ba^*)^*$

2. Obtain the regular expression for the language given by

(a) $L_1 = \{a^{2n} b^{2m+1} \mid n \geq 0, m \geq 0\}$

$R_1 = (aa)^* (bb)^* b$

(b) $L_2 = \{a, bb, aa, abb, ba, bbb, ...\}$

$R_2 = (a + b)^* (a + bb)$

(c) $L_3 = \{w \in \{0, 1)^* \mid w$ has no pair of consecutive zeros$\}$

$R_3 = (1^* 011^*)^* (0 + \in) + 1^* (0 + \in)$

(d) $L_4 = \{$string of 0's and 1's beginning with 0 and ending with 1$\}$

$R_4 = 0 (0 + 1)^* 1$

(e) $L_5 = \{$strings containing a's at most over alphabet $\{a, b\}\}$

$R_5 = b^* (a + \in) b^* (a + \in) b^*$

(f) $L_6 = \{$strings containing a's at least 2 over alphabet $\{a, b\}\}$

$R_6 = b^* ab^* a (a + b)^*$

(g) $L_7 = \{$strings containing only even no of b's$\}$

$R_7 = (a^* ba^* ba)^*$

(h) $L_8 = \{$string containing a's exactly 2 over $\{a, b\}\}$

$R_8 = b^* ab^* ab^*$

(i) $L_8 = \{01, 10, 0101, 101010, ...\}$

$R_9 = (01 + 10)^*$

(j) $L_{10} = \{$string that doesn't have aab as its substring over alphabet

$R_{10} = (ab \cup b)^* a^*$

Exam Solution

1. Define diagonalization principle. Explain principle of mathematical induction with suitable example. **[2075 Ashwin, Back]**

Let R be a binary relation on a set A and let 0 $\{a \mid a \in A$ and $(a, a) \notin R\}$. For each $a \in A$, let $Ra = \{b \mid b \in A$ and $(a, b) \in R\}$. Then D is distinct from Ra for all $a \in A$.

Proof:

Let $A = \{a, b, c, d\}$ and $R = \{(a, b), (c, d), (b, b), (b, c), (c, c), (d, b)\}$ R can e represented as a square array as below:

		a	b	c	d
R_a	a		×		×
R_b	b		×	×	
R_c	c			×	
R_d	d		×		

$Ra = \{b, d\}$, $R_b = \{b, c\}$, $R_c = \{c\}$, $R_d = \{b\}$

$D = \{a, d\}$

Clearly $d \neq Ra; Rb; Rc; Rd$

The diagonalization principle holds for infinite set as well.

The principle of mathematical induction states that if a statement P(n) is

(i) P(0) or P(1) is true

(ii) P(n + 1) is true when P(n) is assumed to be true.

Then P(n) is true for $n \in N$ where N = set of natural numbers.

Prove by mathematical induction method that $2^{3n} - 1$ is divisible by 7.

Let, P(n) be the given statement i.e.

$P(n) = 2^{3n} - 1$ is divisible by 7.

when n = 1

$P(1) = 2^3 - 1 = 7$

$\therefore$ P(1) is true

Let, us suppose that P(n) is true for n = m where $m \in N$.

$P(m) = 2^{3m} - 1$ is divisible by 7 ..... (i)

Now, we shall snow that P(m + 1) is true when P(m) is true

i.e. we show that $2^{3(m+1)} - 1$ is divisible by 7.

Now, $2^{3(m+1)} - 1 = 2^{3m+3} - 1$

$= 2^{3m} 2^3 - 1$

$= 2^{3m} . 8 - 8 + 8 - 1$

$= 8(2^{3m} - 1) + 7$

which is divisible by 7 as $(2^{3m} - 1)$ is divisible by 7 by (i). This relation shows that p(m + 1) is true whenever P(m) is true. Hence, by the principle of mathematical induction P(n) is true for all $n \in N$.

2. **Write a regular expression for the language in which strings start and end with different symbol over alphabet $\Sigma = \{a, b\}$.**

[2075 Ashwin, Back]

Let, R be the required regular expression for the language L in which strings start and end with different symbol over alphabet $\Sigma = \{0, 1\}$

Here,

$L = \{ab, ba, abb, baa, ...\}$

$\therefore R = \{a(a + b)^* b)^* + (b(a + b)^* a)^+$

3. **State the diagonalization principle. Use principle of mathematical induction to prove $n^4 - 4n^2$ is divisible by 3 for $n \geq 0$.**

[2074 Ashwin, Back]

Diagonalization principle states that,

Let R be a binary relation on set A and let $D = \{a \mid a \in A \text{ and } (a, a) \notin R\}$. For each $a \in A$, let $Ra = \{b \mid b \in A \text{ and } (a, b) \in R\}$. Then D is distinct from Ra for all $a \in A$.

Proof:

Let $A = \{a, b, c, d\}$ and $R = \{(a, b), (a, d), (b, b), (b, c), (c, c), (d, b)\}$ R can be represented as a square array as below:

	a	b	c	d
Ra → a		×		×
Rb → b		×	×	
Rc → c			×	
Rd → d		×		

Here, $Ra = \{b, d\}$, $Rb = \{b, c\}$

$Rc = \{c\}$, $Rd = \{b\}$

and $D = \{a, d\}$

Clearly, $D \neq Ra, Rb, Rc, Rd$

The diagonalization principle holds for infinite set as well.

To prove: $n^4 - 4n^2$ is divisible by 3.

Proof: Let, P(n) be the statement

$P(n) = n^4 - 4n^2$ is divisible by 3

Then,

$P(O) = 0 - 4 \times 0 = 0$ divisible by 3

$P(1) = 1 - 4 = -3$ divisible by 3

This statement holds true for n = 0, 1.

Now,

Any number $n \in N$ can be expressed in the form of

3m, 3m + 1, 3m + 2

For example, $5 = 3 \times 1 + 2 = 3m + 2$ (m = 1)

$6 = 3 \times 2 = 3m$ (m = 2)

$7 = 3 \times 2 + 1 = 3m + 1$ (m = 1)

Then,

Let us say this theorem holds true for n = m

i.e. $P(m) = m^4 - 4m^2$

$= m^2 (m^2 - 4)$

$= m^2 (m - 2) (m + 2)$ divisible by 3

Now,

Case – I: When n = 3m

$P(3m) = (3m)^2 (3m - 2) (3m + 2)$

$= 9m^2 (3m - 2) (3m + 2)$ ..... (i)

Here, expression (i) is divisible by 3.

Case – II: When n = 3m + 1

$P(3m + 1) = (3m + 1)^2 (3m + 1 - 2) (3m + 1 + 2)$

$= (3m + 1)^2 (3m - 1) (3m + 3)$

$= (3m + 1)^2 (3m - 1) 3 (m + 1)$ ..... (ii)

Here, expression (ii) is divisible by 3.

Case – III: When n = 3m + 2

$P(3m + 2) = (3m + 2)^2 (3m + 2 - 2) (3m + 2 + 2)$

$= (3m + 2)^2 (3m) (3m + 4)$ ..... (iii)

Here, expression (iii) is divisible by 3.

Since, $\forall n \in N$ can be expressed in one of the three cases defined above and all the three cases are divisible by 3. So, for $\forall n \in N$, this theorem holds true by the principle of mathematical induction.

4. Explain devetailing technique with suitable example write regular expression for even number of (a) followed by odd number of (b) over an alphabet Σ = {a, b}. [2073 Chaitra]

Dovetailing technique is a technique used in algorithm design that interweaves different computations, performing them essentially simultaneously. Algorithms that use dovetailing are sometimes referred as to devetailers it can be used to prove certain theorems. For example,

Show that the union of a countably infinite collection of countably infinite sets is countably infinite using dovetailing technique.

Proof:

Let us show that N × N is countably infinite, note that N × N is the union of {0} × N, {1} × N, {2} × N and so on, that is, the union of a countably infinite collection of countably infinite sets. We proceed as follows:

(a) In the first round, we visit one element from the first set (0, 0)

(b) In the second reound, we visit the next element from the first set, (0, 1) and also the first element from the second set (1, 0).

(c) In the third round, we visit the next unvisited elements of the first and second sets (0, 2) and (1, 1) and also the first element of the third set (2, 0).

(d) In general, in the n^{th} round, we visit the n^{th} element of the first set, the (n – 1) st element of the second set, and the first element of the n^{th} set.

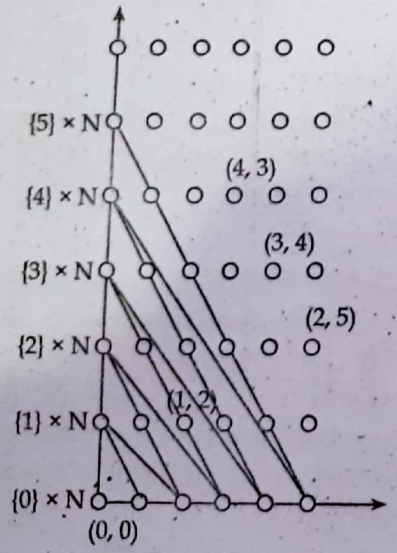


Fig. Dovetailing technique

In this way, using dovetailing technique, it can be shown that the union of a countably infinite collection of countably infinite sets is countably infinite.

Let R be the required regular expression for even number of 'a' followed by odd number of 'b' over an alphabet Σ = {a, b}. Here, the language generates the following strings.

L = {b, aab, aabbb, aabbbbb, ...}

∴ R = (aa)* b (bb)*

5. State the pigeonhole principle prove the following statement by using mathematical induction

$1 \times 1! + 2 \times 2! + 3 \times 3! + \ldots + n \times n!$

$= (n+1)! - 1$ where $(n \geq 1)$ [2073 Shrawan Back]

The pigeonhole principle states that "if A and B are finite sets and $|A| > |B|$, then there is no one-one function from A to B.

In other words if we attempt to pair off the elements of A ("the pigeons") with elements of B ("the pigeonholes"). Sooner or later we will have to put more than one pigeon in a pigeonhole.

Prove by induction

Solution:

Let, P(n) be the statement

i.e. $P(n) = 1 \times 1! + 2 \times 2! + 3 \times 3! + \ldots n \times n!$

$= (n+1)! - 1$

when n = 1

LHS: $P(1) = 1 \times 1 = 1$

RHS: $P(1) = (1+1)! - 1 = 2 - 1 = 1$

Since, LHS = RHS. This theorem holds true for n = 1.

when n = 2

LHS $P(2) = 1 \times 1 + 2 \times 2 = 5$

RHS $P(2) = (2+1)! - 1$

$= 6 - 1$

$= 5$

Since, LHS = RHS this theorem holds true for n = 2.

Let's suppose this theorem is true for n = m.

Then,

$P(m) = 1 \times 1! + 2 \times 2! + 3 \times 3! + \ldots m \times m! = (m+1)! - 1$

when n = m + 1

$$P(m+1) = 1 \times 1! + 2 \times 2! + 3 \times 3! + \ldots + m \times m! + (m+1) \times (m+1)!$$
$$= (m+1)! - 1 + (m+1) \times (m+1)!$$
$$= (m+1)! \{(m+1) + 1\} - 1$$
$$= (m+1)! (m+2) - 1$$
$$= (m+2)! - 1$$
$$= \{(m+1) + 1\}! - 1$$

So, this theorem also holds true for $n = m + 1$ when $n = m$ true. Hence, by the principle of mathematical induction, this theorem is true for $n \geq 1$.

6. Define cartesian product use mathematical induction to show $1 \times 1! + 2.2! + \ldots + n.n! = (n+1)! - 1$ for $n \geq 1$. **[2072 Chaitra]**

Cartesian product of two sets A and B is defined as the set of ordered pairs (a, b) with $a \in A$ and $b \in B$. It is denoted by $A \times B$ note that $A \times B \neq B \times A$

$$A \times B = \{(a, b) : a \in A, b \in B\}$$
$$B \times A = \{(c, d) : c \in B, d \in A\}$$

Let, $A = \{1, 2\}$ and $b = \{3, 4\}$ be the two sets

Then,

$$A \times B = \{(1, 3), (1, 4), (2, 3), (2, 4)\}$$
$$B \times A = \{(3, 1), (3, 2), (4, 1), (4, 2)\}$$
$$\therefore \quad A \times B \neq B \times A$$

[For proof of mathematical induction, please refer to 2073 Shrawan]

7. Find the regular expression for the language $L = \{w \in (0, 9)^* : 0101$ has substring. **[2072 Chaitra]**

Let R be the required regular expression for the language $L = \{w \in \{0, 1\}^* :$ has 0101 as substring$\}$. This language L always generates strings w containing '0101' as substring.

$$R = (0 + 1)^* 0101 (0 + 1)^*$$

Here, the minimum string is 0101.

In the part of R i.e. $(0 + 1)^*$ either we can take any number of 0's or 1's including null string 'ϵ'.

Suppose, we want to generate 00101. Then, we can make $0^1 = 0$ from leftmost $(0 + 1)^*$. And then we can make $(0 + 1)^0 = \epsilon$ from rightmost $(0 + 1)^*$. And adding all the strings, we get 00101.

8. Define countably infinite and uncountable sets with example. Use principle of mathematical induction to prove $(5^n - 1)$ is divisible by 4 for all integers $n \geq 0$ **[2071 Chaitra]**

A set is said to be countably infinite set if its element can be put one-one correspondence with set of natural numbers. For example,

$$z = \{x : x \text{ is a element of integers}\}$$
$$= \{-3, -2, -1, 0, 1, 2, 3, \ldots\}$$

Here, we can't never count the cardinality of set S but if we arrange in such a way that

$$z = \{0, -1, 1, -2, 2, -3, 3, \ldots\}$$

And then we are asked to count number of elements up to –3, we can do that though it may take time for large number but it's possible so, the set of integers is countably infinite set.

A set is uncountable, if it contains so many elements that they can't be put one-one correspondence with the set of natural numbers. In other words, it is opposite to that of countably infinite set. For example, the set of real numbers in {0, 1}.

Prove by mathematical induction.

Let, P(n) be the statement i.e.

$P(n) = (5^n - 1)$ is divisble by 4 for all integers $n \geq 0$.

when $n = 0$

$P(0) = 5^0 - 1 = 1 - 1 = 0$ divisible by 4.

when $n = 1$

$P(1) = 5^1 - 1 = 5 - 1 = 4$ divisible by 4.

So, this theorem holds true for $n = 0, 1$. Let's suppose this theorem holds true for $n = m$. And if we could show that this theorem is ture for $n = m + 1$ when $n = m$ is true.

Then, it is true for all $n \geq 0$.

$P(m) = 5^m - 1$ divisible by 4 (say)

$$P(m+1) = 5^{m+1} - 1$$
$$= 5^m . 5 - 5 + 4$$
$$= 5 (5^m - 1) + 4 \quad \ldots\ldots\ldots \text{(i)}$$

Here, expression 1 holds true for $n = m + 1$ when $n = m$ is true. Hence, by the principle of mathematical induction, this theorem holds true for $n \geq 0$.

9. Explain what an equivalence relation is. Show by induction that for any $n \geq 0$ $1 + 2 + \ldots + n = \frac{(n^2 + n)}{2}$ [2072 Kartik]

A relation that is reflexive, symmetric and transitive is equivalence relation.

Reflexive relation: A binary relation R over a set A is reflexive if (a, a) ∈ A i.e. every element of X is related to itself. In mathematics, "is equal to" relation between real number is reflexive.

Symmetric relation: A binary relation over a set A i.e. $R \subseteq A \times A$ is symmetric if (b, a) ∈ R whenever (a, b) ∈ R. In mathematics "is equal to" relation between real numbers is symmetric relation.

Transitive relation: A binary relation on set A is transitive relation if a, b, c ∈ A and a is related to b and b is related to C then a is related to C. In mathematics "is greater than" or "is less than" relation between numbers is transitive relation.

Prove by mathematical induction

Let, P(n) be the statement

$$P(n): 1 + 2 + 3 + \ldots + n = \frac{n^2 + n}{2} \text{ for } n \geq 0.$$

when n = 0

$$\text{LHS } P(0) = 0$$

$$\text{RHS } P((0) = \frac{0 + 0}{2} = 0$$

when n = 1

$$\text{LHS } P(1) = 1$$

$$\text{RHS } P(1) = \frac{1 + 1}{2} = 1$$

So, this theorem holds true for n = 0, 1. Let's suppose this theorem holds true for n = m. And if we could show that this theorem is true for n = m + 1 when n = m is true. Then it is true for all $n \geq 0$.

$$p(m) = 1 + 2 + 3 + \ldots + m = \frac{m^2 + m}{2} = \frac{m(m+1)}{2}$$

For n = m + 1

$$P(m+1) = 1 + 2 + 3 + \ldots + m + (m+1)$$

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

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

$$= \frac{m^2 + m + 2m + 2}{2}$$

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

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

$$= \frac{(m+1)\{(m+1)+1\}}{2} \quad \ldots\ldots\ldots (i)$$

Here, expression 1 holds true for n = m + 1 when n = m is true. Hence, by the principle of mathematical induction, this theorem holds true for all $n \geq 0$.

10. State and explain pigeonhole principle with an example.

[2071 Shrawan Back]

Pigeonhole principle states that, "If A and B are finite sets and |A| > |B|, then there is no one-one function from A to B".

In other words, if we attempt to par off the elements of A ("the pigeons") with elements of B ("the pigeonholes"), sooner or later we will have to put more than one pigeon in a pigeonhole.

Example, Let R be a binary relation on a finite set A and let a, b ∈ A. If there is path from a to b in R, then there is a path of length at most (A). Prove it by pigeonhole principle.

Proof:

Suppose that $(a_1, a_2, \ldots, a_n)$ is the shortest path from $a_1 = a$ to $a_n = b$, that is, the path with the smallest length and suppose that n > (A). By the pigeonhole principle, there is an element of A that repeats on the path say $a_i = a_j$. For some $1 < i < j \leq n$. But then $(a_1, a_2, \ldots a_i, a_{j+1}, \ldots a_n)$ is a shorter path from a to b, contradicting our assumption that $(a_1, a_2, \ldots a_n)$ is the shorter path from a to b.

11. Prove that $\frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \ldots + \frac{1}{n(n+1)} = \frac{n}{n+1}$ using mathematical induction principle. **[2071 Shrawan, Back]**

Let P(n) be the statement

$$P(n): \frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \ldots + \frac{1}{n(n+1)} = \frac{n}{n+1}$$

when n = 1

$$\text{LHS: } P(1) = \frac{1}{2}$$

$$\text{RHS: } P(1) = \frac{1}{1+1}$$

$$= \frac{1}{2}$$

$\therefore$ LHS = RHS.

when n = 2

$$\text{LHS: } P(2) = \frac{1}{2} + \frac{1}{6}$$

$$= \frac{3+1}{6}$$

$$= \frac{4}{6}$$

$$= \frac{2}{3}$$

$$\text{RHS: } P(2) = \frac{2}{2+1}$$

$$= \frac{2}{3}$$

$\therefore$ LHS = RHS

So, this theorem holds true for n = 1, 2. Let's suppose this theorem holds true for n = m. And if we could show that this theorem is true for n = m + 1 when n = m is true. Then it is true for all $n \geq 0$.

$$P(m): \frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \frac{1}{m(m+1)} = \frac{m}{m+1}$$

when n = m + 1,

LHS: P(m + 1):

$$= \frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \ldots \frac{1}{m(m+1)} + \frac{1}{(m+1)(m+2)}$$

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

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

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

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

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

$$= \frac{(m+1)}{\{(m+1)+1\}} \quad \ldots\ldots\ldots \text{(i)}$$

So, expression 1 holds true for n = m + 1 when n = m is true. Hence, by the principle of mathematical induction, this theorem holds true for $n \geq 0$.

12. Justify that "The complement of diagonal set is different from each row sets" with the help of diagonalization principle. Show that if 3n + 2 is odd then n is odd by using proof by contradiction technique. [2070 Chaitra]

Let R be a binary relation on a set A and let D = {a | a ∈ A and (a, a) ∉ R}. For each a ∈ A, let Ra = {b | b ∈ A and (a, b) ∈ R}. Then D is distinct from Ra for all a ∈ A

Proof:

Let A = {a, b, c, d} and R = {(a, b), (a, d), (b, b), (b, c), (c, c), (d, b)}. R can be represented as a square array as below:

		a	b	c	d
Ra →	a		×		×
Rb →	b		×	×	
Rc →	c			×	
Rd →	d		×		

Here, Ra = {b, d}, Rb = {b, c}

Rc = {c}, Rd = {b}

The complement of diagonal set D = {a, d} clearly D Ra, Rb, Rc, Rd. Hence, the complement of diagonal set is different from each row sets i.e. Ra, Rb, Rc and Rd.

We prove if 3n + 2 is odd then n is odd by contradiction technique.

Let's assume that n is even for which 3n + 2 becomes odd.

Now, we know that to get sum of two numbers even either both of the addendum must be even or odd. Here in the expression 3n + 2, 2 is already even and if we assume n is even then even multiples of n results in number is always even i.e. 3n + 2 is even which contradicts our assumption n is even for which 3n + 2 is odd. So, n must be odd. Hence, proved

13. Let Σ = {a, b}. Write a regular expression for the language with all strings in Σ* with no more than 3 a's. [2072 Kartik, Back]

Let R be the required regular expression for the language with no more than 3 a's which generates following strings

L = {b, bb, ∈, ab, abb, abba, aaabb, ...}

Then

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

♦.♦♦

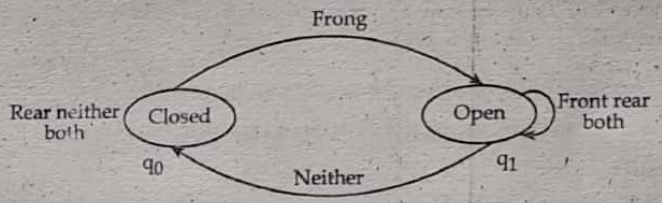
Chapter-2

Finite Automata

The characteristics of finite automata are as follows:

- An automata is a abstract model of digital computer.
- Mechanism to read input, string over a given alphabet, written on an input file which can be read but can't change it.
- Input file is divided into cells, each can hold one symbols.
- Has temporary storage device and has control unit, finite control can sense what symbol is written at any position in the input tape by movable reading head.
- Initially reading head is placed at the left most of the tape, moves through right and indicates it's approval or disapproval.

A state diagram for an automatic controlled door.



State transition table for an automatic door controller.

Transition→ / ↓State	Neither	Front	Rear	Both
Closed	Closed	Open	Closed	Closed
Open	Closed	Open	Open	Open

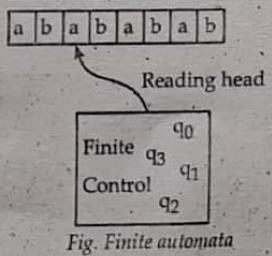


Fig. Finite automata

Formal definition of DFA

A deterministic finite automata is a quintuple or 5-tuple.

$M = (\theta, \Sigma, \delta, q_o, F)$

where,

θ = fintie set of internal states

Σ = finite sets of symbols called alphabets

$\delta = \theta \times \Sigma \rightarrow \theta$: Transition function.

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

$q_0 \rightarrow$ initial states.

$F \subseteq \theta$ = set of final states

Characteristics of deterministic finite automata

The main characteristics of DFA are as follows:

(a) It is the simplest model of computation.

(b) It has a very limited memory.

(c) For a single input there is only one transition state.

Examples

1. **Design a DFA that accepts a language such that,**

$L(m) = \{w : w \in \{a, b\}^*$, w has even number of b's$\}$

Solution:

Let, the required DFA be

$M = (\theta, \Sigma, \delta, q_o, F)$

where,

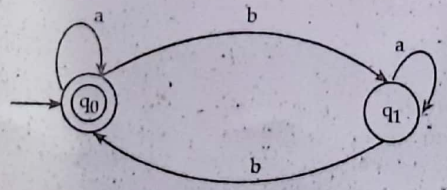
$\theta = \{q_0, q_1\}$

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

$q_o = \{q_o\}$

$F = \{q_o\}$

State diagram



δ is the transition function tabulated.

$\therefore$ Transition table:

↓States \ $\Sigma \rightarrow$	a	b
$\rightarrow$ $*q_0$	q_0	q_1
q_1	q_1	q_o

Note: $\rightarrow$ Represents the starting state

* Represents the final states.

Test for string: 'ba ab'

$(q_0, b\ aab) \vdash M(q_1, aab)$

$\vdash M(q_1, ab)$

$\vdash M(q_1, b)$

$\vdash M(q_0, \in)$ Accepted

Since on empty string it enters to final state q_0, it is accepted.

Test for string – 'baa'

$(q_0, b\ aa) \vdash M(q_1, aa)$

$\vdash M(q_1, a)$

$\vdash M(q_1, \in)$ Not accepted

Since, on empty string it enters to non final state, so string 'baa' is not accepted.

2. **Design a DFA that accepts language $L(M) = \{w \in \{a, b\}^*$, w doesn't contain only 3 consecutive b'S$\}$.**

Solution:

Note: Problem like 'not containing particular string' is much tougher than 'containing string'. So first we would solve 'containing string' and then we should make it 'not containing string'.

Changed questions is

$L(M) = \{w \in \{a, b\}^*$, w contains any 3 consecutive b'S$\}$

Let, required DFA be

$L(M) = (\theta, \Sigma, \delta, q_o, F)$

where,

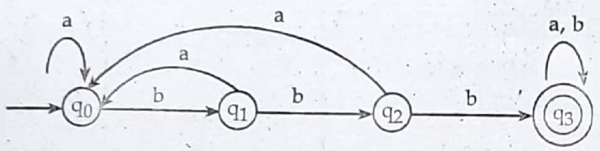
$\theta = \{q_0, q_1, q_2, q_3\}$

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

$q_0 = \{q_0\}$

$F = \{q_3\}$

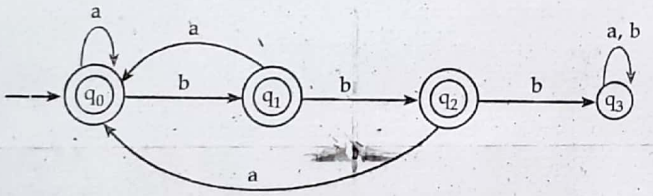
State diagram for w containing 3 consecutive b's



Note: On feeding input a to state q_2, it goes to state q_0 directly not q_1 because for string like bb abb, final state will be q_3 which must be accepted by DFA which is not correct according to our charged question.

This is the state diagram for w containing only 3 consecutive b'S.

For state diagram for w not containing any 3 consecutive b'S, all we have to do is to interchange between non-final states and final states, i.e.



Here, q_3 is dead state to reject the consecutive b'S.

Transition table (δ):

States/Σ	a	b
→ *q_0	q_0	q_1
* q_1	q_0	q_2
* q_2	q_0	q_3
q_3	q_3	q_3

Here for q_0, q_1 is now new state on feeding b to q_1, so q_1 is new state in transition table.

Test for string: 'ab bab'

$(q_0, abb\ ab) \vdash M(q_0, bb\ ab)$

$\vdash M(q_1, b\ ab)$

$\vdash M(q_2, ab)$

$\vdash M(q_0, b)$

$\vdash M(q_1, \in)$ Accepted.

Test for string: 'abbb'

$(q_0, abbb) \vdash M(q_0, bbb)$

$\vdash M(q_1, bb)$

$\vdash M(q_2, b)$

$\vdash M(q_3, \in)$ Not accepted.

Here string 'abbb' is not accepted as its final state is q_3 which is dead final state instead of final state.

3. Design a DAF that accepts language given by,

L(M) = {w ∈ {0, 1}*, that accepts all strings starting with 0}

Solution:

Let the required DAF be

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

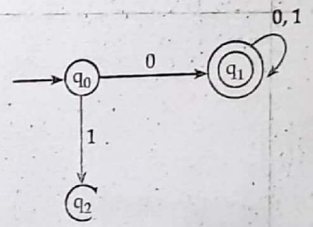
$\theta = \{q_0, q_1, q_2\}$

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

$q_0 = \{q_0\}$

$F = \{q_1\}$

State diagram



Here, q_2 is dead state or t

Transition able

States/Σ
→ q_0
* q_1
q_2

Test for string '001'

$\delta(q_0, 001) \vdash$

$\vdash$

$\vdash$

Test for string '10

$\delta(q_0, 10)$ |

5. Co

Solution:

Let, the require

$L(M) = (\theta, \Sigma, \delta,$

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

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

$q_0 = \{q_0\}$

$F = \{q_1\}$

Non-determin
Finite
Automata
(NFA)

ata
FA)

4. Construct DFA that accepts sets of all strings over {0, 1} of length 2.

Solution:

Let the required DFA be,

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

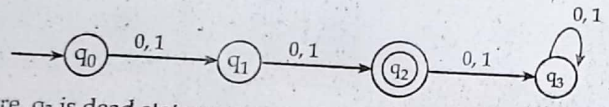
$\theta = \{q_0, q_1, q_2, q_3\}$

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

$q_0 = \{q_0\}$

$F = \{q_2\}$

State diagram



Here, q_3 is dead state or trap state.

Transition able:

States/Σ	0	1
→ q_0	q_1	q_2
q_1	q_2	q_1
* q_2	q_3	q_3

Test for string: '00'

$(q_0, 00) \vdash\!\!\!\!-\!\!\!\!\vdash M(q_1, 0)$

$\vdash M(q_2, \in)$ Accepted

Test for string: '001'

$(q_0, 001) \vdash M(q_1, 01)$

$\vdash M(q_2, 1)$

$\vdash M(q_3, \in)$ Not accepted

5. Construct a DFA that accepts the language defined as:

$L(M) = \{a^n b : n \geq 0\}$

Solution:

Let, the required DFA be

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

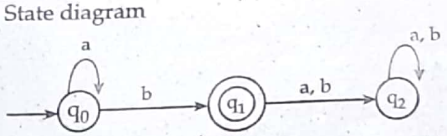
$\theta = \{q_0, q_1, q_2\}$

$\Sigma = \{a_0, b\}$

$q_0 = \{q_0\}$

$F = \{q_1\}$

State diagram



Here, q_2 is dead state.

Transition able:

States/Σ	a	b
→ q_0	q_0	q_1
* q_1	q_2	q_2
q_2	q_2	q_2

When n = 0, then L = {b}, this string should be accepted by finite automata.

Test for string: 'b'

$(q_0, b) \vdash\!\!\!\!-\!\!\!\!\vdash M(q_1, \in)$ Accepted

When n = 1, then string L is 'ab' and should be accepted by finite automata.

$\delta(q_0, ab) \vdash\!\!\!\!-\!\!\!\!\vdash M(q_0, b)$

$\vdash\!\!\!\!-\!\!\!\!\vdash M(q_1, \in)$ Accepted.

Test of string: 'aba'

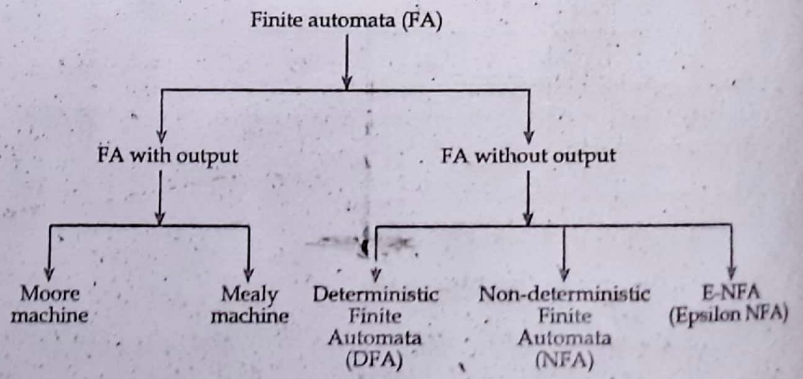
$\delta(q_0, aba) \vdash\!\!\!\!-\!\!\!\!\vdash M(q_0, ba)$

$\vdash\!\!\!\!-\!\!\!\!\vdash M(q_1, a)$

$\vdash\!\!\!\!-\!\!\!\!\vdash M(q_2, \in)$ Not accepted

It is not accepted since q_2 is not a final state

The general classification of finite automata can be described as



Here, the purpose of reading finite automata are as follows:

(a) It indicates wether a string contains a particular substring or not. For example, wether any of the strings air, water, earth, and fire occur in the text of elements of the 'Theory of Computation'.

(b) In computing a program the first step is lexical analysis. This is dates keyword, identifiers, etc while eliminating, irrelevant symbols. And finite state machine i.e. finite automata helps in lexical analysis.

(c) It is applicable to the design of several common types of computer algorithms and programs.

(d) It is used in scanning the text for various purposes like matching, detecting substrings within string etc.

Here, we should mainly focus on deterministic finite automata (DFA) and non-deterministic finite automata (NFA).

Deterministic finite automata

Deterministic finite automata is a finite state machine that accepts or rejects strings of symbols and only produces a unique computation of the automation of each input string. It is also known as deterministic finite automata. A DFA can model sotware that decides wether or not online user input such as email addresses are valid.

Deterministic finite automata posses

Some following characteristics:

(a) Given the current state we know what next state will be

(b) It has only one unique next state for every input.

(c) If has no multiple choices or randomness.

(d) It is simple and easy to design.

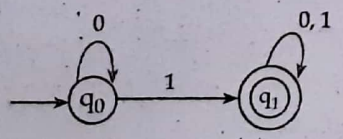


Fig. (i)

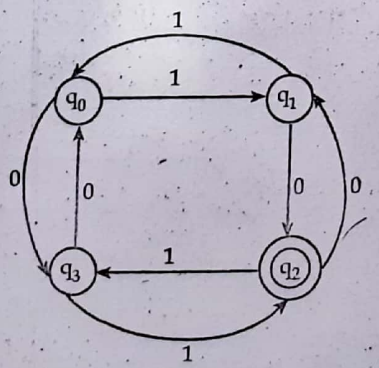


Fig. (ii)

Non-deterministic finite automatic (NFA)

In automata theory a finite state machine is called a deterministic finite automata (DFA) if and only if:

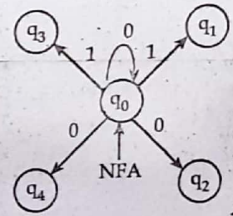
(a) each of its transition is uniquely determined by its source state and input symbol and

(b) reading an input symbol required for each state transition.

But a NFA doesn't need to follow these principle. In fact every DFA is a NFA but not every NFA is a DFA. But there may exists an equivalent DFA for every NFA.

Informal definition of non-deterministic finite automata

An NFA similar to DFA consumes a string of input symbol; for each input symbol it may transit nowhere or may transit to another new state or may transit to two or more new states simultaneously. Thus in the formal definition, the next state is an element of the power set of the states, which is a set of states to be considered at once i.e. $\delta : \theta \times \Sigma \to 2^{\theta}$. The notion of accepting an input is similar to that for the DFA. When the last input symbol is considered the NFA accepts if and only if there is some set of transitions that will take it to an accepting state. Equivalently if rejects no mater what transitions are applied it would not end in accepting state.



Formal definition of NFA

Like DFA, NFA (non-deterministic finite automata) is also a quintuple or 5 tuple.

$M \to (\theta, \Sigma, \delta, q_0, F)$

θ = finite set of internal states

Σ = finite set of symbols called alphabets.

δ : $\theta \times \Sigma \to 2^{\theta}$ transition state

q_0 = input state

$F \subseteq \theta$ = set of final states

The characteristics of non-deterministic finite automata are as follows:

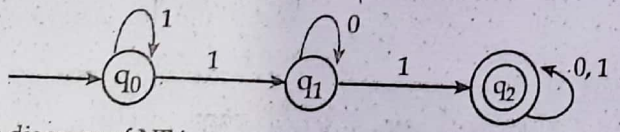
- Given the current state, there could be multiple states or no states.
- A NFA has ability to be in several states at once.
- Transitions from a state on an input symbol can be to any set of states.

- It starts in one state.
- It accepts only if any sequence of choice leads to a final state.
- Intuitively, the 'non deterministic' always "guesses right".

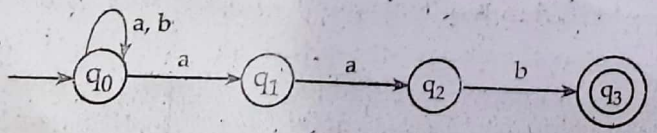
Difference between NFA and DFA

NFA	DFA
(a) In DFA, the transition function (δ) is $\delta : \theta \times \Sigma \to \theta$ (set all possible subsets)	(a) In NFA, the transition function (δ) is $\delta : \theta \times \Sigma \to 2^{\theta}$ (set of power set of possible subsets)
(b) There is always one new state or choked state for every single input.	(b) There may be new state or set of multiple states or chocked states for every single input.
(c) The can't be multiple initial state.	(c) There can be multiple initial states i.e. $\in$ NFA.
(d) DFA accepts if final state of automata is an acceptor state.	(d) NFA accepts only if any of final states obtained is an acceptor state.
(e) It is less efficient than NFA.	(e) It is much more efficient than DFA.
(f) Every DFA can be converted into NFA.	(f) There may exists an equivalent DFA for NFA.
(g) Example, checking sub-string into main string is an example of DFA.	(g) Example, moves on a chessborad is an example of NFA.

State diagram of DFA



State diagram of NFA



Examples

1. **Design a NFA for the language L = all strings are {0, 1} that have at least two consecutive 0'S or 1'S.**

Solution:

Let the required NFA be,

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

where,

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

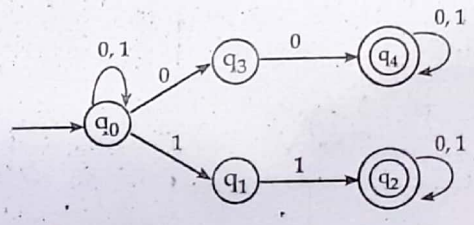
$\Sigma = \{0, 1\}$

$\delta = \theta \times \Sigma \to 2^{\theta}$

$q_0 = \{q_0\}$

$F = \{q_2, q_4\}$

State diagram



As q_3 and q_1 has no transition states for input 1 and 0 respectively, that's what makes it non-deterministic finite automata (NFA). Also for input 0 and 1, to q_0, there are two possibilities:

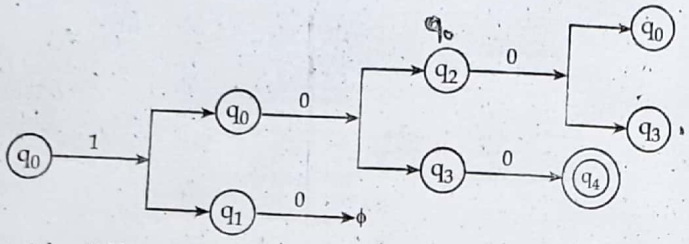
(a) For input 0, it may either go to q_3 or remain at q_0.

(b) For input 1 it may either go to q_1 or remain at q_0.

Transition table:

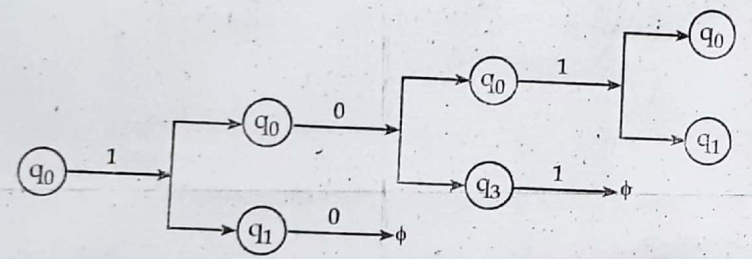
States/Σ	a	b
$\to$ q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_3	q_4	ϕ
* q_4	q_4	q_4
q_1	ϕ	q_2
* q_2	q_2	q_2

Test for '100'



So the string '100' will be accepted by this NFA as in the set of final states $\{q_2, q_3, q_4 \text{ and } \phi\}$, a_4 is acceptor state.

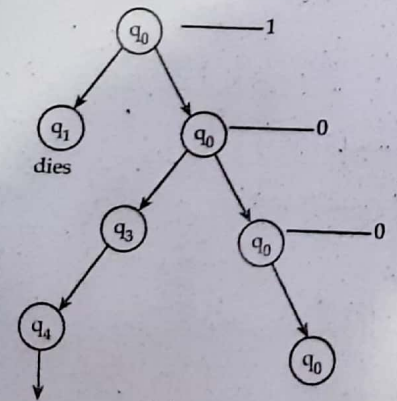
Test for '101'



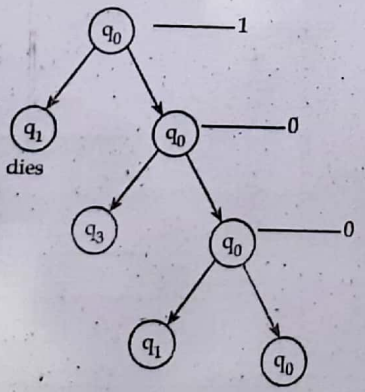
This string '101' will not be accepted by NFA as none of the final states i.e. $\{q_0, q_1, \phi, \phi\}$ is acceptor state.

When does a NFA accepts?

If there is any way to run the machine that ends in any set of final states out of which at least one of state is q final state, then NFA accepts



Accepting path for string '100'



No accepting path

2. **Construct a NFA that accepts set of all strings that start with 0.**

L = {set of all strings starting with 0} = {0, 01, 001, 011, ...}

Solution:

Let the required NFA be,

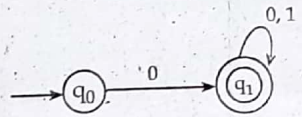
$L(M) = (\theta, \Sigma, \delta, q_0, F)$

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

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

$q_0 = \{q_0\}$

$F = \{q_1\}$



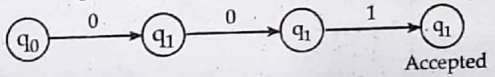
State diagram

This is NFA as state q_0, has no output for input '1'.

Transition table:

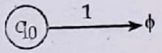
States/Σ	0	1
→ q_0	q_1	ϕ
* q_1	q_1	q_1

Test for string '001'



It is accepted since q_1 is accepting sate.

Testing for string: '101'



[Dead configuration or trapped configuration]

Since, the final state is not acceptor state it is not accepted.

3. **Design a NFA that accepts set of all strings that ends with '0'.**

L = {set of all strings ending at '0'}

L = {10, 011, 00, ...

Solution:

Let the required NFA be,

$L = (\theta, \Sigma, \delta, q_0, F)$

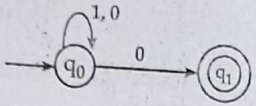
$\theta = \{q_0, q_1\}$

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

$q_0 = \{q_0\}$

$F = \{q_1\}$

State diagram

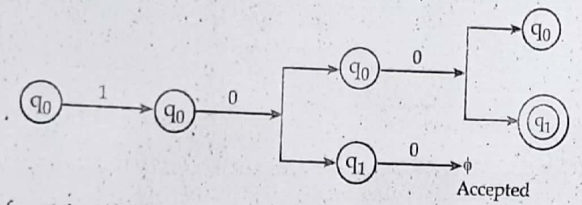


Here, for input '0' to state q_0, there are two possibilities q_0 and q_1 state. And there are no outputs for states q_1, when any of the inputs is applied. That is what makes it NFA.

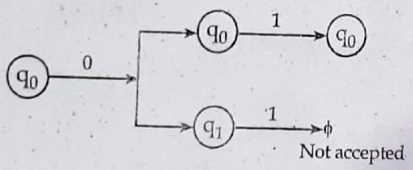
Transition table

States/Σ	0	1
→ q_0	$\{q_0, q_1\}$	q_0
* q_1	ϕ	ϕ

Test for string '100'



Test for string '01'



[*Note:* Finite automata doesn't produce output]

4. Design a NFA that accepts sets of all strings over {0, 1} of length 2.

Solution:

Let the required NFA be,

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

where,

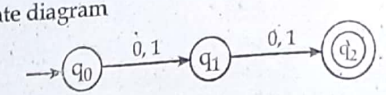
$\theta = \{q_0, q_1, q_2\}$

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

$q_0 = \{q_0\}$

$F = \{q_2\}$

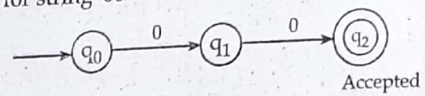
State diagram



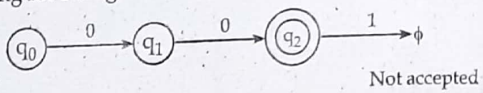
Transition table:

States/Σ	0	1
→ q_0	q_1	q_1
q_1	q_2	q_2
* q_2	ϕ	ϕ

Test for string '00'



Testing for string: '001'



5. Design a NFA that accepts strings containing '01'.

Solution:

Let the required NFA be,

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

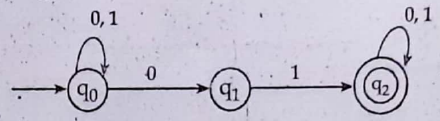
$\theta = \{q_0, q_1, q_2\}$

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

$q_0 = \{q_0\}$

$F = \{q_2\}$

State diagram

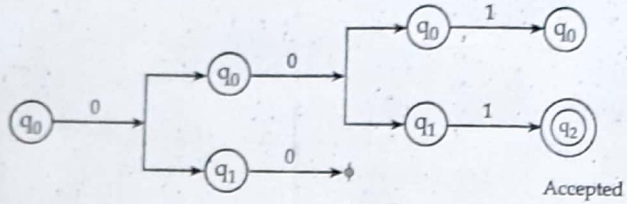


Here, q_1 has no transition state for input '0' to its. so, it is NFA.

Transition table:

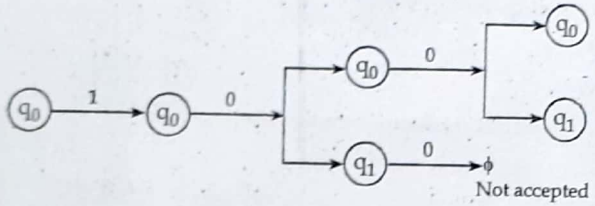
States/Σ	0	1
→ q_0	q_0	$\{q_0, q_1\}$
q_1	ϕ	q_2
* q_2	q_2	q_2

Testing for string '001'



Accepted

Testing for string: '100'



Not accepted

6. Design a NFA that accepts string ending at '11'.

Solution:

Let the required NFA be,

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

where,

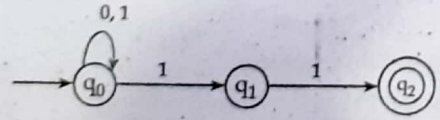
$\theta = \{q_0, q_1, q_2\}$

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

$q_0 = \{q_0\}$

$F = \{q_2\}$

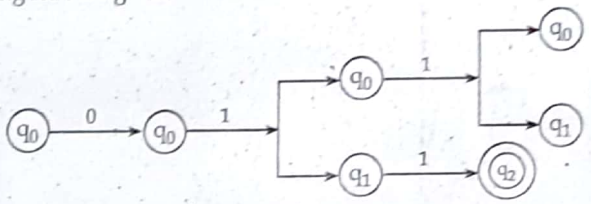
State diagram



Transition table:

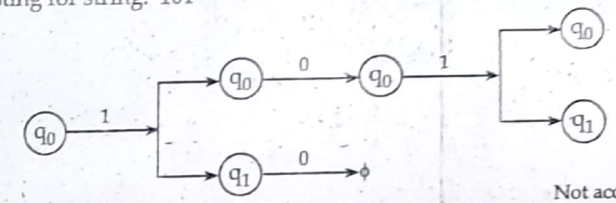
States \ Σ	0	1
→ q_0	q_0	$\{q_0, q_1\}$
q_1	ϕ	q_2
* q_2	ϕ	ϕ

Testing for string '011'



Accepted

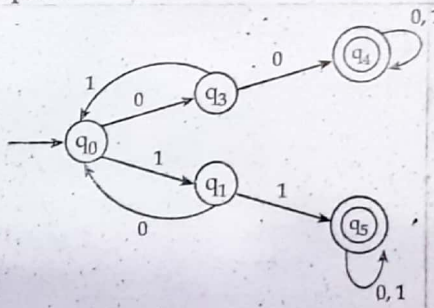
Testing for string: '101'



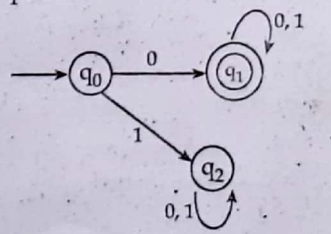
Not accepted

Now, equivalent DFAs for above examples of NFAs

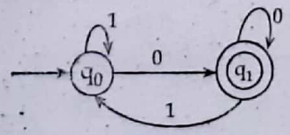
Example: 1



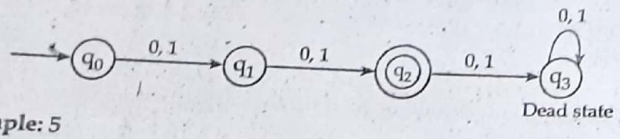
Example: 2



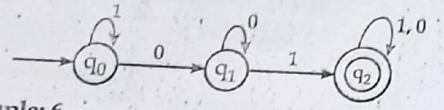
Example: 3



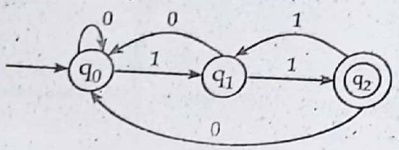
Example: 4



Example: 5



Example: 6



7. **Draw the state diagram for NFA. L = (ab)* (ba)* ∪ (aa)***

Solution:

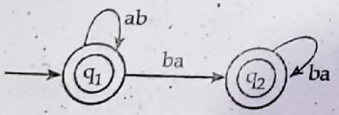
Given, L = (ab)* (ba)* ∪ (aa)* ..... (i)

Comparing (i) with L = $L_1 \cup L_2$

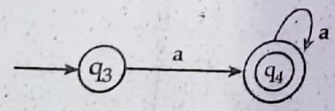
L_1 = (ab)* (ba)*

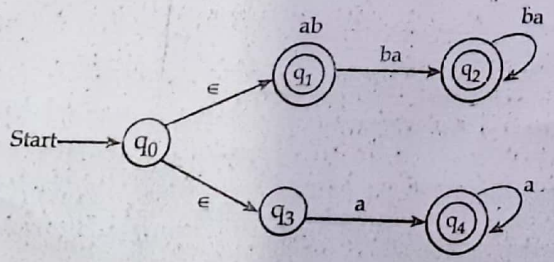
L_2 = (aa)*

For L_1,



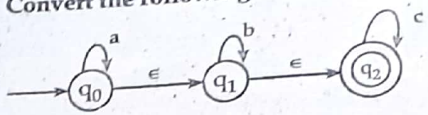
For L_2,





ϵ – NFA

1. **Convert the following ϵ – NFA to DFA.**



Solution:

Here, any input or even number input can lead from q_0 to q_2. And we call it as a e-closure (q_0).

Let, e-closure (q_0)

= {q_0, q_1, q_2}

= A (say)

Now,

Transition states for each input

Step: – I

δ(A, a) = e-closure (δ (q_0, q_1, q_2), a)

= e-closure (δ (q_0, a) ∪ δ(q_1, a) ∪ δ(q_2, a))

= e-closure ($q_0 \cup \phi \cup \phi$)

= e-closure (q_0)

= A

δ(A, b) = e-closure (δ (q_0, q_1, q_2), b)

= e-closure (δ (q_0, b) ∪ δ(q_1, b) ∪ δ(q_2, b))

= e-closure ($\phi \cup q_1 \cup \phi$)

= e-closure (q_1)

= {q_1, q_2}

= B

∴ [Let, e-closure (q_1) = {q_1, q_2} = B]

δ(A, c) = e-closure (δ (q_0, q_1, q_2), c)

= e-closure (δ (q_0, c) ∪ δ(q_1, c) ∪ δ(q_2, c)

= e-closure ($\phi \cup \phi \cup q_2$)

= e-closure (q_2)

= {q_2}

= q_c (say)

Now, we have new transition states as B and q_c.

Step – II:

$\therefore \delta(B, a)$ = e-closure $(\delta(q_1, q_2), a)$

= e-closure $(\delta(q_1, a) \cup \delta(q_1, a))$

= e-closure $(\phi \cup \phi)$

= e-closure (ϕ)

= D (say)

$\delta(B, b)$ = e-closure $(\delta(q_1, q_2), b)$

= e-closure $(\delta(q_1, b) \cup \delta(q_2, b))$

= e-closure $(q_1 \cup \phi)$

= e-closure (q_1)

= B

$\therefore \delta(B, c)$ = e-closure $(\delta(q_1, q_2), c)$

= e-closure $(\delta(q_1, c) \cup \delta(q_2, c))$

= e-closure $(\phi \cup q_2)$

= e-closure (q_2)

= $\{q_2\}$

= q_c

Similarly,

Step - III

$\delta(q_c, a)$ = e-closure $(\delta(q_2, a)) = \phi$ = D

$\delta(q_c, b)$ = e-closure $(\delta(q_2, b))$

= e-closure (ϕ)

= D

$\delta(q_c, c)$ = e-closure $(\delta(q_2, c)$

= e-closure (q_2)

= $\{q_2\}$

= q_c

And for dead state D

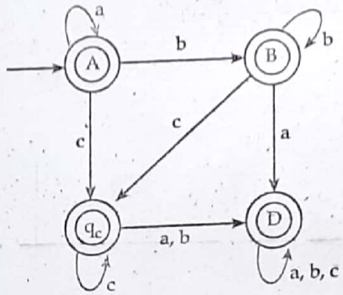
Step - IV

$\delta(D, a) = \delta(D, b) = \delta(D, c)$ = D

Transition table:

States/Σ	a	b	c
→ *A	A	B	q_c
*B	D	B	q_c
*q_c	D	D	q_c
D	D	D	D

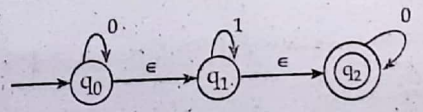
State diagram:



Here, the final states are A, B and q_c as they contain q_2 (previous final state).

$\therefore$ This is the required DFA.

2. Convert the following ∈-NFA to DFA.



M = (θ, Σ, δ, q_0, F)

Solution:

θ = e-closure (q_0), e-closure (q_1), e-closure $(q_2$

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

Let,

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

$\{q_1, q_2\}$ = B

$\{q_2\}$ = C

Transition states:

Step – I

$\delta(A, 0)$ = e-closure $(\delta(q_0, q_1, q_2), 0)$
= e-closure $(\delta(q_0, 0) \cup \delta(q_1, 0) \cup (q_2, 0))$
= e-closure $(q_0 \cup \phi \cup q_2)$
= e-closure (q_0) ∪ e-closure (q_2)
= $\{q_0, q_1, q_2\} \cup \{q_2\}$
= $\{q_0, q_1, q_2\}$
= A

$\delta(A, 1)$ = e-closure $(\delta(q_0, q_1, q_2), 1)$
= e-closure $(\delta(q_0, 1) \cup \delta(q_1, 1) \cup (q_2, 1))$
= e-closure $(\phi \cup q_1 \cup \phi)$
= e-closure (q_1)
= $\{q_1, q_2\}$
= B

Similarly,

Step – II

$\delta(B, 0) = C$
$\delta(B, 1) = B$

Step – III

$\delta(C, 0) = C$
$\delta(C, 1) = D$ (dead state)

Step – IV

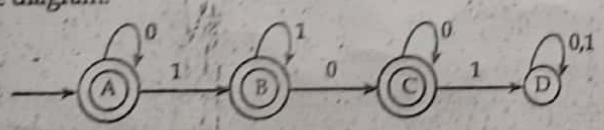
$\delta(D, 0) = D$
$\delta(D, 1) = D$

Note: Readers are requested to solve these themselves.

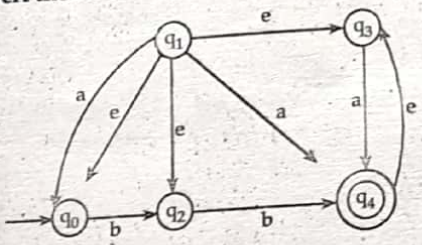
Transition table

States/Σ	0	1
→ *A	A	B
*B	C	B
*C	C	D
D	D	D

State diagram:



3. Convert the following ∈ - NFA to DFA.



Solution:

Let, e-closure $(q_0) = \{q_0, q_1, q_2, q_3\} = A$
e-closure $(q_4) = \{q_3, q_4\}$
e-closure $(q_2) = \{q_2\}$
e-closure $(q_1) = \{q_2, q_3\}$
e-closure $(q_3) = \{q_3\}$

Step - I

$\delta(A, a)$ = e-closure $(\delta(q_0, q_1, q_2, q_3)\ a)$
= e-closure $(\delta(q_0, a) \cup \delta(q_1, a) \cup (q_2, a) \cup \delta(q_3, a))$
= e-closure $(\phi \cup q_0 \cup q_4)$
= e-closure (q_0) ∪ e-closure (q_4)
= $\{q_0, q_1, q_2, q_3\} \cup \{q_3, q_4\}$
= $\{q_0, q_1, q_2, q_3, q_4\}$
= B (say)

$\delta(A, b)$ = e-closure $(\delta(q_0, q_1, q_2, q_3)\ b)$
= e-closure $(\delta(q_0, b) \cup \delta(q_1, b) \cup (q_2, b) \cup \delta(q_3, b))$
= e-closure $(q_2 \cup \phi \cup q_4 \cup \phi)$
= e-closure (q_2) ∪ e-closure (q_4)
= $\{q_2\} \cup \{q_3, q_4\}$
= $\{q_2, q_3, q_4\}$
= C (say)

We have, next state B = $\{q_0, q_1, q_2, q_3, q_4\}$

Step – II

$\delta(B, a)$ = e-closure $(\delta(q_0, q_1, q_2, q_3, q_4), a)$
= e-closure $(\delta(q_0, a) \cup \delta(q_1, a) \cup (q_2, a) \cup \delta(q_3, a) \cup \delta(q_4, a))$
= e-closure $(\phi \cup q_0 \cup \phi \cup q_4 \cup \phi)$

= e-closure (q_0) ∪ e-closure (q_4)

= {q_0, q_1, q_2, q_3} ∪ {q_3, q_4}

= {q_0, q_1, q_3, q_4}

= B

δ (B, b) = e-closure (δ (q_0, q_1, q_2, q_3, q_4), b)

= e-closure (δ (q_0, b) ∪ δ(q_1, b) ∪ (q_2, b) ∪ δ(q_3, b)∪δ(q_4, b))

= e-closure (q_2 ∪ ϕ ∪ q_4 ∪ ϕ ∪ ϕ)

= e-closure (q_2) ∪ e-closure (q_4)

= {q_2} ∪ {q_3, q_4}

= {q_2, q_3, q_4}

= C (say)

Step – III

δ (C, a) = e-closure (δ (q_2, q_3, q_4), a)

= e-closure (δ (q_2, a) ∪ δ(q_3, a) ∪ δ(q_4, a))

= e-closure (ϕ ∪ q_4 ∪ ϕ)

= {q_3, q_4}

= D (say)

δ (C, b) = e-closure (δ (q_2, q_3, q_4), b)

= e-closure (δ (q_2, b) ∪ δ(q_3, b) ∪ δ(q_4, b))

= e-closure (q_4 ∪ ϕ ∪ ϕ)

= {q_3, q_4}

= D

Step – IV

δ (D, a) = e-closure (δ (q_3, q_4), a)

= e-closure (δ(q_3, a) ∪ δ(q_4, a))

= e-closure (q_4 ∪ ϕ)

= e-closure (q_4)

= {q_3, q_4}

= D

δ (D, b) = e-closure (δ (q_3, q_4), b)

= e-closure (δ(q_3, b) ∪ δ(q_4, b))

= e-closure (ϕ ∪ ϕ)

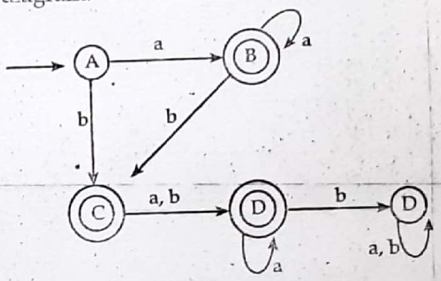
= E (dead state)

Step – V

δ (E, a) = δ (E, b) = E

States/Σ	a	b
→ *A	B	C
*B	B	C
*C	D	D
*D	D	E
E	E	E

State diagram:



A = {q_0, q_1, q_2, q_3}

B = {q_0, q_1, q_2, q_3, q_4}

C = {q_2, q_3, q_4}

D = {q_3, q_4}

This is the required DFA

Conversions of NFA to DFA

Lets start with easy example.

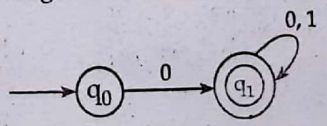
1. **Construct a NFA that accepts set of all strings over {0, 1} that starts with '0'}**

Solution:

L(M) = {set of all strings over {0, 1} that starts with '0'}

Here, NFA goes like this.

State diagram:



Transition table:

States/Σ	0	1
→ q_0	q_1	ϕ
* q_1	q_1	q_1

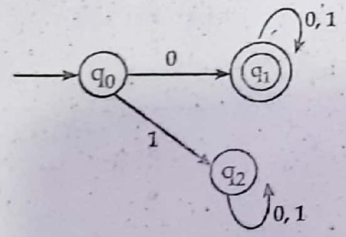
Here, an input '1' to state q_0 it goes nowhere. Now, for changing this NFA to DFA we have to solve this condition. For this we should introduce a new state q_2 called dead state.

So, the equivalent DFA's would look like.

Transition table for DFA:

States/Σ	0	1
→ q_0	q_1	q_2
* q_1	q_1	q_1
q_2	q_2	q_2

And finally the state diagram is



This is the equivalent DFA for above NFA.

2. Design NFA that accepts the set of all strings over {0, 1} that ends with '1'. And then design its equivalent DFA.

Solution:

Let, the required NFA be,

$$L(M) = (\theta, \Sigma, \delta, q_0, F)$$

where,

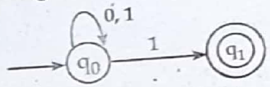
$\theta = \{q_0, q_1\}$

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

$q_0 = \{q_0\}$

$F = \{q_1\}$

State diagram



Transition table:

States/Σ	0	1
→ q_0	q_0	$\{q_0, q_1\}$
q_1	ϕ	ϕ

Here, an input '1' to the state q_0 it goes to both the states q_0 and q_1 which is non-deterministic. To make it deterministic we would introduce a new state q_a which denotes both state q_0 and q_1.

i.e. $\{q_0, q_1\} = q_a$

Now, the transition table for deterministic finite automata.

States/Σ	0	1
→ q_0	q_0	q_a
* q_a	q_0	q_a

On input '0' to new state q_a which is combination of q_0 and q_1, the transition state is union of final state q_0 and q_1. i.e.

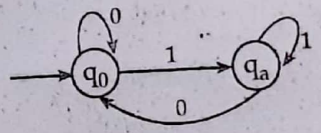
$$\begin{aligned}\delta(q_a, 0) &= \delta(\{q_0, q_1\}, 0)\\ &= \delta(q_0, 0) \cup \delta(q_1, 0)\\ &= q_0 \cup \phi \qquad \text{[From table of FNA]}\\ &= q_0\end{aligned}$$

Similarly, on input '1' to new state 'q_a' the transition state is:

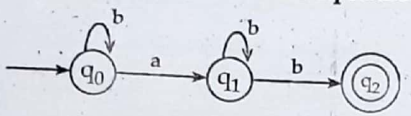
$$\begin{aligned}\delta(q_a, 1) &= \delta(\{q_0, q_1\}, 1)\\ &= \delta(q_0, 1) \cup \delta(q_1, 1)\\ &= \{q_0, q_1\} \cup \phi\\ &= \{q_0, q_1\}\\ &= q_a\end{aligned}$$

Also, on feeding any of the inputs to q_a, no new state is obtained. So, transition table stops. Equivalent DFA's.

State diagram is:



3. Convert the following NFA into equivalent DFA.



Solution:

Drawing equivalent transition table of above question.

States/Σ	a	b
q_0	q_1	q_0
q_1	ϕ	$\{q_1, q_2\}$

Initial state is $\{q_0\}$

Step - I

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

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

Step - II

For new state q_1

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

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

Step - III

For new stat $\{q_1, q_2\}$

$\delta\ ((q_1, q_2], a) = \phi(q_1, a) \cup \delta(q_2, a)$

$= \phi \cup \phi$

$= \phi$

$\delta\ ((q_1, q_2], ba) = \phi(q_1, b) \cup \delta(q_2, b)$

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

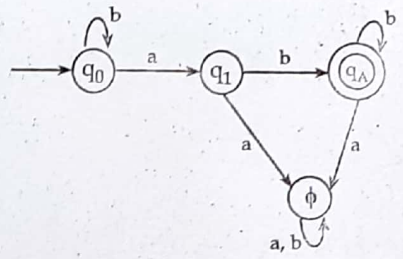
$= \{q_1, q_2\}$

Let $\{q_1, q_2\} = q_A$

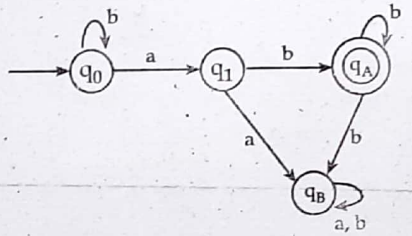
Then, the transition table for equivalent DFA is,

States/Σ	a	b
→ q_0	q_1	q_0
q_1	ϕ	q_A
*q_A	ϕ	q_A

State diagram of equivalent DFA is,



If in DFA there is null state, i.e. ϕ then describe it as shown above or you can call it a dead state and name whatever you like. Then diagram is



Here q_B is dead state.

4. Find the equivalent DFA for the NFA given by M = [{q_0, q_1, q_2}, {a, b}, δ, q_0, {q_2}]

where δ is given by:

States/Σ	a	b
→ q_0	$\{q_1, q_1\}$	q_2
q_1	q_0	q_1
*q_2	–	$\{q_0, q_1\}$

Solution:

Given, M = [{q_0, q_1, q_2}, {a, b}, δ, q_0, {q_2}] ..... (i)

Comparing (i) with

M = (θ, Σ, δ, q_0, F)

where,

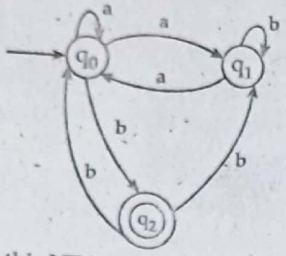
θ = {q_0, q_1, q_2}

Σ = {a, b}

q_0 = {q_0}

F = {q_2}

Drawing state diagram fro given transition table (δ)



Now, this NFA is to be converted into DFA.

Step – I

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

$\delta(q_0, b) = q_2$ = new state

Let, $\{q_0, q_1\} = q_A$

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

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

$= q_A \cup q_0$ [From given table]

$= q_A$

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

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

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

$= \{q_1, q_2\}$

$= q_B$ (say) = new state

Now,

Step – II

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

$= \delta(q_1, a) \cup \delta(q_2, a)$

$= \{q_0\} \cup \phi$

$= q_0$

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

$= \delta(q_1, b) \cup \delta(q_{21}, b)$

$= \{q_1\} \cup \{q_A\}$

$= q_A$

Step – III

$\delta(q_2, a) = \phi = q_c$ (say)

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

$= q_A$

Step – IV

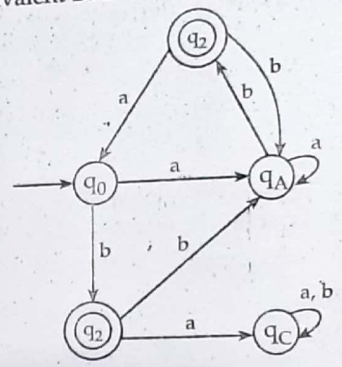
$\delta(q_C, a) = q_C$

$\delta(q_C, b) = q_C$

Now, transition table of equivalent DFA, we get

States/Σ	a	b
→ q_0	q_A	q_2
q_A	q_A	q_2
* q_B	q_0	q_A
* q_2	q_C	q_A
q_C	q_C	q_C

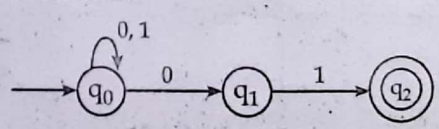
Equivalent DFA for NFA is



Note: Here q_2 is already final state as given in the question and q_B contains state q_2. So, it is also a final state.

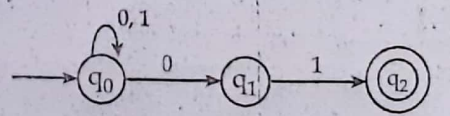
5. **Given below is the NFA for a language:**

L = {set of all strings over {0, 1} that ends with '01'}. Construct its equivalent DFA.



Solution:

Given, NFA is



The transition table of above state diagram is

States/Σ	a	b
→ · q_0	$\{q_0, q_1\}$	q_0
q_1	ϕ	q_2
* q_2	ϕ	ϕ

Step – I

$\delta(q_0, 0) = \{q_0, q_1\} = q_A$ (say) = new state

$\delta(q_0, 1) = q_0$

Step – II

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

$= \delta(q_0, 0) \cup \delta(q_1, 0)$

$= \{q_A\} \cup \{q_0\}$ [From table of NFA]

$= q_A$

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

$= \delta(q_0, 1) \cup \delta(q_1, 1)$

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

$= \{q_0, q_2\}$

$= q_B$ (say) = New state

Step – III

$\delta(q_B, 0) = \delta(\{q_0, q_2\}, 0)$

$= \delta(q_0, 0) \cup \delta(q_2, 0)$

$= q_A \cup \phi$

$= q_A$

$\delta(q_B, 1) = \delta(\{q_0, q_2\}, 1)$

$= \delta(q_0, 1) \cup \delta(q_2, 1)$

$= q_0 \cup \phi$

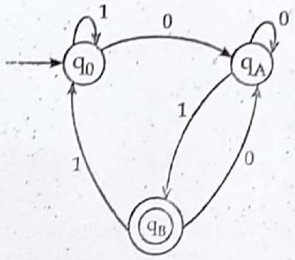
$= q_0$

Constructing a transition table from above, we get

States/Σ	0	1
→ q_0	q_A	q_0
q_A	q_A	q_B
* q_B	q_A	q_0

Here, q_B contains q_2 (the final state). So, it is a final state.

Now, the equivalent DFA of NFA is given by.



6. Design a NFA that accepts all strings over {0, 1} in which second last symbol is always '1'. Then convert it into DFA.

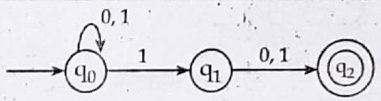
Solution:

Given,

L = {accepts all strings over {0, 1} in which second last symbol is always '1'}

= {0010, 110, 111, ...}

The state diagram for NFA is



Transition table for NFA,

States/Σ	0	1
→ q_0	q_0	$\{q_0, q_1\}$
q_1	q_2	q_2
* q_2	ϕ	ϕ

Step - I

$\delta(q_0, 0) = q_0$ = old state

$\delta(q_0, 1) = \{q_0, q_1\} = q_A$ (say) = New state

Step - II

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

$= \delta(q_0, 0) \cup \delta(q_1, 0)$

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

$= \{q_0, q_2\}$

$= q_B$ (say) = New state

$\delta(q_A, 1) = \delta(\{q_0, q_1\}, 1)$
$= \delta(q_0, 1) \cup \delta(q_1, 1)$
$= \{q_0, q_1\} \cup q_2$
$= \{q_0, q_1, q_2\}$
$= q_C$ (say) = New state

Step - III

$\delta(q_B, 0) = \delta(\{q_0, q_2\}, 0)$
$= \delta(q_0, 0) \cup \delta(q_2, 0)$
$= \{q_0\} \cup \phi$
$= q_0$

$\delta(q_B, 1) = \delta(\{q_0, q_2\}, 1)$
$= \delta(q_0, 1) \cup \delta(q_2, 1)$
$= \{q_0, q_1\} \cup \phi$
$= \{q_0, q_1\}$
$= q_A$

Step - IV

$\delta(q_C, 0) = \delta(\{q_0, q_1, q_2\}, 0)$
$= \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)$
$= \{q_0\} \cup \{q_2\} \cup \phi$
$= \{q_0, q_2\}$
$= q_B$

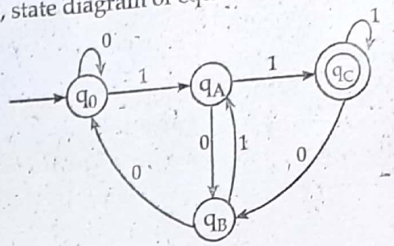
$\delta(q_C, 1) = \delta(\{q_0, q_1, q_2\}, 1)$
$= \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)$
$= \{q_0, q_1\} \cup \{q_2\} \cup \phi$
$= \{q_0, q_1, q_2\}$
$= q_C$

∴ Transition table for equivalent DFA is

States/Σ	0	1
→ q_0	q_0	q_A
q_A	q_B	q_C
* q_B	q_0	q_A
* q_C	q_B	q_C

Since, q_B and q_C both contains state q_2. So, these states are the final states of DFA as q_2 is the final state in NFA.

Now, state diagram of equivalent DFA is



This DFA accepts the strings that has second last element '1'.

Minimization of DFA

The same problem in finite automata can be solved by many DFAs varying in the number of states. But the DFA, with lesser number of states is considered to be the best.

So, two states of any DFA can be made single state if they are equivalent and hence DFA has lesser number of states. And this process is known as minimization of DFA.

Two states A and B of DFA are said to be equivalent if

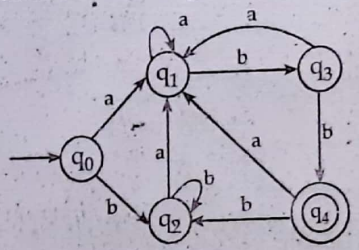
$\delta(A, X) \rightarrow F$ and $\delta(B, X) \rightarrow F$ OR $\delta(A, X) \longrightarrow F$ and $\delta(B, X) \longrightarrow F$

where, X is any string.

If state A on input X goes to any of final state F. And at the same time, state B also goes to any of final state then state A and B are said to be equivalent. The types of equivalences are:

(a) 0 equivalence ⊣ $|X| = 0$, i.e. length of string is 0
(b) 1 equivalence ⊣ $|X| = 1$, i.e. length of string is 1
(c) 2 equivalence ⊣ $|X| = 2$, i.e. length of string is 2
(d) n equivalence ⊣ $|X| = n$, i.e. length of string is n

1. Minimize the following DFA



Solution:

The transition table of above DFA is

States/Σ	a	b
→ q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
q_4	q_1	q_2

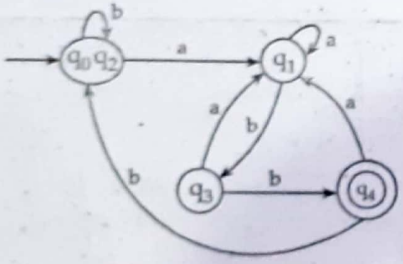
0 equivalence, $\{q_0, q_1, q_2, q_3\}$ $\{q_4\}$

1 equivalence, $\{q_0, q_1, q_2\}$ $\{q_3\}$

2 equivalence, $\{q_0, q_2\}$ $\{q_1\}$ $\{q_3\}$

3 equivalence, $\{q_0, q_2\}$ $\{q_1\}$ $\{q_3\}$ $\{q_4\}$

Note: This equivalence process goes on until two consecutive equivalences are matched.



This is the required minimized DFA.

2. Construct a minimum DFA equivalent to the DFA described by

States/Σ	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
* q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Solution:

From the transition table on the question above. We can use the equivalence method.

Writing non-final states and final state separately we get,

0 equivalence

$\{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}$ $\{q_2\}$

1 equivalence

Combine those states which have same transition states for all of the inputs or some of the inputs.

$\{q_0, q_4, q_6\}$ $\{q_1, q_7\}$ $\{q_3, q_5\}$ $\{q_2\}$

Note: q_1 and q_7 have same transition state for input 0 and 1. And so do the q_3 and q_5 have

2 equivalence

q_0 and q_4 have transition state q_1 and q_7 respectively for input '0' and q_1 and q_7 belongs to same set in 1 equivalence. And on input '1' q_0 and q_4 have same transition state q_5. So, q_0 and q_4 are 2 equivalent checking in similar way for q_4 and q_6, we get to know q_6 is not 2-equivalent.

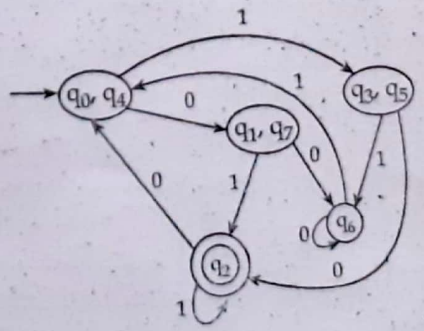
∴ $\{q_0, q_4\}$ $\{q_6\}$ $\{q_1, q_7\}$ $\{q_3, q_5\}$ $\{q_2\}$

Since, 2 equivalence and 3 equivalence are same. So process stops.

∴ Minimized transition table.

States/E	0	1
→ $\{q_0, q_4\}$	$\{q_1, q_7\}$	$\{q_3, q_5\}$
$\{q_6\}$	$\{q_6\}$	$\{q_0, q_4\}$
$\{q_1, q_7\}$	$\{q_6\}$	$\{q_2\}$
$\{q_3, q_5\}$	$\{q_2\}$	$\{q_6\}$
* $\{q_2\}$	$\{q_0, q_4\}$	$\{q_2\}$

State diagram



Regular languages

A language is said to be regular language, if and only if finite state machine recognizes it.

Note: Irregular language require memory like

→ A set of strings that has repeated form of string aab. Example; a abaaba ab.

In order to check repeated form, we need to store and somewhere that is what makes it irregular.

→ A set of strings that has equal number of 0 and 1. Example; 0011.

Now, in order to check equality in number of 0 and 1 we need to store the number of 0's and 1's for equality operation which is irregular.

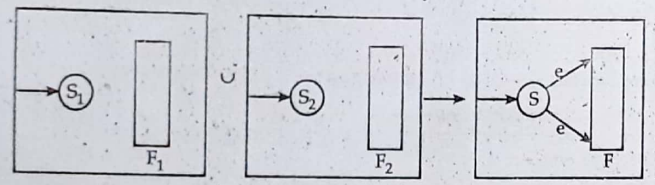
Closure properties of regular language

Let, $L(M_1) = \{K_1, E_1 \Delta_1, S_1, F_1\}$

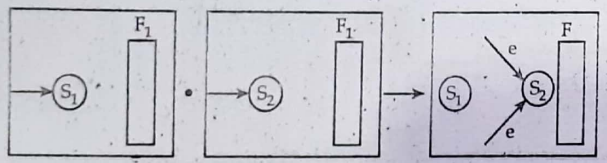
$\Lambda(M_2 = \{K_2, E_2 \Delta_2, S_2, F_2\}$

(a) **Union:** Then we can construct,

$L(M) = L(M_1) \cup L(M_2)$

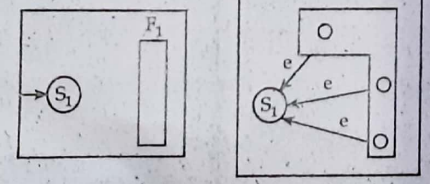


(b) **Concatenation:**



(c) **Kleene star:**

$L(M) = L(M)^*$



Examples: Let a ⟶ O —a→ ⊙

b ⟶ O —b→ ⊙

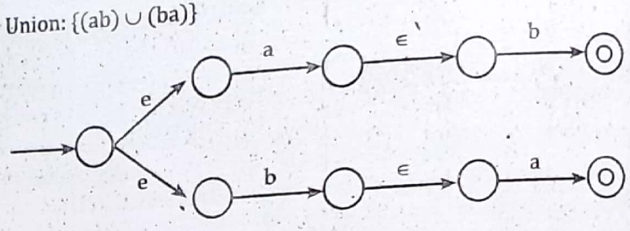
Then,

Concatenation of ab

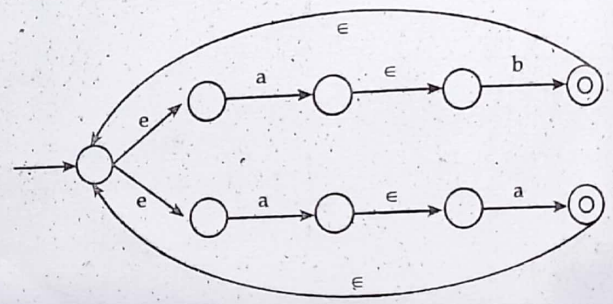
ab ⟶ O —a→ O —∈→ O —b→ ⊙

ba ⟶ O —b→ O —∈→ O —a→ ⊙

Union: {(ab) ∪ (ba)}



Kleene star: {(ab) ∪ (ba)}*



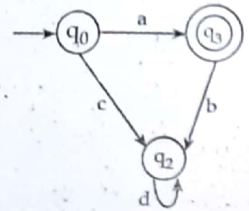
(d) **Complementation:** Let M = K, Σ, Δ, S, F) be a deterministic finite automata. Then the complementary language $\overline{L} = \Sigma^* - L(M)$ is accepted by the DFA $\overline{M}$ = (K, Σ, Δ, S, K – F). That is $\overline{M}$ identical to M except that final and non-final states are interchanged

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

Note: Proofs of closure properties of regular languages is given in set 2072 Chaitra]

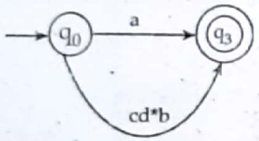
Equivalence of finite automata and regular expression

1. **Write a regular expression of the following state-diagram.**



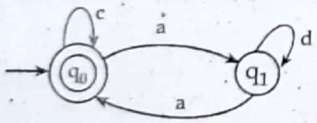
Solution:

The above state diagram can be change into,



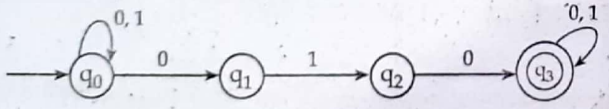
Ans is R = (a ∪ cd* b)

2. **Write the equivalent regular expression for the following state diagram.**



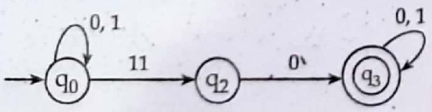
⇒ R = (c* ∪ ad * b)*

3. **Find the regular expression from the following**

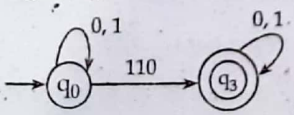


Solution:

q_1 is neither initial nor final state. So we can eliminate q_1



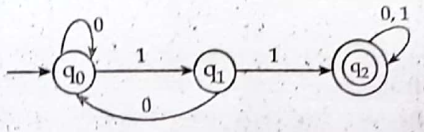
Again, q_2 is neither initial nor final state. We eliminate q_2 also.



Hence, the required expression is

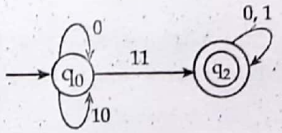
R = (0 ∪ 1)* 110 (0 ∪ 1)*

4. **Find the regular expression of the following DFA.**



Solution:

q_1 is neither initial nor final state. So, we eliminate q_1:



∴ Regular expression is

R = (0 ∪ 10)* 11 (0 ∪ 1)*

Pumping Lemma for regular language

Statement: If language A is regular language the A has a pumping length 'p' such that any string 's' where |s| ≥ p any be divided into 3 points s = xyz such that following conditions must be true.

(a) $xy^iz \in A$ for every $i \geq 0$

(b) |y| > 0 (length of y)

(c) $|xy| \leq p$ (length of x and y)

Steps to be followed while proving language is not reular by pumping lemma:

(a) Assume that A is regular.

(b) It has to have a pumping length (say P).

(c) All strings longer than P can be pumped.

(d) Now find a string 's' in A such that |S| ≥ P.

(e) Divide S into x, y, z.

(f) Show that, $xy^iz \in A$ for some i.

(g) Then consider all ways that S can be divided into x, y, z.

(h) Show that none of these can satisfy all the 3 pumping conditions at the same time.

(i) S can't be pumped = contradiction.

(j) We should learn it by some examples.

1. Using pumping lemma prove that the language $A = \{a^n b^n / n \geq 0\}$ is not regular.

Solution:

(Theoretically, we can conclude that this language is not regular as the language generates strings which contains certain number of a's followed by equal number of b's. And to do that, finite state machine need to have some memory for storing counted number of a but in actual it doesn't have memory. So, if a language is not recognized by finite state machine, it is not regular.)

Using pumping lemma, let's assume that language A is regular. Now, it must follow following conditions.

(a) $XY^iz \in A$ for every $i \geq 0$ ... ... (i)

(b) $|y| > 0$ ... ... (ii)

(c) $|xy^i| \leq p$ where pis pumping length... ... ... (iii)

Let, $P = 7$ (say)

Let, $S \in A = a^P b^P$

$= a^7 b^7$

= aaaaaaabbbbbbb

Dividing S into x, y and z. Gets these cases.

Case – I

When y is in the 'a' part.

i.e. S = $\underbrace{a}_{x}$ $\underbrace{aaaa}_{y}$ $\underbrace{aabbbbbbb}_{z}$

Here, condition 2 and 3 are satisfied as $|y| = 4 > 0$ and $|xy| = 5 < 7$.

For condition 1

$xy^iz = xy^2z$ [i = 2 say]

Then,

S = a aaaa aaaa aabbbbbbb

So, number of 'a' = 11

number of 'b' = 7

∴ Number of 'a' ≠ number of 'b'.

The first condition is not satisfied.

i.e. $xy^2z \notin A$

Case – II:

When y in the both 'a' and 'b' part

i.e. S = $\underbrace{aaaaaa}_{x}$ $\underbrace{ab}_{y}$ $\underbrace{bbbbbb}_{z}$

Let $S = xy^2$ (i = 2)

= aaaaaa abab bbbbbb

Here, pattern is not followed i.e. $a^n b^n$.

So, condition 1 is not satisfied and condition 3 as well as $|xy| = 8 > 7$

Case - III

When y is in the 'b' part.

i.e. S = $\underbrace{aaaaaaabb}_{x}$ $\underbrace{bbb}_{y}$ $\underbrace{bb}_{z}$

Let, $S = xy^2z$

= aaaaaaabb bbbbbb bb

Here, also condition 1 is not satisfied as, number of a's ≠ number of b's,

So, none of the cases satisfy the all 3 conditions of regular language as stated by pumping lemma. Therefore language A is not regular.

2. Using pumping lemma prove that $A = \{yy/y \in \{0, 1\}^*$ is not regular.

Solution:

According to pumping lemma, 'if a language is regular then it must have string $|S| \geq P$ where P is pumping length and can be divided into xyz such that:

(a) $xy^iz \in A$ for some i ... ... (i)

(b) $|y| > 0$ ... ... (ii)

(c) $|xy| \leq p$ ... ... (iii)

To prove: Language $A = \{yy//y \in (0, 1)^*\}$ is not regular

Proof: Let us consider language A is regular then it must follow above 3 conditions. Then it must have pumping length P = 7 (say).

Let, the strings be accepted by finite state machine

$S = 0^p10^p1$

$= 0^710^71$

$= 0000000100000001$

Dividing S into x, y and z we get,

$S = \underbrace{00}_{x}\ \underbrace{00000}_{y}\ \underbrace{100000001}_{z}$

Here, the above condition (ii) and (iii) are satisfied as

$|y| \geq 0$

$|xy| = 7 \leq 7$

Changing for condition 1

$S = xy^iz$ let, $i = 2$

$= xy^2z$

$= 00\ 0000000000\ 100000001$

Here, , on pumping i.e. (xy^iz) the language doesn't follow pattern (0^p10^p1) (i.e. first half is equal to the second half). So none of the cases follow all 3 conditions of regular language as stated by pumping lemma. Therefore language A is not regular language.

◆◆◆

Exam Solution

1. List closure properties of regular expression. If M and N are any two regular languages then show that L = (M ∪ N) is also regular language [2075 Ashwin, Back]

The closure properties of regular language are as follows:

(a) Union

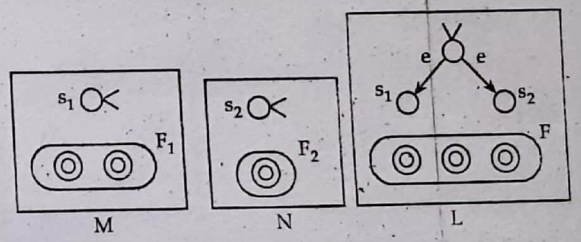
(b) Concatenation

(c) Kleene star

(d) Complementation

(e) Intersection

Let $M = (\theta_1, \Sigma, \Delta_1, S_1, F_1)$ and $N = (\theta_2, \Sigma, \Delta_2, S_2, F_2)$ be non-deterministic finite automata, we shall construct a non-deterministic finite automata such that $L = M \cup N$. The construction of L is rather simple and intuitively clear, illustrated in fig. (i). Basically, L uses non-determinism to guess whether the input is in M or in N, and then processes the string exactly as the corresponding automaton would, it follows that $L = M \cup N$. But let us give the formal details and proof for this case.



Without loss of generally, we may assume that θ_1 and θ_2 are disjoint sets. Then the finite automaton L accepts that accepts $M \cup N$ is defined as follows:

$L = (\theta, \Sigma, \Delta, S, F)$, where S is a new state not in θ_1 or θ_2

$\theta = \theta_1 \cup \theta_2 \cup \{S\}$

$F = F_1 \cup F_2$

$\Delta = \Delta_1 \cup \Delta_2 \cup \{(S, e, S_1), (S, e, S_2)\}$

That is, L beings any computation by non-deterministically choosing to enter either the initial state of M or the initial state of N and thereafter, L imitates either M or N. Hence, L accepts w if and only if M accepts w or N accepts w and $L = M \cup N$.

2. **Write statement of pumping lemma for regular languages. Show that $L = \{a^n b^n, n \geq 0\}$ is not a regular language by using pumping lemma.** **[2075 Ashwin, Back]**

Please refer to the theory part

3. **What is the significance of finite automata? Design a DFA that accepts the strings over an alphabet $\Sigma = \{0, 1\}$ that either start with 01 or end with 01. Hence test your design for any two strings.** **[2075 Ashin, Back]**

The significance of finite automata are as follows:

(a) It is very useful in processing strings i.e. it can accept certain input strings and reject others.

(b) It is useful for communication protocols.

(c) It is used to parse formal languages.

(d) It is useful in the creation of compiler and interpreter techniques.

Let L(M) be the required DFA

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$q_0 = \{q_0\}$

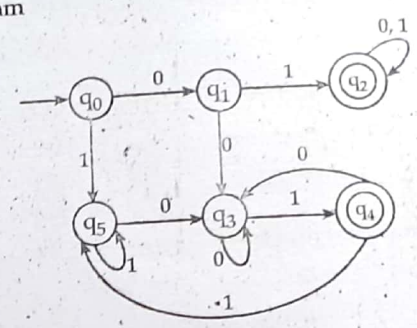
$F = \{q_2, q_4\}$

and δ is defined as follows:

Transition able

States/Σ	0	1
→ q_0	q_1	q_5
q_1	q_3	q_2
* q_2	q_2	q_2
q_3	q_3	q_4
* q_4	q_3	q_5
q_5	q_3	q_5

State diagram



Test for input string '0110'

$(q_0, 0010)) \vdash M(q_1, 110)$

$\vdash M(q_2, 10)$

$\vdash M(q_2, 0)$

$\vdash M(q_2, \epsilon)$ Accepted

Since, q_2 is final state, string '0110' is accepted

Test for input string '11100'

$q_0, 11100) \vdash M(q_5, 1100)$

$\vdash M(q_5, 100)$

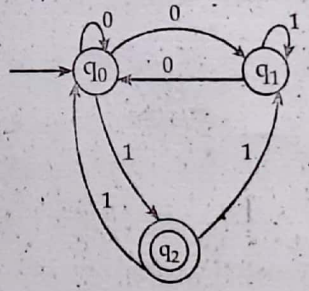
$\vdash M(q_5, 00)$

$\vdash M(q_3, 0)$

$\vdash M(q_3, \epsilon)$ Rejected

Since, q_3 is not final state, string '11100' is rejected.

4. **Differentiate between DFA and NDFA. Convert the following NDFA to its DFA.** **[2074 Ashwin, Back]**



For the differences between DFA and NDFA, see theory part.

Here, transition table of given NDFA.

States/Σ	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_2\}$
q_1	q_0	q_1
* q_2	ϕ	$\{q_0, q_1\}$

Transition table of required DFA

States/Σ	0	1
→ q_0	q_A	q_2
q_A	q_A	q_B
* q_B	q_0	q_A
* q_2	q_d	q_A
q_d	q_d	q_d

Note: readers are requested to derive steps on their own using previous theory part.

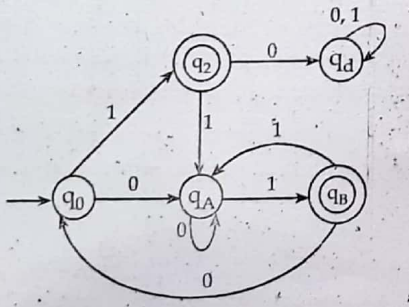
where, $q_A = \{q_0, q_1\}$

$q_B = \{q_1, q_2\}$

q_d = dead state

Here, aestrick is used to denoted final states in transition table.

State diagram of equivalent DFA,



5. What are the components of finite automata? Design a DFA that accepts the strings given by L = {w ∈ {a, b}* : w has number of a divisible by 3 and number of b divisible by 2. [2075 Ashwin Back]

The components of finite automata are:

(a) Input tape: It contains single string.

(b) Reading head: It reads input string one symbol at a time.

(c) Memory: It is in one of a finite number of states.

Let, L(M) be the required DFA

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

where, $\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

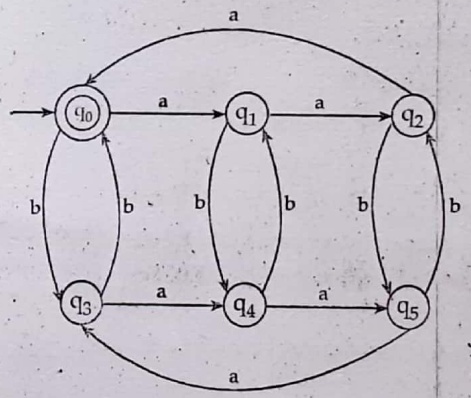
$q_0 = \{q_0\}$

$F = \{q_0\}$

and transition δ is defined as

States/Inputs	a	b
→ *q_0	q_1	q_3
q_1	q_2	q_4
q_2	q_0	q_5
q_3	q_4	q_0
q_4	q_5	q_1
q_5	q_3	q_2

States diagram



Accepted strings

$(q_0, ababa)$ $\vdash M (q_1, baba)$

$\vdash M (q_4, aba)$

$\vdash M (q_5, ba)$

$\vdash^* M (q_2, a)$

$\vdash M (q_0, \epsilon)$ Accepted

Since, q_0 is final state, string 'ababa' is accepted.

Test for input string 'ababbba'

$(q_0, ababbba) \vdash M(q_1, babbba)$
$\vdash M(q_4, abbba)$
$\vdash M(q_5, bbba)$
$\vdash M(q_2, bba)$
$\vdash M(q_5, ba)$
$\vdash M(q_2, a)$
$\vdash M(q_0, \epsilon)$ Accepted

Since, q_0 is final state, string 'ababbba' is accepted.

Rejected strings

Test for input string 'bbabb'

$(q_0, bbabb) \vdash M(q_3, babb)$
$\vdash M(q_0, abb)$
$\vdash M(q_1, bb)$
$\vdash M(q_4, b)$
$\vdash M(q_1, \epsilon)$ Rejected

Since, q_1 is not final state, so string 'bbabb' is rejected

Test for input string 'bba'

$(q_0, bba) \vdash M(q_3, ba)$
$\vdash M(q_0, a)$
$\vdash M(q_1, \epsilon)$

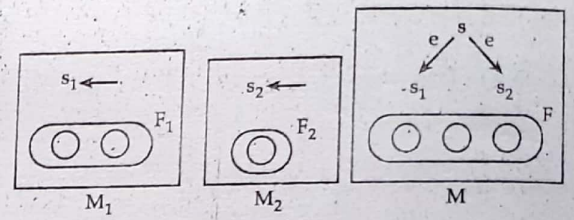
6. Define closure properties of regular language prove that regular are closed under union, intersection and complementation operation. [2074 Ashwin, Back]

The closure properties of regular language are

(a) Union
(b) Concatenation,
(c) Kleene star
(d) Complementation
(e) Intersection

To prove: Regular languages are closed under union i.e. union of two regular languages is also regular.

(a) **Union:** Let $M_1 = (Q_1, \Sigma, \delta_1, S_1, F_1)$ and $M_2 = \{\theta_2, \Sigma, \delta_2, S_2, F_2\}$ be non-determinist finite automata. We shall construct a non-deterministic finite automata such that $L(M) = L(M_1) \cup L(M_2)$. The construction of M is rather simple and intuitively clear illustrated in fig below. Basically, M uses non-determinism to guess whether input is in $L(M_1)$ or in $L(M_2)$ and then processes the string exactly as the corresponding automata would it follows that $L(M) = L(M_1) \cup L(M_2)$. But let us give the formal details and proof for this case.



Without loss of generally, we may assume that K_1 and K_2 are disjoint sets. Thus, the finite automata M that accepts $L(M_1) \cup L(M_2)$ is defined as shown in figure above $M = \{\theta, \Sigma, \delta, S, F\}$ where S is a new state not in θ_1 and θ_2.

$\theta = \theta_1 \cup \theta_2 \cup \{s\}$

$F = F_1 \cup F_2$

$\delta = \delta_1 \cup \delta_2 \cup \{(s, e, s_1), (s, e, s_2)\}$

That is M beings any computation by non deterministically choosing to enter either initial sate of M_1 or the initial state of M_2 and therefore M imitates either M_1 or M_2 thereafter. Hence M accepts w if and only if M_1 accepts w or M_2accepts w and $L(M) = L(M_1) \cup L(M_2)$. Hence, regular properties are closed under union.

(b) **Complementation:** Let $M = (\theta, \Sigma, \delta, q_0, F)$ be a DFA that accepts L, then DFA

$\overline{M} = (\theta, \Sigma, \delta, q_0, \theta, F)$ accepts $\overline{L}$.

We can justify it as if M accepts the language L then some of the states of this FA are final states and must likely, some are not. Let us reverse the final state of each state, that is if it was a final state, make it a non-final state and vice-versa. If an input string formally ended in a non-final state, it now ends in a final

state and vice-versa. This new machine we have built accepts all input strings that weren't accepted by the original FA (all the words in $\overline{L}$) and rejects all input strings that were accepted by original FA. Therefore, in machine accepts exactly the language $\overline{L}$. So L is also regular language.

(c) **Intersection:** From De-Margin's law, we can write

$$L_1 \cap L_2 = \overline{\overline{L_1} + \overline{L_2}}$$

Because L_1 and L_2 are regular so $\overline{L_1}$ and $\overline{L_2}$ are regular and so is $\overline{L_1} + \overline{L_2}$ as we have proved that regular language is closed under union and complementation. And then because $\overline{L_1} + \overline{L_2}$ is regular then so is $\overline{\overline{L_1} + \overline{L_2}}$ which means $L_1 \cap L_2$ is regular. i.e. regular languages are closed under intersection.

7. **Explain finite automata with their application. Design a DFA that accepts the language L = {w ∈ {a, b} : w must have either aaa or bbb as substring} [2074 Chaitra, Regular]**

Finite automata is a abstract model of digital computer. Its mechanisms to read input strings on a input tape and either accepts it or rejects it. Some of the application of finite automata are discussed below:

(a) It is used in parsing formal languages.

(b) It is useful in creation of compiler and interpreter techniques.

(c) It is useful for communication protocols.

(d) It is used in processing input strings.

Let, the required DFA be

$$L(M) = (\theta, \Sigma, \delta, q_0, F)$$

where,

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_4, q_5, q_6\}$

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

$q_0 = \{q_0\}$

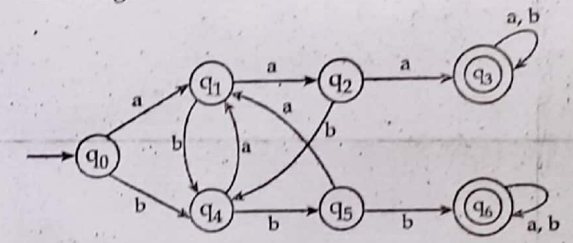
$F = \{q_3, q_6\}$

and transition δ is defined as follows:

Transition table

States/Σ	a	b
→ q_0	q_1	q_4
q_1	q_2	q_4
q_2	q_3	q_4
* q_3	q_3	q_3
q_4	q_1	q_5
q_5	q_1	q_6
* q_6	q_6	q_6

State diagram



Test for input string 'baaabb'

$(q_0, baaabb) \vdash M(q_4, aaabb)$

$\vdash M(q_1, aabb)$

$\vdash M(q_2, abb)$

$\vdash M(q_3, bb)$

$\vdash M(q_3, b)$

$\vdash M(q_3, \epsilon)$ Accepted

Since, q_3 is final state, string 'baaabb' is accepted.

Test for input string 'bbaab'

$(q_0, bbaa) \vdash M(q_4, baab)$

$\vdash M(q_5, aab)$

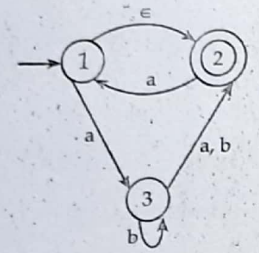
$\vdash M(q_1, ab)$

$\vdash M(q_2, b)$

$\vdash M(q_4, \epsilon)$ Rejected

Since, q_4 is not final state, string 'bbaab' is not accepted.

8. Convert the following NFA into its equivalent DFA. [2074 Chaitra]



Here, this is ϵ-NFA

and e-closure (1) = {1, 2} = A (say)

Transition states

Step I

$\delta(A, a)$ = e-closure (δ (1, 2), a)
= e-closure (δ (1, a) $\cup$ δ (2, a))
= e-closure (3 $\cup$ 1)
= e-closure (3) $\cup$ e-closure (1)
= {3} $\cup$ {1, 2}
= {1, 2, 3}
= B (say)

$\delta(A, b)$ = e-closure (δ (1, 2), b)
= e-closure (δ (1, b) $\cup$ δ (2, b))
= e-closure ($\phi \cup \phi$)
= e-closure (ϕ)
= C (say) (dead state)

Step II

$\delta(B, a)$ = e-closure (δ (1, 2, 3), a)
= e-closure (δ (1, a) $\cup$ δ (2, a) $\cup$ δ(3, a))
= e-closure (3 $\cup$ 1 $\cup$ 2)
= e-closure (3) $\cup$ e-closure (1) $\cup$ e-closure (2)
= {3} $\cup$ {1, 2} $\cup$ {2}
= {1, 2, 3}
= B

$\delta(B, b)$ = e-closure (δ (1, 2, 3), b)
= e-closure (δ (1, b) $\cup$ δ (2, b) $\cup$ (3, b))
= e-closure ($\phi \cup \phi \cup$ {3, 2})
= e-closure (3) $\cup$ e-closure (2)
= {2} $\cup$ {3} = {2, 3}
= D (say)

$\delta(D, a)$ = e-closure (δ (2, 3), a)
= e-closure (δ (2, a) $\cup$ δ (3, a))
= e-closure (1 $\cup$ 2)
= e-closure (1) $\cup$ e-closure (2)
= {1, 2} $\cup$ {2}
= {1, 2}
= A

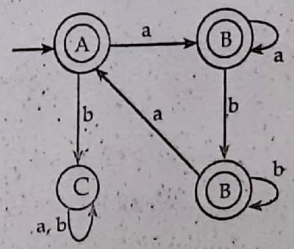
$\delta(D, b)$ = e-closure (δ (2, 3), b)
= e-closure (δ (2, b) $\cup$ δ (3, b))
= e-closure ($\phi \cup$ {3, 2})
= e-closure (3) $\cup$ e-closure (2)
= {3} $\cup$ {2}
= {2} $\cup$ {3}
= D

e-closure δ(c, a) = e-closure δ(c, b) = c

Transition table

States/Σ	a	b
$\rightarrow$ * A	B	C
C	C	C
* B	B	D
* D	A	D

State diagram



Note: Final states of DFA are all those new states which contains final states of NFA with ϵ-transitions.

9. State the pumping lemma for the regular languages. Show that the languages

$L = \{0^{n^2} \mid n >= 1\}$ not regular example,

if x = 1, w = 0, n = 2, w = 0000, n = 3, w = 000000000 [2074, Chaitra]

Pumping lemma states that if a language A is regular then it must have string $|s| \geq P$ where P is pumping length and can be divided into xyz such that

(a) $xy^iz \in A$ for some i

(b) $|y| > 0$

(c) $|xy| \leq P$

Here, given language is $L = \{0^{n^2}\} / n >= 1\}$

To prove: L is not regular

Proof: Using pumping lemma

Let's assume language L is regular.

Then, if must have pumping length P.

Let P = 3 (say)

Then, let s be the string accepted by finite automaton.

i.e. $s = o^{n^2}$

$= 0^{3^2}$

$= 0^9$

= 000000000

Dividing s into x, y and z as

$\underbrace{0}_{x}$ $\underbrace{00}_{y}$ $\underbrace{000000}_{z}$

Checking for condition 1

$s = xy^iz = xy^2z$ (e = 2 (say))

= 0 0000 000000

Now, $|s| = 11$

Here, no value of P(or n) would give (s) = 1 in other words, length of string s on being pumped is not perfect square which means it doesn't belong to language L so, condition 1 fails. Though, condition 2 and 3 are satisfied.

$|y| = 2 > 0$

$|xy|\ 3 \geq P$

Since, condition 1 is failed, our assumption contradicts. That's why language L is not regular.

10. Differentiate between NFA and DFA. Design a DFA that accepts the language $L = \{x : x \in \{0, 1\}^* \mid 10110$ doesn't occur as substring in x}. Verify your design with supporting examples. [2073 Chaitra]

For the difference between NFA and DFA, see theory part.

Let, L(M) be the required DFA

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

where,

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$q_0 = \{q_0\}$

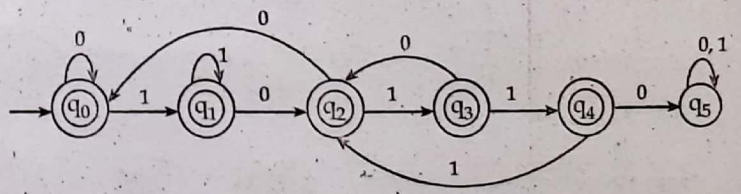
$F = \{q_0, q_1, q_2, q_3, q_4\}$

and δ is defined as

Transition table

States/Σ	0	1
→ * q_0	q_0	q_1
* q_1	q_2	q_1
* q_2	q_0	q_3
* q_3	q_2	q_4
* q_4	q_5	q_1
q_5	q_5	q_5

State diagram



Test for input string '1101100'

$(q_0, 1101100) \vdash M(q_1, 101100)$

$\vdash M(q_1, 01100)$

$\vdash M(q_2, 1100)$

$\vdash M(q_3, 100)$

$\vdash M(q_4, 00)$

$\vdash M(q_5, 0)$

$\vdash M(q_5, \in)$ Rejected

Sine, q_5 is not final state and string contains '10110' as substring. So, it is rejected.

Test for input string '11100'

$$(q_0, 11100) \vdash M(q_1, 1100)$$
$$\vdash M(q_1, 100)$$
$$\vdash M(q_1, 00)$$
$$\vdash M(q_2, 0)$$
$$\vdash M(q_0, \epsilon)$$

Since, q_0 is one of the final states string '11100' is accepted.

11. State pumping lemma for regular language. Use pumping lemma and prove that language L = {w | w ∈ {0, 1}* and w has an equal number of 0's and 1's} is not regular. [2073 Chaitra]

For statement of pumping lemma for regular language, please see theory part.

Proof: Let, assume that language L is regular. Now, since it is regular, it must follow these conditions:

(a) $xy^iz \in L$ for every $i \geq 0$

(b) $|y| > 0$,

(c) $|xy| \leq P$ where P is pumping length

Lent, S ∈ L = 01010101 and P = 3

Now, dividing S into x, y and z, we get

$$\underbrace{0}_{x} \quad \underbrace{10}_{y} \quad \underbrace{10101}_{z}$$

Here, the above condition (ii) and (iii) are satisfied as

$$|y| = 2 > 0$$
$$|xy| = 2 \leq 3$$

Now, checking for condition 1,

xy^iz

Let, $i = 2$

xy^2z

= 01010 10101

Then, no of 0's is not equal to number of 1's so, it doesn't belong to L and condition 1 fails. If any of the above condition fails, then our assumption is also false which means language L is not regular.

12. Why is NDFA important although it is equivalent to a DFA? Design NDFA which accepts L {w | w ∈ {a, b}* such that w contains either two consecutive a's or two consecutive b's} [2073 Chaitra]

NDFA (non-deterministic finite automata) is important although it's equivalent to DFA (Deterministic Finite Automata) because of following reasons:

(a) NDFA's are easy to implement and to construct than DFA's.

(b) For choosing the right path in implementation of a particular algorithm, usually NFAs to DFAs conversion works.

(c) NDFAs can be used to reduce the complexity.

(d) NDFAs can be used to reduce the complexity of the mathematical work.

(e) It is much easier to prove closure properties of regular language using NDFAs than DFAs.

Let, the required NDFA be

$$L(M) = (\theta, \Sigma, \delta, q_0, E)$$

where,

$\theta = \{A, B, C, D, E\}$

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

$q_0 = \{A\}$

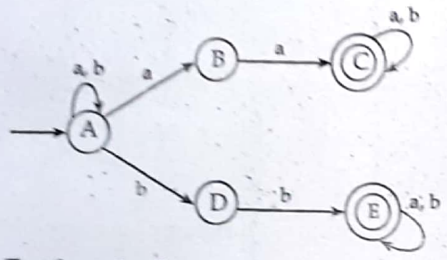
$F = \{C, E\}$

and δ is defined as follows:

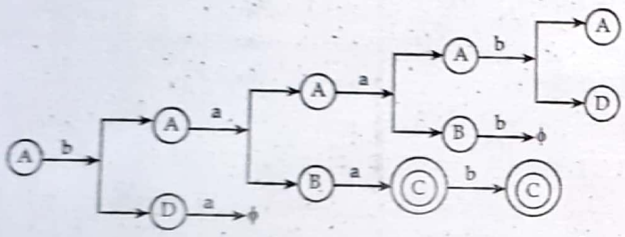
States/Σ	a	b
A	{A, B}	{A, D}
B	C	ϕ
C	C	C
D	ϕ	E
E	E	E

In transition table, some states have multiple states to go example A and some states have nowhere to go on input like (B and D). Therefore, this represents NDFA.

State diagram



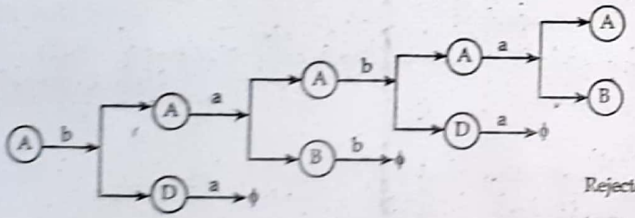
Test for string 'baab'



Accepted

Since, one of the final state is acceptor state i.e. C it is accepted.

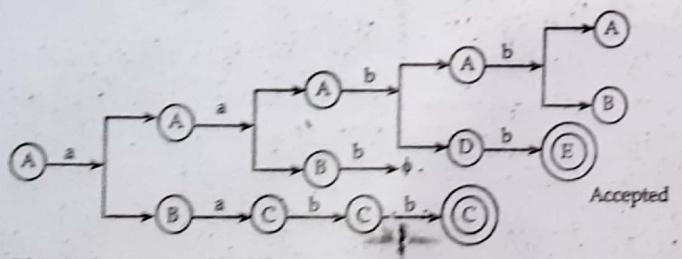
Test for string (baba)



Rejected

Hence, none of final state is acceptor state, so string 'baba' is rejected.

Test for string 'aabb'



Accepted

This is the required NDFA

13. For every NDFA, there exists a DFA which simulates the behaviour of NDFA. Alternative if L is the set accepted by NDFA. Then exists a DFA which also accepts L.

Let M = {Q, Σ, δ, q_0, F} be q NDFA accepting language L. We construct a DFA M' as follows.

$$M' = \{Q' \Sigma, \delta', q_0', F'\}$$

where,

(a) $Q' = 2^Q$ (any state in Q is denoted by $\{q_1, q_2, \ldots q_i\}$ where $q_1, q_2, \ldots q_i \in Q$)

(b) $q'_0 = [q_0]$

(c) F' is the set of all subsets of Q containing an element of F.

(d) $(\delta' (q_1, q_2, \ldots, q_i), a) = \delta (q_1, a) \cup \delta(q_2, a) \ldots \cup \delta(q_i, a)$ equivalently $\delta' (q_1, q_2, q, j, q) = [p_1, p_2, p_j]$ if and only if $\delta[\{q_1, q_2, \ldots q_i\}, q] = \{p_1, p_2, \ldots p_j\}$ if and only if $\delta(q_0, x) = \{q_1, \ldots q_i\}$ for all x in Σ

First we are going to prove by induction that $\delta^* (q_0, w) = \delta'^* (q'_0, w)$ for any string w, when it is proven, it obviously implies that NDA M and DFA M' accept the same strings.

Now, we have q task to prove $\delta^* (q_0, w) = \delta'^* (q'_0, w)$. This is going to be proven by induction on w.

Note: δ^* represents the transition function for processing strings.

Basic step: For w = t

$\delta'^* = (q'_0, t) = q'_0$ by the definition d'*.

$= q_0$ by construction of DFA M'.

$= \delta^* (q_0, t)$ by the definition of δ^*.

Inductive step:

Assume that $\delta^* (q_0, w) = \delta'^* (q'_0, w)$ for an arbitrary string w ... Induction hypothesis.

For string w and arbitrary symbol a in Σ

$\delta^* (q_0, wa) = U_{PE}\, \delta^* (q_0, w)\, \delta(p, a)$

$= \delta' (\delta^* (q_0, w), a)$

$= \delta' (\delta'^* (q'_0, w), a)$

$= \delta^* (q'_0, wa)$

Thus, for any string w $\delta^* (q_0, w) = \delta'^* (q'_0, w)$ holds. So, we can say that L(M) = L(M'). i.e. To every NDFA, there exists an equivalent DFA.

14. **Construct a NFA for the language (ab* a ∪ b*aa) provide any two accepted string and two rejected strings.** **[2073 Shrawan]**

Let, the required NFA be

$$L(M) = (\theta, \Sigma, \delta, q_0, F)$$

where,

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$q_0 = \{q_0\}$

$F = \{q_2, q_5\}$

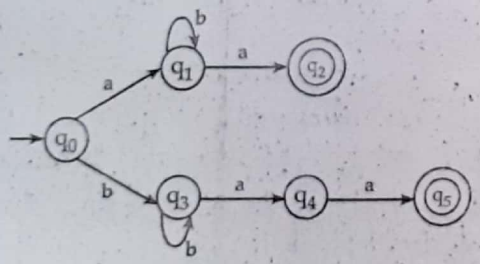
and transition δ is defined as

Transition table (δ)

States/Σ	a	b
→ q_0	q_1	q_3
q_1	q_2	q_1
* q_2	ϕ	ϕ
q_3	q_4	q_3
q_4	q_5	ϕ
* q_5	ϕ	ϕ

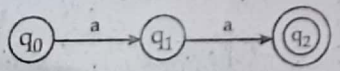
As we can see there are no transition state for some inputs on some states. So, it is NFA.

State diagram



Accepted strings

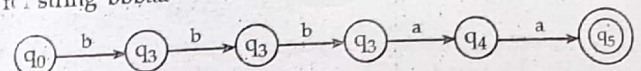
Test for string 'qq'



Accepted

Here, b* means b can be empty (∈). So, string aa can be generated from (ab* a ∪ b*aa). And since, q_2 is accepting state of final state, string 'aa' is accepted.

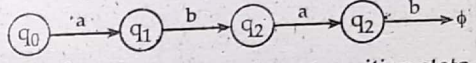
Test for string 'bbbaa'



Hence, q_5 is one of the accepting state of final state. Hence, string 'bbbaa' is accepted.

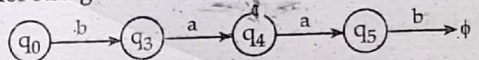
Rejected strings

Test for string 'abab'



On input 'b' to state q_2, there is no transition state defined. So, string 'abab' is rejected.

Test for string 'baab'



Similarly, on input 'b' to state q_2, there is no transition state defined. So, string 'baab' is rejected.

15. **Define configuration of DFA design a deterministic. Finite automata (DFA) for language L = {w ∈ {0, 1} : w has both 01 and 10 as substring} verify your design by taking one accepted and one rejected strings.** **[2073 Shrawan]**

The general configuration of DFA is

$$(q, w) \in \theta \times \Sigma^*$$

where,

q ⟶ initial state

w ⟶ string

θ ⟶ set of states of finite automata

The derivable configuration of DFA is

$$(q, w) \vdash M(q', w')$$

where

q ⊢ initial state

w ⊢ initial string

q' ⊢ transition state

w' ⊢ new string

M ⊢ finite state machine

Let, the required DFA be

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

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

$q_0 = \{q_0\}$

$F = \{q_3, q_6\}$

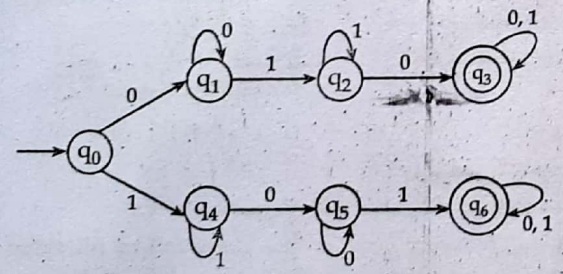
and transition δ is defined as follows:

Transition table

States/Σ	0	1
→ q_0	q_1	q_4
q_1	q_1	q_2
q_2	q_3	q_2
* q_3	q_3	q_3
q_4	q_5	q_4
q_5	q_5	q_6
* q_6	q_6	q_6

Let us analyze the language of given DFA it generates following strings {010, 101, 01011, 00110, ...}

State diagram



Test for input string '01100'

$(q_0, 01100) \vdash M(q_1, 1100)$

$\vdash M(q_1, 100)$

$\vdash M(q_2, 00)$

$\vdash M(q_3, 0)$

$\vdash M(q_3, \in)$ Accepted

Since, string '01100' contains both '01' and '10' as substring, it is accepted.

Test for input string '0011'

$(q_0, 0011) \vdash M(q_1, 011)$

$\vdash M(q_1, 11)$

$\vdash M(q_2, 1)$

$\vdash M(q_2, \in)$ Rejected

Since, string '0011' contains '01' as substring but doesn't contain '10' as substring, so it is rejected.

16. State pumping lemma, for regular language and use the theorem to prove that $L = \{a^n b^{2n} : n \geq 1\}$ is not regular. [2073, Shrawan]

If a language L is a regular language, then it must have a pumping length 'P' such that any string $S \in L$ having length $|S| \geq P$ can be divided into 3 parts such that following conditions are true.

(a) xy^iz for every $i \geq 0$

(b) $|y| > 0$

(c) $|xy| \leq p$

Given, $L = \{a^n b^{2n} : n \geq 1\}$

To prove: L is not regular

Proof: We prove it by contradiction using pumping lemma.

Let's assume language L is regular having pumping length 'P'.

Let, $S \in L = a^P b^{2P}$

Let, P = 4 (say)

Then, $S = a^4 b^8$

$= aaaabbbbbbbb$

Dividing S into x, y and z, we get

Case - I: When y contains m' y 'a'

$S = \underbrace{aa}_{x} \underbrace{aa}_{y} \underbrace{bbbbbbbb}_{z}$

Checking for condition I

Let, $i = 2$

$S = xy^2z$

$= aa\ aaaa\ bbbbbbbb$

$= a^6b^8 \notin L$

Here, condition 1 fails as 'b' is not as twice as 'a'.

Case – II: Why y contains both kind of symbols 'a' and 'b' i.e.

$S = \underbrace{aa}_{x}\ \underbrace{aaa}_{y}\ \underbrace{bbbbbbb}_{z}$

Checking for condition I

Let, $i = 2$

$S = xy^2z$

$= aa\ aab\ aab\ bbbbbbb \notin L$

Again, condition 1 fails as it doesn't follow pattern

Case – III: When y contains only b's i.e.

$S = \underbrace{aaaa}_{x}\ \underbrace{bbbb}_{y}\ \underbrace{bbbb}_{z}$

Checking for condition I

Let, $i = 2$

$S = xy^2z$

$= aaaa\ bbbbbbbbbbbb$

$= a^4 b^{12} \notin L$

Again, condition 1 fails

Here, in pumping (i.e. xy^iz) doesn't follow condition 1. So, none of the cases follow all three condition of regular language as stated by pumping lemma. Therefore, language L is not regular language.

17. Define DFA formally sate and prove closure properties of regular languages. **[2072, Chaitra]**

For formula definition of DFA see theory part.

for statement and proof of closure properties of regular language see 2074, Ashwin, Back part.

18. Define pumping lemma for regular languages. Use pumping lemma for regular language to show L [a^nba^n for n = 0, 1, 2, ...] is not regular. **[2072, Chaitra]**

Pumping lemma states that if a language L is regular then it must have $|s| \geq p$ where p is pumping length and s is the string, $S \in L$ which can be divided into x, y and z parts such that following conditions are true.

(1) $xy^iz \in L$ for every $i \geq 0$.

(2) $|y| > 0$

(3) $|xy| \leq p$

To prove: $L = \{a^nba^n \text{ for } n = 0, 1, 2, ...\}$ is not regular.

Proof: Let's assume language L is regular. Then it must have pumping length 'p'.

Let, $S \in L = q^pba^p$

Let, $p = 3$

$S = a^3ba^3$

$aaabaaa$

Dividing S into x, y and z, we get

Case – I: When 'y' contains only 'a'

$S = \underbrace{a}_{x}\ \underbrace{aa}_{y}\ \underbrace{baaa}_{z}$

Checking for condition I

Let, $i = 2$

$S = xy^2z$

$= a\ aaaa\ bbaaa$

$= a^5ba^3 \notin L$

So, condition 1 fails as the power of a's at the start and end of the string are not equal.

Case – II: When y contains both symbol 'a' and 'b'

$S = \underbrace{aa}_{x}\ \underbrace{ab}_{y}\ \underbrace{aaa}_{z}$

Checking for condition I

Let, $i = 2$

$S = xy^2z$

$= aa\ abab\ aaa \notin L$

So, condition 1 fails as it doesn't follow pattern $a^n\ ba^n$.

Hence, on pumping (i.e. xy^iz), none of the cases follow all three conditions of regular language as stated by pumping lemma. Therefore, language L is not regular.

19. Construct a DFA over {a, b} accepting strings having even number of 'a' and odd number of 'b'. **[2072 Chaitra]**

Let, the required DFA be

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

where,

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

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

$q_0 = \{q_0\}$

$F = \{q_3\}$

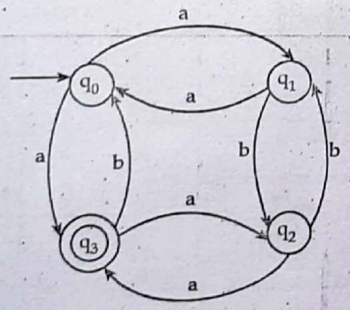
and transition δ is defined as

Transition table (δ)

States/Σ	a	b
$\rightarrow q_0$	q_1	q_3
q_1	q_0	q_2
q_2	q_3	q_1
$*q_3$	q_2	q_0

Let us analyse the language. The given language L generates to strings like {b, bbb, aab, aabbb, ...}

State diagram



Test for string 'bbbaa'

$(q_0, bbbaa) \vdash M(q_3, bbaa)$

$\vdash M(q_0, baa)$

$\vdash M(q_3, aa)$

$\vdash M(q_2, a)$

$\vdash M(q_3, \epsilon)$ Accepted

Since, q_3 is final state, string 'bbbaa' is accepted.

Test for input string' aabb'

$(q_0, aabb) \vdash M(q_1, abb)$

$\vdash M(q_0, bb)$

$\vdash M(q_3, b)$

$\vdash M(q_0, \epsilon)$ Rejected

Since, q_0 is not final state, string 'aabb' is rejected.

20. **Design a deterministic finite automata, (DFA) for the regular expression (a (ab)* b)* verify your design by taking one accepted and one rejected strings.** [2071 Chaitra]

Let, the required DFA be

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

where,

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

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

$q_0 = \{q_0\}$

$F = \{q_0\}$

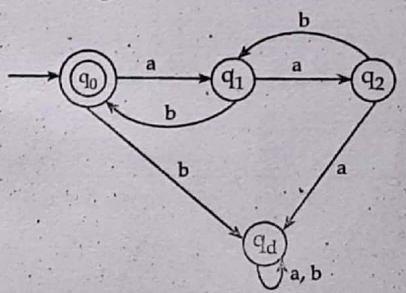
and transition δ is defined as follows

Transition table (δ)

States/Σ	a	b
$\rightarrow *q_0$	q_1	q_d
q_1	q_2	q_0
q_2	q_d	q_1
q_d	q_d	q_d

Let us analyse the language. The given language L generates to strings like {ϵ, ab, abab, aabb, aababb, aabbaabb, ...}

State diagram



Test for string 'aababb'

$(q_0, aababb) \vdash M(q_1, ababb)$

$\vdash M(q_2, babb)$

$\vdash M(q_1, abb)$

$\vdash M(q_2, bb)$

$\vdash M(q_1, b)$

$\vdash M(q_0, \epsilon)$ Accepted

Since, q_0 is final state, string 'aababb' is accepted.

Test for input string 'aabbb'

$(q_0, aabbb) \vdash M(q_1, abbb)$

$\vdash M(q_2, bbb)$

$\vdash M(q_1, bb)$

$\vdash M(q_0, b)$

$\vdash M(q_d, \in)$ Rejected

Since, q_d is dead state or not final state, string 'aabbb' is rejected.

21. State pumping lemma for regular language. Use this lemma to prove language L : $\{a^{n^2} : n \geq 0\}$ is not regular. [2071 Chaitra]

For statement of pumping lemma, see theory part.

Here, given language $L = \{a^{n^2}, n \geq 0\}$

To prove: L is not regular

Proof: Let's assume language L is regular.

Then, it must have pumping length P.

Let, P = 3 (say)

Then, let S be the string accepted by finite automata.

i.e. $S = a^{P^2}$

$= a^{3^2}$

$= a^9$

Dividing S into x, y and z, we get

$S = \underbrace{a}_{x}\ \underbrace{aa}_{y}\ \underbrace{aaaaaa}_{z}$

Now, checking for condition I

$S = xy^iz$

Let, i = 2

$S = xy^2z$

= a aaaa aaaaaa

Now, |S| = 11

Here, no value of P (or n) would give |S| = 11 in other words, length of string S an being pumped is not perfect square which means it doesn't belong to language L.

So, condition 1 fails.

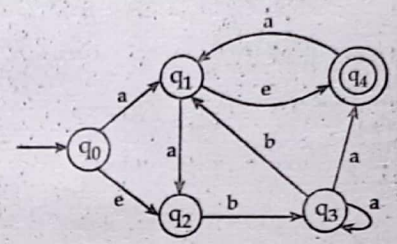
But, condition (2) and (3) are stratified.

i.e. $|y| = 2 > 0$

$|xy| = 3 \geq P$

Since, condition 1 fails, our assumption contradicts. That's why language L is not regular.

22. What are the differences between a DFA and a NFA? convert the following NFA into its equivalent DFA. [2071 Chaitra]



For the differences between a DFA and q NFA se3e theory part.

Here, e-closure $(q_0) = \{q_0, q_2\} = A$ (say)

e-closure $(q_1) = \{q_1, q_4\} = B$ (say)

Transition states

$\delta(A, a) = e\text{-closure}(\delta(q_0, q_2), a)$

$= e\text{-closure}(\delta(q_0, a) \cup \delta(q_2, a))$

$= e\text{-closure}(q_1 \cup \phi)$

$= e\text{-closure}(q_1)$

$= \{q_1, q_4\}$

$= B$

$\delta(A, b) = e\text{-closure}(\delta(q_0, q_2), b)$

$= e\text{-closure}(\delta(q_0, b) \cup \delta(q_2, b))$

$= e\text{-closure}(\phi \cup q_3)$

$= q_3$

$\delta(B, a) = e\text{-closure}(\delta(q_1, q_0), a)$

$= e\text{-closure}(\delta(q_1, a) \cup \delta(q_4, a))$

$= e\text{-closure}(q_2 \cup \phi)$

$= e\text{-closure}(q_2)$

$= \{q_2\}$

$\delta(B, b)$ = e-closure $(\delta(q_1, q_4), b)$

= e-closure $(\delta(q_1, b) \cup \delta(q_4, b))$

= e-closure $(\phi \cup q_3)$

= e-closure (q_1)

= $\{q_1, q_4\}$

= B

$\delta(q_2, a) = \phi = q_D$ (dead state)

$\delta(q_2, b)$ = e-closure $(\delta(q_2, b))$ = e-closure $(q_3) = q_3$

$\delta(q_3, a)$ = e-closure $(\delta(q_3, a))$ = e-closure $(q_3) = q_3$

$\delta(q_3, b)$ = e-closure $(\delta(q_3, b))$ = e-closure $(q_1) = B$

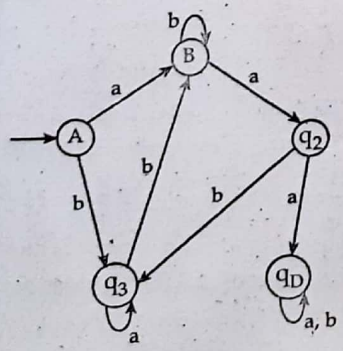
$\delta(q_D, a) = \delta(q_D, b) = q_D$

Now, transition states are

States/Σ	a	b
→ A	B	q_3
* B	q_2	B
q_2	q_D	q_3
q_3	q_3	B
q_D	q_D	q_D

Note: The final states of equivalent DFA are all those new states which contain final state of NFA with ∈-transition as member.

State diagram



This is the required equivalent DFA.

23. Formally define a Deterministic Finite Automata (FA). **[2072, Kartik, Back]**

See formal definition of DFA in theory part.

24. Design a DFA accepting strings over the alphabet {0, 1} defined by {0 0}* {1 1}*. **[2072 Kartik, Back]**

Let, L(M) be the required DFA.

Let us anlayse the language. This language generates strings like {00, 11, 0011, 1111, 0000, 0000111, ...}.

i.e. even number of 0's followed by even number of b's.

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

where, $\theta = \{q_0, q_1, q_2, q_3, q_D\}$

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

$q_0 = \{q_0\}$

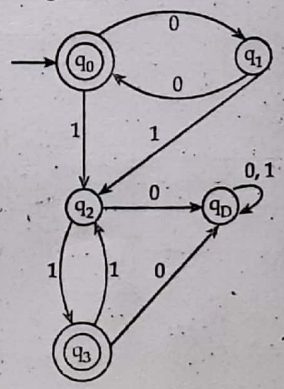
$F = \{q_0, q_3\}$

and transition δ is defined as follows

Transition table

States/Σ	0	1
→ * q_0	q_1	q_2
q_1	q_0	q_2
q_2	q_D	q_3
* q_3	q_D	q_2
q_D	q_D	q_D

State diagram



Test for input string '0011'

$(q_0, 0011) \vdash M(q_1, 011)$
$\vdash M(q_0, 11)$
$\vdash M(q_2, 1)$
$\vdash M(q_3, \in)$ Accepted

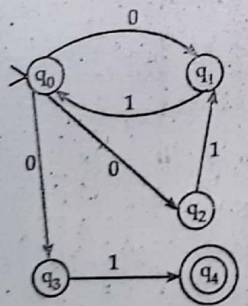
Since, q_3 is final state, string '0011' is accepted.

Test for input string '001100'

$(q_0, 001100) \vdash M(q_1, 01100)$
$\vdash M(q_0, 1100)$
$\vdash M(q_2, 100)$
$\vdash M(q_3, 00)$
$\vdash M(q_D, 0)$
$\vdash M(q_D, \in)$ Rejected

Since, q_D is dead state, string '001100' is not accepted i.e. rejected.

25. Convert the following non-deterministic finite automaton to DFA. **[2072 Kartik, Back]**



Transition table of given NDFA (non-deterministic finite automata)

States/Σ	0	1
q_0	$\{q_1, q_2, q_3\}$	ϕ
q_1	ϕ	q_0
q_2	ϕ	q_1
q_3	ϕ	q_4
q_4	ϕ	ϕ

Transition state

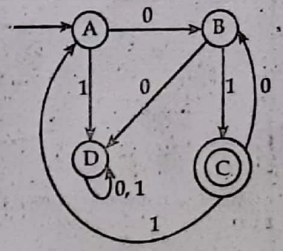
$\delta(q_0, 0) = \{q_1, q_2, q_3\}$
$\delta(q_0, 1) = \phi$
$\delta(\{q_1, q_2, q_3\}, 0) = \delta(q_1, 0) \cup \delta(q_2, 0) \cup \delta(q_3, 0)$
$= \phi \cup \phi \cup \phi$
$= \phi$
$\delta(\{q_1, q_2, q_3\}, 1) = \delta(q_1, 1) \cup \delta(q_2, 1) \cup \delta(q_3, 1)$
$= \{q_0\} \cup \{q_1\} \cup \{q_4\}$
$= \{q_0, q_1, q_4\}$
$\delta(\{q_0, q_1, q_4\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_4, 0)$
$= \{q_1, q_2, q_3\} \cup \phi \cup \phi$
$= \{q_1, q_2, q_3\}$
$\delta(\{q_0, q_1, q_4\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_4, 1)$
$= \phi \cup \{q_0\} \cup \phi$
$= \{q_0\}$

Let, $\{q_0\} = A$
$\{q_1, q_2, q_3\} = B$
$\{q_0, q_1, q_4\} = C$
$\phi = D$ (dead state)

Transition table of equivalent DFA.

States/Σ	0	1
→ A	B	D
B	D	C
* C	B	A
D	D	D

State diagram



The final states of equivalent DFA are all those new states which contains final states of NDFA.

26. State the pumping lemma for regular language show that the language L = {a^n : n is prime} is not regular using the pumping lemma. [2072 Kartik, Back]

For statement of pumping lemma, see theory part.

To prove: L = {a^n : x is a prime} is not a regular

Proof: Let's assume that language L is regular. Then, according to pumping lemma, it must follow these conditions.

(1) $xy^iz \in L$ for every $i \geq 0$

(2) $|y| > 0$

(3) $|xy| \leq P$

Since, language L is regular, it must have pumping length P.

Let P = 7

Let, $S \in L = a^P$

$= a^7$

$=$ aaaaaaa

Dividing S into x, y and z, we get

$S =$ $\underbrace{a}_{x}$ $\underbrace{aa}_{y}$ $\underbrace{aaaa}_{z}$

Checking for condition I

Let i = 2

Then, $S = xy^2z$

$=$ a aaaa aaaa

$= a^9 \notin L$

Since, 9 is not a prime number. Therefore, on pumping i.e. xy^iz, condition 1 fails. Though condition (2) and (3) are satisfied.

$|y| = 2 > 0$

$|xy| = 3 < 7$

Here, assuming language L as regular language doesn't follow all three conditions of pumping lemma to be regular language. So, it contradicts our assumption. Therefore, language L is not regular.

27. Design a DFA that accepts the language given by L = {w ∈ {0, 1}* : w beings with 0 and ends with 10}. Your design should accept strings like 010, 011110, 000010, 01011010, and should not accept strings like 1010, 0011, 01011. [2071 Shrawan, Back]

Let, the required DFA be

$L(M) = (\theta, \Sigma, \delta, q_0, F)$

where,

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

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

$q_0 = \{q_0\}$

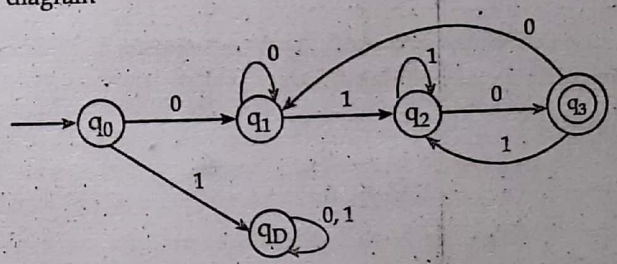
$F = \{q_3\}$

and transition δ is defined as follows

Transition table

States/Σ	0	1
→ q_0	q_1	q_D
q_1	q_1	q_2
q_2	q_3	q_2
* q_3	q_1	q_2
q_D	q_D	q_D

State diagram



Test for accepted string 010, 011110, 000010, 01011010

$(q_0, 010) \vdash M(q_1, 10)$

$\vdash M(q_2, 0)$

$\vdash M(q_3, \epsilon)$ Accepted

Since, q_3 is final state, so, string '010' is accepted.

$(q_0, 011110) \vdash M(q_1, 11110)$
$\vdash M(q_2, 1110)$
$\vdash M(q_2, 110)$
$\vdash M(q_2, 10)$
$\vdash M(q_2, 0)$
$\vdash M(q_3, \epsilon)$ Accepted

Since, q_3 is final state, so string '011110' is accepted

$(q_0, 000010) \vdash M(q_1, 00010)$
$\vdash M(q_1, 0010)$
$\vdash M(q_1, 010)$
$\vdash M(q_1, 10)$
$\vdash M(q_2, 0)$
$\vdash M(q_3, \epsilon)$ Accepted

Since, q_3 is final state, so string '000010' is accepted

$(q_0, 01011010) \vdash M(q_1, 1011010)$
$\vdash M(q_2, 011010)$
$\vdash M(q_3, 11010)$
$\vdash M(q_2, 1010)$
$\vdash M(q_2, 010)$
$\vdash M(q_3, 10)$
$\vdash M(q_2, 0)$
$\vdash M(q_3, \epsilon)$ Accepted

Since, q_3 is final state, so string '01011010' is accepted

Test for rejected strings 1010, 0011, 01011

$(q_0, 1010) \vdash M(q_D, 010)$
$\vdash M(q_D, 10)$
$\vdash M(q_D, 0)$
$\vdash M(q_D, \epsilon)$ Rejected

Since, q_3 is dead state, so string '1010' is rejected.

$(q_0, 0011) \vdash M(q_1, 011)$
$\vdash M(q_1, 11)$
$\vdash M(q_2, 1)$
$\vdash M(q_2, \epsilon)$ Rejected

Since, q_2 is not final state, so string '0011' is rejected.

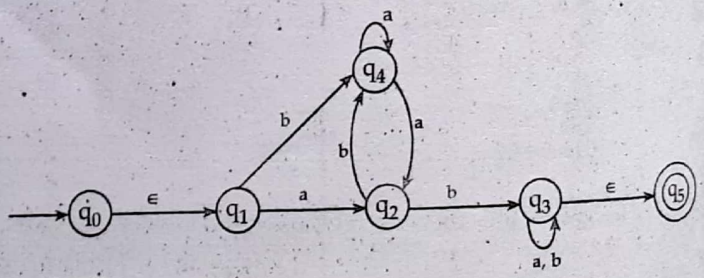
$(q_0, 01011) \vdash M(q_1, 1011)$
$\vdash M(q_2, 011)$
$\vdash M(q_3, 11)$
$\vdash M(q_2, 1)$
$\vdash M(q_2, \epsilon)$ Rejected

Since, q_2 is not final state, so string '01011' is rejected.

28. Find the regular expression represented by NFA M = (k, Σ, Δ, S, F), where K = {q_0, q_1, q_2, q_3, q_4, q_5}, S = q_0, F = {q_5} and Δ is given as follows: [2071 Shrawan, back]

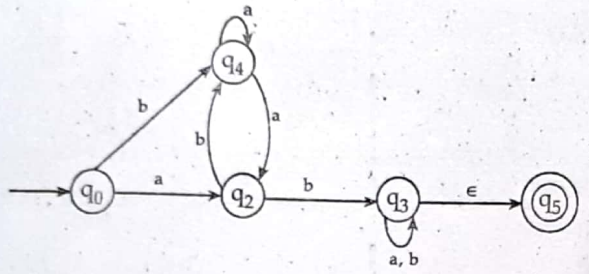
δ/Σ	a	b	∈
→ q_0	–	–	q_1
q_1	q_2	q_4	–
q_2	–	q_3, q_4	–
q_3	q_3	q_3	q_5
q_4	q_2, q_4	–	–
*q_5	–	–	–

Drawing equivalent states diagram of the given Δ,

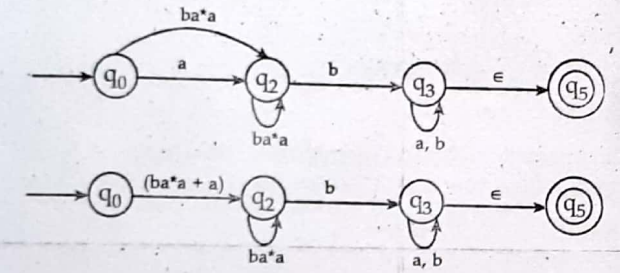


Here, q_0 is initial state and q_5 is final state. Eliminating intermediate states like q_1, q_2, q_3 and q_4 with their equivalent regular expression give regular expression on which we reach from initial state to final state and that will be our required regular expression.

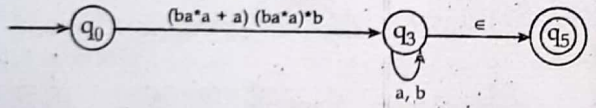
Step – I: Eliminating q_1



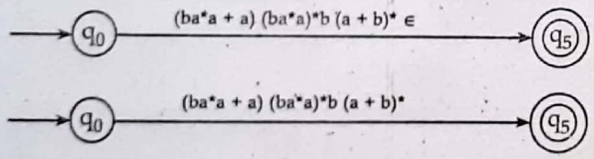
Step – II: Eliminating q_4



Step – III: Eliminating q_2



Step – IV: Eliminating q_3



Therefore, the required expression (R) is

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

29. Explain about decision algorithms for regular language.

[2071 Shrawan, Back]

The decision algorithms / properties of regular language are membership problem, emptiness problem and equivalence problems. And they are discussed below:

Algorithm 1: Given a regular language L over T and $w \in T^*$, there exists an algorithm for determining whether or not w is in L.

Let L be accepted by a DFA M(say). Then, for input w one can see whether w is accepted by M or not. The complexity of this algorithm is 0(n) where $|w| = n$.

Hence, membership problem for regular sets can be solved in linear time.

Algorithm – 2: There exists an algorithm for determining whether a regular language L is empty, finite or infinite. (Emptiness algorithm).

Let M be a DFA accepting L. In the state diagram representation of M with inaccessible states from the initial state removed, one has to check whether there is a simple directed path from initial state of M to a final state. If so, L is not empty. Consider a DFA, M' accepting L, where inaccessible state from the initial states are removed and also states from which a final state can't be reached are removed.

If in the graph of the state diagram of DFA, there are no cycles, then L is finite. Otherwise L is infinite.

One can see that the automaton accepts sentence of length less than n (where n is the number of states of the DFA) if and only if L(M) is non-empty. One can prove this statement using pumping lemma. That is $|w| < n$ for if w where the shortest and $|w| > n$ then $w = xyz$ and xz is shorter than w that belongs to L.

Also, L is infinite if and only if the automaton M accepts at least one word of length l where $n \leq l < 2n$. One can prove one can prove this by using pumping lemma. If $w \in L(M)$, $|w| \geq n$ and $|w| < 2n$, directing from pumping lemma, L is infinite.

Conversely, if L is infinite, we show that there should be a word in L whose length is l where $n \leq l < 2n$. If there is no word whose length is l where $n \leq l$ 2n let w be the word whose length is l where $n \leq l < 2n$. Let w be the word whose length at least 2n, but as short as any word in L(M) whose length is greater than or equal to 2n. Then by pumping lemma, $w = w_1\ w_2\ w_3$ where $1 \leq |w_2| \leq n$ and $w_1w_2 \in L(M)$. Hence, either w was not the shortest word of length 2n or more or $|w_1\ w_3|$ is between n and 2n – 1, which is a contradiction.

Algorithm 3: For any two regular languages L_1 and L_2, there exists an algorithm to determine whether or not $L_1 = L_2$ (Equivalence algorithm)

Consider $L = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$. Clearly, L is regular by closure properties of regular languages. Hence, there exists a DFA M which accepts L. Now, by the previous algorithm, one can determine whether L is empty or not L is empty if and only if $L_1 = L_2$. Hence, the theorem.

30. Prove that language which contains set of strings of balanced parenthesis is not regular. [2071 Shrawan, Back]

Let us suppose language is regular, and n be a constant according to pumping lemma. We can divide each string in three parts xyz as $|xy| \leq n$ and $y \neq \in$. So, xy^iz must be in L for $i \geq 0$.

Observation it is clear that language will contains the strings like (( ) ( )), ( ) ( ) ( ), (((( )))), ...

Let us select a string w = (((.. ( ) ...))), where there are n left parenthesis by n right parenthesis. this string is in L, and is of length at least n. So, we can divide w in three parts $w \simeq xyz$ as $|xy| \leq x$ and $y \neq \in$.

Conclusion: Not let us choose i = 0. The resulting xz is not in L reason is very clear. Since $w \simeq xyz\,|xy| < n$.

It means y will be made of only left parenthesis. So xz is not in L since it has fewer left parenthesis than right. This contradicts the pumping lemma, so our original assumption, that L was regular, must have been incorrect.

31. Design a DFA that accepts the language L = {x ∈ {0, 1}* : x has an even number of 0's and an even number of 1's {verify your design for at least two strings that are accepted by this DFA and 2 strings that are rejected.]2070 Chaitra]

Let, the required DFA be

$$L(M) = (\theta, \Sigma, \delta, q_0, F)$$

Let us analyse the language L. This language generates strings like {00, 11, 001100, 1100, ...}

Here,

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

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

$q_0 = \{q_0\}$

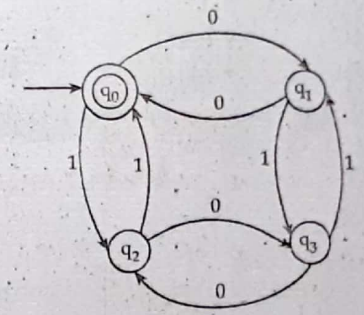
$F = \{q_0\}$

and transition (δ) is defined as follows

Transition table

States/Σ	0	1
→ * q_0	q_1	q_2
q_1	q_0	q_3
q_3	q_2	q_1
q_2	q_3	q_0

State diagram



Accepted strings

Test for input string '001001'

$(q_0, 001001) \vdash M(q_1, 01001)$
$\vdash M(q_0, 1001)$
$\vdash M(q_2, 001)$
$\vdash M(q_3, 01)$
$\vdash M(q_2, 1)$
$\vdash M(q_0, \in)$ Accepted

Since, q_0 is final state, string '00111' is accepted

$(q_0, 100111) \vdash M(q_2, 00111)$
$\vdash M(q_3, 0111)$
$\vdash M(q_2, 111)$
$\vdash M(q_0, 11)$
$\vdash M(q_2, 1)$
$\vdash M(q_0, \in)$ Accepted

Since, q_0 is final state, string '100111' is accepted

Rejected strings

$(q_0, 100) \vdash M(q_2, 00)$
$\vdash M(q_3, 0)$
$\vdash M(q_2, \in)$ Rejected

Since, q_2 is not final state, string '100' is not accepted i.e. rejected.

Test for input string '01001'

$(q_0, 01001) \vdash M(q_1, 1001)$
$\vdash M(q_3, 001)$
$\vdash M(q_2, 01)$
$\vdash M(q_3, 1)$
$\vdash M(q_1, \in)$ Rejected

Since, q_1 is not final state, string '01001' is not accepted i.e. rejected.

32. Show that for any regular expression R, there is a NFA that accepts the same language represented by R. Construct a e-NFA for regular expression bb (a ∪ b)* ab. [2070 Chaitra]

R is obtained from a, (a ∈ Σ, ∈, ϕ by finite number of applications of 'U', ' . ' and '*'. (. is usually left out).

For ∈, ϕ and a we can construct NFA with ∈-moves as shown in fig (i)

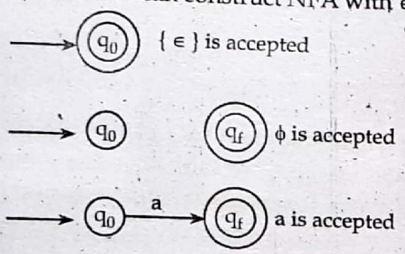


Fig. (i) NFA for ∈, ϕ for and a

Let R represents the regular set R_1 and R_1 is accepted by NFA M_1 with ∈-transitions fig (ii).

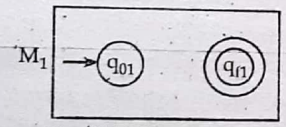


Fig. (ii) M_1 accepting R_1

Without less of generality, we can assume that each such NFA with ∈-moves has only one final states.

R_2 is similarly accepted by a NFA M_2 with ∈-transitions fig (iii)

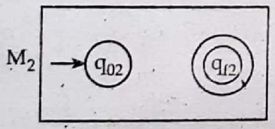


Fig. (iii) M_2 accepting R_2

Now, we can easily see that $R_1 \cup R_2$ is accepted by NFA given in fig. (iv).

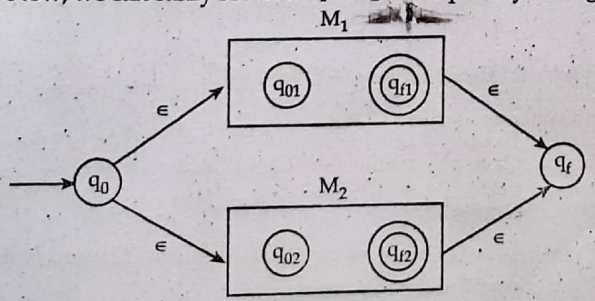


Fig. (iv)

Fig (iv) NFA for $R_1 \cup R_2$

For this NFA, q_0 is the 'start' state and qf is the 'end' state.

R_1R_2 is accepted by the NFA with ∈-moves given in fig (v)

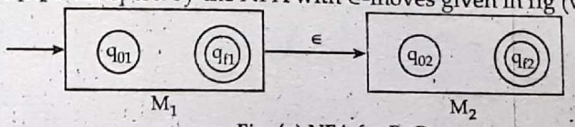


Fig. (v) NFA for R_1 R_2

For this NFA with ∈-moves, q_{01} is the initial state and q_{f2} is the final state.

$R_1^* = R_1^0 \cup R_1^1 = R_1^2 \cup \ldots R_1^k \cup \ldots$

$R_1^0 \{\in\}$ and $R_1^1 = R_1$

R_1^* is accepted by the NFA with e-moves given in fig (vi)

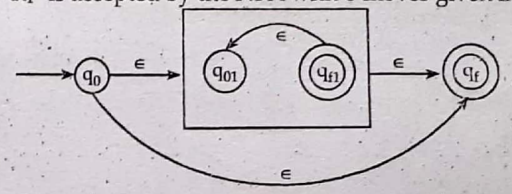


Fig. (vi) NFA for R_1^*

For this NFA with ∈-moves, q_0 is the initial state and qf is the final state it can be seen that R_1^* contains strings of the form $x_1x_2 \ldots x_k$ each $x_1 \in R_1$. To accept this string, the control goes from q_0 to q_{01} and then after reading x_1 and reaching q_{f1}, it goes to q_{01}, by an ∈-transition. From q_{01}, it again reads x_2 and goes to q_{f1}. This can be repeated a number of (k) times and finally the control goes to q_f from q_{f1} by an ∈-transition $R_1^0 = \{\in\}$ is accepted by going to q_f from q_0 by an ∈-transitions.

Thus, we have seen that for a given regular expression one can construct an equivalent NFA with ∈-transitions.

We know that we can construct an equivalent NFA without ∈-transitions from this, and can construct a DFA as show in figure below.

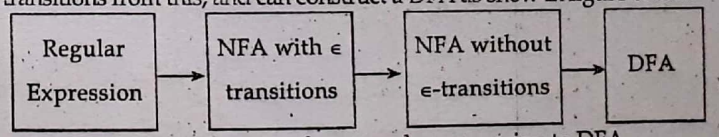
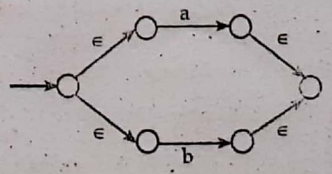


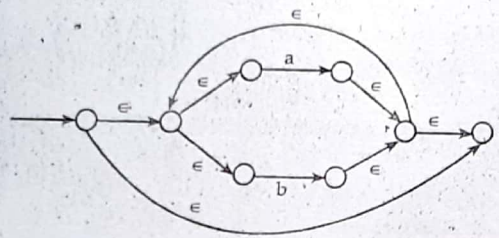
Fig. Transformation from regular expression to DFA.

Now, construction of regular expression bb(a ∪ b)* ab.

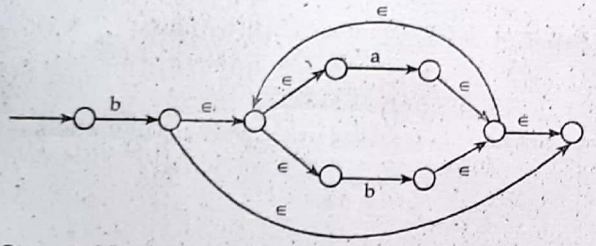
Step – 1: We first construct (a ∪ b)



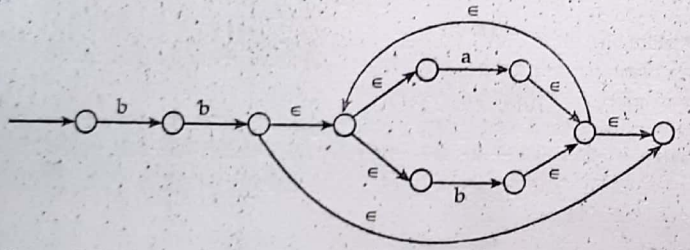
Step – 2: Then construct $(a \cup b)^*$



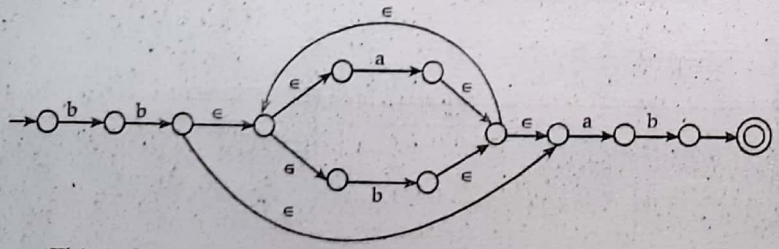
Step – 3: Now construct $b(a \cup b)^*$



Step – 4: Now construct $bb(a \cup b)^*$



Step – 5: In the similar manner, we can construct $bb(a \cup b)^* ab$.



This is the required e-NFA for regular expression $bb(a \cup b)^* ab$.

33. Use pumping lemma to prove that $L = \{a^n b^{2n} : n \geq 1\}$ is not regular.

Please see 2073 Shrawan.

♦♦♦

Chapter - 3

Context Free Grammar

Introduction

All of us know that a grammar is nothing but a set of rules to define any sentences in any languages. In this chapter, we introduce the context free grammar, which generate context free languages. Context free languages have great practical significance in defining programming languages and in simplifying the translation for programming languages.

Initially linguists were trying to define precisely valid sentences and to give structural description for these sentences. They tried to define rules for natural languages. Noam Chomsky gave a mathematical idea model for the grammars in 1956. Although it was useless to describe natural languages but it become very useful for computer languages. The original motivation for grammars was the description of natural languages. We can write rules for the grammar of natural languages as follows:

<sentence> – <noun phrase> <verb phrase>

<noun article> – < article> <noun>

According to above set of rules "we are fine" is valid sentence.

Formal definition of context free grammars

A context free grammar G is a quadruple defined as follows:

$G = (V, \Sigma, R, S)$

where

v is an alphabet (set of terminals and non-terminal

Σ (the set of terminals) is a subset of v.

R (the set of rules) is a finite subset of $(v - \Sigma) \times v^*$ and

S (the start symbol) is an element

Note: Terminals are generally denoted by small letters. non-terminals or variables are generally denoted by capital letters.

Example of CFG (context-free grammar)

Given a grammar G = ({S}, {a, b}, R, S)

Here,

v = {S}

Σ = {a, b}

The set of rules R is

$S \rightarrow a\ S\ b$

$S \rightarrow S\ S$

$S \rightarrow \epsilon$

This grammar generates strings such as abab, aaabb, aababb, etc.

Mathematical definition of context free grammar

Mathematically, context free grammar is defined as follws:

Definition: "A grammar $G = \{V_n, V_t, P, S)$ is said to be context free.

where,

V_n: A finite set of non-terminals, generally represented by capital letters A, B, C and D.

V_t: A finite set of terminals generally represented by capital letters like a, b, c, d, e, f.

S: Starting non-terminal, called start symbol of the grammar S belongs to V_n.

P: Set of rules or productions in CFG.

G is context free and all productions in P have the form $\alpha \rightarrow \beta$.

$\alpha \in V_n$ and $\beta \in (V_n \cup V_t)^*$

Examples

1. Write CFG for the language $L = \{wcw^R / w \in (a, b)^*\}$

Solution:

Let, G be the required CFG and is defined as

$G = (V, \Sigma, R, S)$

where, V = {S, a, b}

Σ = {a, b}

R = {

$S \rightarrow a\ S\ a$

$\rightarrow b\ S\ b$

$\rightarrow C$

}

Now, to derive any string $S \in L$, we use productions of R.

Let, S = aabcbaa

We start with S

$S \Rightarrow a\ S\ a$

$\Rightarrow aa\ S\ aa$ $\quad [S \rightarrow a\ S\ a]$

$\Rightarrow aab\ S\ baa$ $\quad [S \rightarrow bsb]$

$\Rightarrow aabc\ baa$ $\quad [S \rightarrow c]$

So, string aabcbaa can be derived from G.

2. Write a CFG which generates string of balanced parenthesis.

Solution:

Let G be the required CFG defined as follows:

$G = (V, \Sigma, R, S)$

where, V = {S, (,)}

Σ = {(,)}

R = {

$S \rightarrow SS$

$S \rightarrow (S)$

$S \rightarrow \epsilon$

}

Let, S = ( ) ( ) to be derived from CFG G.

$S \Rightarrow S\ S$

$\Rightarrow (S)\ S$

$\Rightarrow (\ )\ S$

$\Rightarrow (\ )\ (S)$

$\Rightarrow (\ )\ (\ )$

3. Write a CFG for the regular expression $r = 0^*\ 1\ (0 + 1)^*$.

Solution:

Given, regular expression $r = 0^*\ 1\ (0 + 1)^*$. This regular expression r generates strings containing any number of 0's followed by single 1 and ending with any combination of 0's and 1's.

Let, the required CFG G be

$G = (V, \Sigma, R, S)$

where V = {S, A, B, 0, 1}

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

R = {

$S \rightarrow A\,1\,B$

$A \rightarrow 0A / \in$

$B \rightarrow 0B / 1B / \in$

}

Let, S = 00101 to be generated by required CFG'G.

$S \Rightarrow A\,1B$

$\Rightarrow 0A\,1B \quad [A \rightarrow 0A]$

$\Rightarrow 00A1\,B \quad [A \rightarrow 0A]$

$\Rightarrow 001\,B \quad [A \rightarrow \in]$

$\Rightarrow 001\,0B \quad [B \rightarrow 0B]$

$\Rightarrow 001\,01B \quad [B \rightarrow 1B]$

$\Rightarrow 001\,01 \quad [B \rightarrow \in]$

∴ So, clearly G is CFG for regular expression r.

4. Write a CFG for $\Sigma = \{a, b\}$ that generates the set of

(a) all strings with exactly one a

(b) all strings with at least one a

(c) all strings with at least 3 a's

Solution:

Let, the required CFG G be

G = (V, Σ, R, S)

where

(a) all strings with exactly one a.

V = {S, A, a, b}

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

R = {

$S \rightarrow Aa\,A$

$A \rightarrow bA / \in$

}

Let, S = bbabb to be generated, then

$S \Rightarrow A\,a\,A$

$\Rightarrow b\,A\,a\,A$

$\Rightarrow b\,b\,A\,a\,A$

$\Rightarrow b\,b\,a\,A$

$\Rightarrow b\,b\,a\,b\,A$

$\Rightarrow b\,b\,a\,b\,b\,A$

$\Rightarrow b\,b\,a\,b\,b$

(b) All strings with at least one A

V = {S, A, a, b}

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

R = {

$S \rightarrow Aa\,A$

$A \rightarrow aA / bA / \in$

}

Let, S = baba to be generated, then

$S \Rightarrow A\,a\,A$

$\Rightarrow b\,A\,a\,A$

$\Rightarrow b\,a\,A$

$\Rightarrow b\,a\,b\,A$

$\Rightarrow b\,a\,b\,a\,A$

$\Rightarrow b\,a\,b\,a$

(c) All strings with at least 3 a's, then

V = {S, A, a, b}

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

R = {

$S \rightarrow AaAaAaA$

$A \rightarrow aA / bA / \in$

}

Let, S = bababa to be generated by CFG.

S ⇒ AaAaAaA
⇒ bAaAaAaA
⇒ baAaAaA
⇒ babAaAaA
⇒ babaAaA
⇒ bababAaA
⇒ bababaA
⇒ bababa

5. Design a CFG for the language L = {$a^n b^{2n}$ / n >= 0}

Solution:

Let, the required CFG 'G' be

G = (V, Σ, R, S)

where, V = {S, a, b}

Σ = {a, b}

R = {

S → aSbb / ∈

}

Let, string S = aabbbb to be generated

S ⇒ aSbb
⇒ aaSbbbb
⇒ aabbbb

Left most and right most derivations

In derivation of strings consisting of many variables in any CFG, we could choose replacement of variables from any position in string on the basis of position or side of variables to be replaced by one of its bodies, the string derivation are of following types:

Leftmost derivation: If we replace the "left most" variable by one of its production bodies such a derivation is called a left most derivation.

Let, CFG be

G = (V, Σ, R, S)

where,

V = {S, a, c,), +, ×}

Σ = {a, c,), ×, +}

R = {

S → S × S
→ S + S
→ a
→ (S)

}

Let us derive a × (a + a)

$S \xRightarrow[lm]{} S \times S$
$\xRightarrow[lm]{} a \times S$
$\xRightarrow[lm]{} a \times S$
$\xRightarrow[lm]{} a \times (S + S)$
$\xRightarrow[lm]{} S \times (a + S)$
$\xRightarrow[lm]{} a \times (a + a)$

Its symbolic representation is $\xRightarrow[lm]{}$

Rightmost derivation: If we replace the "right most" variable by one of its production bodies such a derivation is called a right most derivation. Let us generate S = a × a + a) from above CFG.

$S \xRightarrow[rm]{} S \times S$
$\xRightarrow[rm]{} S \times (S)$
$\xRightarrow[rm]{} S \times (S + S)$
$\xRightarrow[rm]{} S \times (S + a)$
$\xRightarrow[rm]{} S \times (a + a)$
$\xRightarrow[rm]{} a \times (a + a)$

Let symbolic representation is $\xRightarrow[rm]{}$

Mixed derivation: If we replace any variable from any position and any side by one of its production bodies such a derivation is mixed derivation.

$$\begin{aligned} S &\Rightarrow S \times S \\ &\Rightarrow S \times (S) \\ &\Rightarrow S \times (S + S) \\ &\Rightarrow S \times (a + S) \\ &\Rightarrow a \times (a + S) \\ &\Rightarrow a \times (a + a) \end{aligned}$$

Its symbolic representation is $\Rightarrow$

Parse tree/ derivation tree

A parse tree is an ordered tree in which the nodes are labeled with left side of productions and in which the children of nodes represent the corresponding right sides.

Let, $G = (V_n, V_t, P, S)$ be a context free grammar. An ordered tree for this CFG is is a derivation tree if any only if it has following properties.

(a) The root is labeled by the starting non-terminal of the CFG that is S.

(b) Every leaf of the ordered tree has a label from $V_t \cup \{\in\}$.

(c) Every interior node of ordered tree has a label from V_n.

(d) Let us assume that a vertex has label $X \in V_n$ and its children are labeled (from left to right) $y_1, b_2, y_3, \ldots y_n$ then production must contain a product of the form

$$X \rightarrow y_1, y_2, \ldots y_n$$

(e) A left labeled $\in$ has no siblings, that is vertex with a child labeled $\in$ can have no other children.

Examples

1. Write the CFG for the language $L = \{x\, 0^n\, y\, 1^n\, z / x \geq 0\}$ and given the parse tree for the string x000, y11z.

Solution:

Let, the CFG for the language L is

where, $G = (V, \Sigma, R, S)$

where, $V = \{S, B, x, y, z, 0, 1\}$

$\Sigma = \{x, y, z, 0, 1\}$

and production p is defined as

$S \rightarrow x\, Bz$

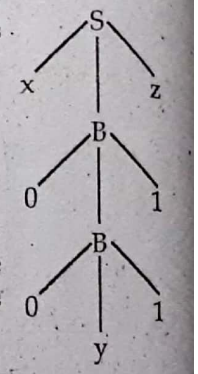
$B \rightarrow y \,/\, 0B1$

Now, construct an arbitrary deviation

$$S \overset{*}{\underset{G}{\Rightarrow}} x00y11z$$

One possible derivation using the above grammar is

$$\begin{aligned} S &\Rightarrow x\,\underline{B}\,z \\ &\Rightarrow x\,0\,\underline{B}\,1\,z \\ &\Rightarrow x\,0\,0\,\underline{B}\,1\,1\,z \\ &\Rightarrow x\,0\,0\,y\,1\,1\,z \end{aligned}$$



Now, we are able to design the parse tree for the string x 000y 11z from the above CFG. Parse tree is shown in figure.

2. Let G be CFG

$S \rightarrow bB / aA$

$A \rightarrow b / bS / aAA$

$B \rightarrow a / as / bBB$

For the stirng bbaababa find

(a) left most derivation

(b) right most derivation

(c) parse tree

Solution:

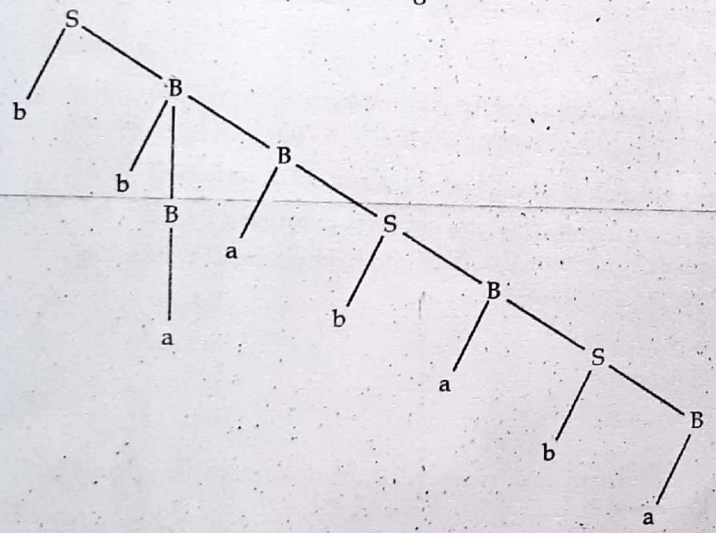
(a) Left most derivation for string w = bbaababa is

$$\begin{aligned} S &\Rightarrow b\underline{B} \\ &\Rightarrow bb\underline{B}B \\ &\Rightarrow bba\underline{B} \\ &\Rightarrow bbaa\underline{S} \\ &\Rightarrow bbaab\underline{B} \\ &\Rightarrow bbaaba\underline{S} \\ &\Rightarrow bbaababB \\ &\Rightarrow bbaababa \end{aligned}$$

(b) The rightmost derivation is

S ⇒ bB
⇒ bbBB
⇒ bbBaS
⇒ bbBabB
⇒ bbBababB
⇒ bbBababa
⇒ bbaababa

(c) The derivation tree is following



Yield is bbaababa

3. Consider the grammar

S → aB / bA

A → aS / bAA / a

B → bS / aBB / b

For the string aaabbabbba find

(a) The left most derivation and find left most derivation tree.

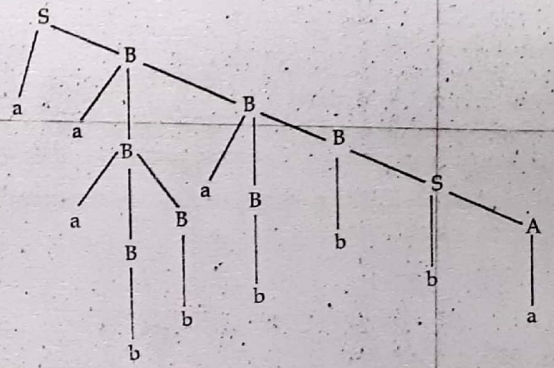
(b) The right most derivation and right most derivation tree.

Solution:

Given string is aabba bbba

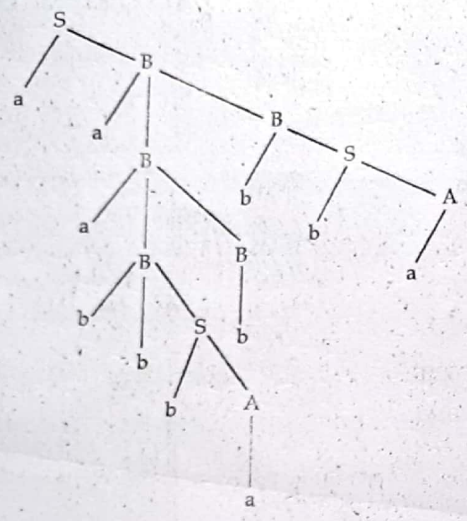
(a) Left most derivation

S ⇒ aB
⇒ aaBB
⇒ aaaBBB
⇒ aaabBB
⇒ aaabbB
⇒ aaabbaBB
⇒ aaabbabB
⇒ aaabbabbS
⇒ aaabbabbbA
⇒ aaabbabbba



(b) Rightmost derivation

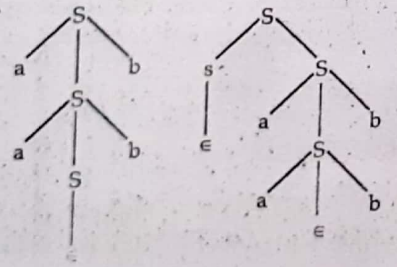
S ⇒ aB
⇒ aaBB
⇒ aaBbS
⇒ aaBbbA
⇒ aaBbbba
⇒ aaaBBbba
⇒ aaaBbbba
⇒ aaabSbbba
⇒ aaabbAbbba
⇒ aaabbabbba



Ambiguous grammars

Let, a grammar G = {V, Σ, R, S) be a CFG. A grammar G is said to be ambiguous if any string S ∈ L(G) has two or more than two leftmost, rightmost derivation or parse tree for its deviation.

For example, consider the CFG S → aSb / SS / ∈. The string aabb has two derivation trees as shown below.



Since, there are two parse tree for same string S = aabb, this grammar (CFG) is said to be ambiguous.

Inherently ambiguous language

A "inherently ambiguous language" is a language for which no unambiguous grammar exists or,

A language L is said to be "inherently ambiguous" if every grammar that generates L is ambiguous. Otherwise, it is said to be unambigous language.

Simplification of context free grammar

There may be more than one CFG 'G' for the same language L. And all the CFG may not be simplified (i.e. may contain useless symbols, null productions (∈) and unit production. And for simplified CFG, we need to remove these conditions.

(a) Elimination of useless symbols

Useless symbols are of two types.

(i) **Non-generating symbols:** If a symbol that doesn't produce some set of terminals symbol, then it is non-generating symbol consider a grammar as

S → ABa / BC

B → bcc

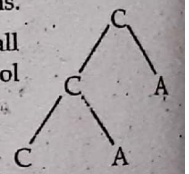
C → CA

D → E

A → ac / Bcc / a

E → ∈

Here, symbol 'C' doesn't produce any set of terminals.



It is never terminating. So, we need to remove the all symbols containing 'C'. After removal of all symbol 'C' we get grammar as

S → ABa

B → bcc

D → E

A → a / Bcc

E → ∈

(ii) **Not-reachable symbol:** If a symbol that can't be produced from start symbol 'S', it is said to be not-reachable symbol. For example; in above grammar, there is no way to reach D and E directly or indirectly from any of the variables on the right hand side of start symbol 'S'. So, there need to be removed. After removal or simplification, we get

S → ABa

B → bcc

A → Bcc/a

This is the simplified grammar.

(b) Elimination of null production

If a production A has the form $A \to \epsilon$, then it is called null-productions or ϵ-productions. Surely if ϵ is in L(G), we can't eliminate ϵ production from G bat if ϵ is not L(G), we can eliminate all productions from G.

Examples

1. **Consider the following grammar G**

$S \to aA$

$A \to b / \epsilon$

remove the Nullable non-terminal

Solution:

Here, $A \to \epsilon$ is null production. So, put ϵ in place of A at the right side of productions and add the resulted productions to the grammar.

Here is only one production $S \to aA$ whose right side contains A. So, replacing A by ϵ, we get $S \to a$, we get simplify grammar G as

$S \to aA / a$

$A \to b$

2. **Consider the following grammar G**

$S \to ABAC$

$A \to aA / \epsilon$

$B \to bB / \epsilon$

$C \to c$

and remove ϵ production from the above grammar.

Solution:

Here two ϵ productions are

$A \to \epsilon$

$B \to \epsilon$

First removing $A \to \epsilon$, we replace A by ϵ in every production containing A on right hand side and adding the resulted production, we get

$S \to ABAC / ABC / BAC / BC$

$A \to aA / a$

$B \to bB / \epsilon$

$C \to c$

Now, removing $B \to \epsilon$, we get

$S \to ABAC / BAC / ABC / BC / AAC / C / AC$

$A \to aA / a$

$B \to bB / b$

$C \to c$

(c) Removal of unit production

A production of the form

Non terminal → one non-terminal that is a production of the form $A \to B$ (where A and B, both are non-terminals) is called unit production.

Algorithm: Removal of unit production

While (there exists a unit production, $A \to B$)

{

select a unit production $A \to B$, such that there exists a production $B \to \alpha$, where α is a terminal.

For (every non-unit production, $B \to \alpha$

Add production $A \to \alpha$ to the grammar

Eliminate $A \to B$ from the grammar

}

For example

Consider the context free grammar G

$S \to AB$

$A \to a$

$B \to C / b$

$C \to D$

$D \to E$

$E \to a$

Solution:

Here, three unit productions are present

$B \to C$

$C \to D$

$D \to E$

Consider, that unit production, whose right side variable has a terminal production

$D \rightarrow E$

$E \rightarrow a$

Assign non terminal 'a' from production 'E' which is the production of D so,

$D \rightarrow a$

$E \rightarrow a$

Similar things holds for production $C \rightarrow D$ and $D \rightarrow a$ i.e.

$C \rightarrow a$

$D \rightarrow a$

$E \rightarrow a$

Now, removing last unit production $B \rightarrow C$, we get

$B \rightarrow a / b$

$C \rightarrow a$

$D \rightarrow a$

$E \rightarrow a$

After removing all unit production, we have

CFG G as

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow a / b$

$C \rightarrow a$

$D \rightarrow a$

$E \rightarrow a$

Now, it can be easily seen that productions $C \rightarrow a$, $D \rightarrow a$, $E \rightarrow a$ are useless because if we start deriving from S, these production will never be used. Hence, eliminating them gives

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow a / b$

which is completely reduced grammar.

Chomsky normal form (CNF)

A grammar is in Chomsky normal form if all productions are of the form

$A \rightarrow BC$

or, $A \rightarrow a$

where A, B and C are variables and 'a' is terminal. Any context free grammar that doesn't contain $\in$ can be put into Chomsky normal form.

Procedure to find equivalent grammar in CNF

(a) Eliminate the useless grammar (non generating and non-reachable symbols).

(b) Eliminate the unit productions and $\in$-productions.

(c) Eliminate the terminals on the right hand side of length two or more.

(d) Restrict the number of productions to two variables.

The equivalent grammar in CNF on be understood by following examples:

Examples

1. **Consider a context free grammar G as**

$S \rightarrow ABA / BaA / A$

$A \rightarrow Ba / S / \in$

$B \rightarrow Ba / b / Ca$

$C \rightarrow Ca$

$D \rightarrow Da\ D / a$

Solution:

Steps followed in CNF

(a) Eliminate useless symbols

(b) Eliminate null ($\in$) production

(c) Eliminate unit production.

Step – I: Here,

C is a useless symbol as it doesn't generate any kind of terminal symbol entirely. In other words, it generates combination of terminal 'a' any non-terminal 'C'. "C' is non-generating symbol and 'D' is not reachable symbol. Here, reachable symbols from start symbol are as follows:

$S \rightarrow A, B$

$A \rightarrow B, S$

$B \rightarrow B, C$

$\therefore$ D is not reachable symbol.

Eliminating all useless symbol (non-generating and not-reachable symbols from above grammar, we get

$S \rightarrow ABA / BaA / A$

$A \rightarrow Ba / S / \epsilon$

$B \rightarrow Ba / b$

Step – II: Elimination of null productions.

Here, production A is null production as it generates null symbol (epsilon)

$A \rightarrow \epsilon$

Now, eliminating null production ($A \rightarrow \epsilon$), we get

$S \rightarrow ABA / BA / AB / B / BaA / Ba / A / \epsilon$

$A \rightarrow Ba / S$

$B \rightarrow Bb / b$

Here, again new null production $S \rightarrow \epsilon$, now we have to remove this null production S as well then grammar is

$S \rightarrow ABA / BA / AB / B / BaA / Ba / A$

$A \rightarrow Ba / S$

$B \rightarrow Ba / b$

Here, in production of $A \rightarrow \epsilon$ is not written as we have already removed $A \rightarrow \epsilon$ production

Step – III: Elimination of unit production

Here, we have three unit production

$A \rightarrow S$

$S \rightarrow A$

$S \rightarrow B$

First remove, $A \rightarrow S$

$S \rightarrow ABA / BA / AB / B / BaA / Ba / A$

$A \rightarrow Ba / ABA / BA / AB / B / BaA$

$B \rightarrow Ba / b$

Secondly, remove $S \rightarrow A$

$S \rightarrow ABA / AB / BA / B / BaA / Ba$

$A \rightarrow Ba / ABA / BA / AB / B / BaA$

$B \rightarrow Ba / b$

Lastly, remove $S \rightarrow B$ and $A \rightarrow B$, we get

$S \rightarrow ABA/ BA / AB / b / BaA / Ba$

$A \rightarrow Ba / ABA / BA / AB / b / BaA$

$B \rightarrow Ba / b$

Now, finally conversion to Chomsky normal form,

Let, $C_a \rightarrow a$

Then,

$S \rightarrow ABA / BA / AB / b / BC_aA / BC_a$

$A \rightarrow B C_a / ABA / BA / AB / b / BC_aA$

$B \rightarrow BC_a / b$

$C_a \rightarrow a$

For, $S \rightarrow ABA$, we have

$S \rightarrow DA$

$D \rightarrow AB$

For, $S \rightarrow B C_a A$, we have

$S \rightarrow EA$

$E \rightarrow B C_a$

Then,

$S \rightarrow DA / BA / AB / b / EA / B C_a$

$A \rightarrow B C_a / DA / BA / AB / b / EA$

$B \rightarrow BC_a / b$

$C_a \rightarrow a$

$D \rightarrow AB$

$E \rightarrow B C_a$

This is the required Chomsky normal form.

2. Obtain a grammar in Chomsky normal form (CNF) equivalent to the grammar G with production P given by

$S \rightarrow ABa$

$A \rightarrow aab$

$B \rightarrow Ac$

Solution:

Here, the given set P doesn't have any unit productions or ϵ productions

Let, $C_a \to a$, then

$S \to AB\,C_a$

$A \to C_a\,C_a b$

$B \to A_c$

Let, $C_c \to C$, then

$S \to AB\,C_a$

$A \to C_a\,C_a\,b$

$B \to A\,C_c$

Let, $C_b \to b$, then

$S \to ABC_a$

$A \to C_a\,C_a\,C_b$

$B \to AC_c$

For, $S \to ABC_a$, we have

$S \to D_1\,C_a$

$D_1 \to AB$

For, $A \to C_a\,C_a\,C_b$, we have

$A \to D_2\,C_b$

$D_2 \to C_a\,C_a$

Therefore, G has a new set of production P' given by

$S \to D_1\,C_a$

$A \to D_2\,C_b$

$B \to A\,C_c$

$D_1 \to AB$

$D_2 \to C_a\,C_a$

$C_a \to a$

$C_b \to b$

$C_c \to c$

3. Convert CFG which is given below into CNF form.

$S \to bA / aB$

$A \to bAA / aS / a$

$B \to aBB / bS / b$

Solution:

Let us replace b by C_b and a by C_a, then CFG becomes

$S \to C_bA / C_aB$

$A \to C_b\,AA / C_aS / a$

$B \to C_a\,BB / C_bS / b$

$C_b \to b$

$C_a \to a$

Now, let us replace C_bA by D and C_aB by E then grammar becomes as follows:

$S \to C_bA / C_aB$

$A \to DA / C_aS / a$

$B \to EB / C_bS / b$

$C_b \to b$

$C_a \to a$

$D \to C_bA$

$E \to CaB$

This is the required CNF from.

4. Reduce the given CFG with P given by

$S \to a / b / cSS$

Solution:

(a) There are no $\in$ productions and no unit productions in given P.

(b) Among the given productions,

$S \to a$

and $S \to b$

are in proper form

For $S \to cSS$, we have

$S \to B_c\,SS$

$B_c \to c$

Now set of productions P' given by

P' = {

$S \to B_c\,SS$

$B_c \to c$

$S \to a$

$S \to b$

}

In P' above, we have

$S \rightarrow B_c SS$

not in proper form.

Hence, we have new variables D_1 and new productions,

$S \rightarrow B_c D_1$

$D_1 \rightarrow SS$

Therefore, the grammar in Chomsky normal form (CNF) is G_2, with productions given by

$S \rightarrow B_c D_1$

$D_1 \rightarrow SS$

$B_c \rightarrow c$

$S \rightarrow a$

and $S \rightarrow b$

This is the required CNF form

5. Reduce the given CFG with P given by

$S \rightarrow ab\ Sb / a / aAb$

and $A \rightarrow bs / aAAb$

to Chomsky normal form (CNF).

Solution:

(a) There are neither ϵ-productions nor unit productions in the given set of P.

(b) Among the given productions, we have

$S \rightarrow a$

in proper form

Let us replace a by B_a and b by B_b, then

P' = {

$S \rightarrow B_a B_b S B_b / a / B_a AB_b$

$A \rightarrow B_b S / B_a AA B_b$

}

(c) In P' above, we have

$S \rightarrow B_a B_a S B_b$

$S \rightarrow B_a A B_b$

and $A \rightarrow B_a AA B_b$

not in proper form

Hence, we assume new variable D_1, D_2, D_3, D_4 and D_5 with productions given as below:

For, $S \rightarrow B_a B_a S B_b$, we have

$S \rightarrow B_a D_1$

$D_1 \rightarrow B_a D_2$

$D_2 \rightarrow S B_b$

For, $S \rightarrow B_a AB_b$, we have

$S \rightarrow B_a D_3$

$D_3 \rightarrow AB_b$

For $A \rightarrow B_a AA B_b$, we have

$A \rightarrow B_a D_4$

$D_4 \rightarrow AD_5$

$D_5 \rightarrow AB_b$

Therefore, the grammar in Chomsky normal form (CNF) is

$S \rightarrow B_a D_1 / a / B_a D_3$

$A \rightarrow B_b S / B_a D_4$

$D_1 \rightarrow B_b D_2$

$D_2 \rightarrow S B_b$

$D_3 \rightarrow A B_b$

$D_4 \rightarrow AD_5$

$D_5 \rightarrow A B_b$

$B_a \rightarrow a$

$B_b \rightarrow b$

6. Design a CFG for the language $L = \{a^{4n} / n \geq 1\}$ and convert that CFG into CNF form.

Solution:

Let CFG for the language

$L = \{a^{4n} / n \geq 1\}$ is

$G = (V_n, V_t, P, S)$

$V_n = \{S\}$

$V_t = \{a\}$

and P is defined as

$S \rightarrow aaaa\,S / aaaa$

Now, let us convert this CFG into CNF form, replace a by A then CFG becomes

$S \rightarrow AAAAS / AAAA$

$A \rightarrow a$

Then CNF form will be

$S \rightarrow AR_1$
$R_1 \rightarrow AR_2$
$R_2 \rightarrow AR_3$
$R_3 \rightarrow AS$
$S \rightarrow AR_4$
$R_4 \rightarrow AR_5$
$R_5 \rightarrow AA$
$A \rightarrow a$

Greibach normal form (GNF)

A grammar is in Greibach normal form by all productions are of the form

$A \rightarrow a\alpha$

where 'a' is terminal and α denotes no or many variables.

Grammars in GNF are much longer than the CFG from which they were derived. GNF is useful for proving the equivalence of NPDA (non-deterministic push-down automata) and CFG.

Difference between CNF (Chomsky normal form) and GNF (Greibach normal form)

CNF	GNF
(a) In CNF, all productions take the following forms. $A \rightarrow BC$ $A \rightarrow a$ where A, B and C are (non-terminals) variables and 'a' is terminal symbol.	(a) In GNF, all productions take the following form $A \rightarrow a\alpha$ where 'a' is terminal symbol and α denotes no or many variables.
(b) It is not useful for proving the equivalence of NPDA and CFG.	(b) It is useful for proving the equivalence of NPDA and CFG.
(c) It has language derivation than GNF.	(c) It has shorter derivation CNF.
(d) It has two forms.	(d) It has a single form.
(e) It is not useful in converting a CFG to NPDA.	(e) It is useful in converting a CFG to NPDA.

Examples

1. Convert the grammar

$A \rightarrow AB / BC$
$A \rightarrow aB / bA / a$
$B \rightarrow bB / cC / b$
$C \rightarrow c$

into GNF

Solution:

Here, the production $S \rightarrow AB / BC$ is not is GNF. On applying, the substitution rule we immediately get equivalent grammar.

$S \rightarrow aBB / bAB / aB / bBC / cCC / bC$
$A \rightarrow aB / bA / a$
$B \rightarrow bB / cC / b$
$C \rightarrow c$

2. **Consider the following grammar and write its equivalent GNF.**

$S \rightarrow AB$
$A \rightarrow aA / bB / b$
$B \rightarrow b$

Solution:

Here, the production $S \rightarrow AB$, is not in GNF. On applying the substitution rule, we immediately get equivalent grammar

$S \rightarrow aAB / bBB / bB$
$A \rightarrow aA / bB / b$
$B \rightarrow b$

This is the required GNF.

3. **Convert the grammar into GNF.**

$S \rightarrow abSb / aa$

Solution:

Here, we can use a method similar to the one introduced in the construction of Chomsky normal form. We introduce new variables A and B that are essentially symbols for a and b, respectively, substituting for the terminals with their associated variables leads to the equivalent grammar.

$S \rightarrow aBSB / aA$
$A \rightarrow a$
$B \rightarrow b$

This is the required GNF.

Regular grammars

A grammar is regular if it has rules of form $A \to a$ or $A \to aB$ or $A \to \epsilon$ where ϵ is a special symbol called NULL (epsilon). A regular grammar may be left linear or right linear.

If all production of a CFG are of the form $A \to wB$ or $A \to w$, where A and B are variables and $w \in V_t^*$, then we say that grammar is right linear.

If all the production of a CFG are of the form $A \to Bw$ or $A \to w$, we call it left linear.

A right or left linear is called a regular grammar.

Examples

1. **Write the left linear and right linear grammar for the regular expression r = 0 (10)***

Solution:

Given regular expression r = 0(10)*, it means any string which starts with 0 followed by any number of 10's is in regular expression.

Left linear: Let grammar be $G = (V_n, V_t, P, S)$

$V_n = \{S\}$

$V_t = \{0, 1\}$

$P = \{$

$S \to S10 / 0$

$\}$

This is left linear as we substitute S from left.

Right linear: Let grammar be G'

where, $G' = (V_n', V_t', P, S)$

where, $V_n' = \{S, A\}$

$V_t' = \{0, 1\}$

$P = \{$

$S \to 0A$

$A \to 10A / \epsilon$

This is right linear as we substitute A from right.

2. **Write the right linear and left linear regular grammar for regular expression**

r = (ab)* a

Solution:

This regular expression r generates string containing any number of ab's ending with single 'a'.

Left linear: Left grammar be G_1

$G_1 = (Vn, V_t, P, S)$

where,

$V_n = \{S, S_1\}$

$V_t = \{a, b\}$

and $P = \{$

$S \to S_1 a$

$S_1 \to S_1 ab / \epsilon$

$\}$

Right linear: Let grammar be G_2

$G_2 = (V_n', V_t', P', S)$

where,

$V_n' = \{S\}$

$V_t = \{a, b\}$

$P' = \{$

$S \to ab S / a$

$\}$

Pumping lemma for CFG

If language A is CFL, then it must have a pumping length 'p' such that any string 'S' $\in$ A having length $|S| \geq P$ can be divided into 5 parts uvxyz such that following conditions must be true

(a) $uv^ixy^iz \in A$ for every $i \geq 0$.

(b) $|vy| \geq 1$

(c) $|vxy| \leq P$

Examples

1. **Show that language $L = \{a^N b^N c^N / N \geq 0\}$ is not context free grammar using pumping lemma.**

Solution:

We will prove it by contradiction using pumping lemma. And pumping lemma states, "If a language L is context free then it must have a pumping length 'P' such that any string 'S' $\in$ L having length $|S| \geq P$ can be divided into 5 parts uvxyz such that following conditions must be true.

(a) $uv^ixy^iz \in A$ for every $i \geq 0$

(b) $|vy| \geq 1$

(c) $|vxy| \leq P$

Let, $S \in L = a^P b^P c^P$

Let, P = 3

$S = a^3 b^3 c^3$

Dividing S into u, v, x, y and z, we get

Case – I: When 'v' and 'y' contain only one type of symbol.

$\underbrace{a}_{u}\ \underbrace{a}_{v}\ \underbrace{abbb}_{x}\ \underbrace{cc}_{y}\ \underbrace{c}_{z}$

Checking for condition 1 of pumping lemma

Let, i = 2

$S = uv^2xy^2z$

$= a\ aa\ abbb\ ccccc$

$= a^4 b^3 c^5 \notin L$

Here, condition 1 fails.

Case II: Either v or y contains more than one kind of symbol i.e.

$\underbrace{aa}_{u}\ \underbrace{ab}_{v}\ \underbrace{bbc}_{x}\ \underbrace{c}_{y}\ \underbrace{c}_{z}$

Checking for condition 1 of pumping lemma

Let, i = 2

$S = uv^2xy^2z$

$= aa\ abab\ bbc\ cc\ c \notin L$

Here, it doesn't follow the pattern $a^N b^N c^N$ so, condition 1 fails. So, none of the cases satisfy all the three conditions of context free language as stated by pumping lemma. Therefore, language L is not context free language.

2. Show that language $L = \{ww / w \in \{0, 1\}^*\}$ is not context free.

Solution:

We prove it by contradiction using pumping lemma pumping lemma states that, "if a language A is CFG, then it must have a pumping length 'P' such that any string $S \in A$ having length $|S| \geq P$ can be divided into 5 parts uvxyz such that following conditions must be true.

(a) $uv^ixy^iz \in A$ for every $i \geq 0$

(b) $|vy| \geq 1$

(c) $|vxy| \leq P$

Let, $S \in L = 0^P 1^P 0^P 1^P$

Let, P = 3

Then, $S = 0^3 1^3 0^3 0^3$

$= 000111\ 000111$

Dividing S into u, v, x, y and z, we get

Case I: When vxy i.e. S in first half of the string

$S = \underbrace{0}_{u}\ \underbrace{0}_{v}\ \underbrace{011}_{x}\ \underbrace{1}_{y}\ \underbrace{000111}_{z}$

Checking for condition 1 of pumping lemma, we get

Let, i = 2

$S = uv^2xy^2z$

$= 00001111\ 000111$

$= 0^4 1^4 0^3 1^3 \notin L$

i.e. first half is not equal to second half so condition 1 fails.

Case II: When vxy lies in second half of string.

$S = \underbrace{000111}_{u}\ \underbrace{0}_{v}\ \underbrace{001}_{x}\ \underbrace{1}_{y}\ \underbrace{1}_{z}$

Checking for condition 1

Let, i = 2

$S = uv^2xy^2z$

$= 00011100001111$

$= 0^3 1^3 0^4 1^4 \notin L$

Again, condition 1 fails.

Case III: When vxy lies in both half of string i.e.

$S = \underbrace{0}_{u}\ \underbrace{00}_{v}\ \underbrace{111000}_{x}\ \underbrace{11}_{y}\ \underbrace{1}_{z}$

Checking for condition 1

Let, i = 2

$S = uv^2xy^2z$

$= 0\ 0000\ 111000\ 11111$

$= 0^5 1^3 0^3 1^5 \notin L$

Again, condition 1 fails.

Here, on pumping (i.e. uv^2xy^2z) doesn't follow pattern $0^P 1^P 0^P 1^P$ in any of the cases (i.e. first half is equal to second half). So, none of the cases follow all three conditions of context free languages as stated by pumping lemma. Therefore, language L is context free language.

Properties of context free languages

Property 1: The family of context free languages is closed under union, concatenation and kleene star-closure.

Proof:

Let L_1 and L_2 be two context free languages generated by the following context free grammars respectively.

$G_1 = (V_{n1}, V_{t1}, P_1, S_1)$

and $G_2 = (V_{n2}, V_{t2}, P_2, S_2)$

Union: Consider the language L(G), generated by the following grammar

$G = (V_n, V_t, P, S)$

where $V_n = V_{n1} \cup V_{n2} \cup \{S\}$

$V_t = V_{t1} \cup V_{t2}$

S is the start symbol and production P is defined as follows:

$P = \{P_1 \cup P_2 \cup \{S \to S_1 / S_2\}\}$

Now, let us choose a string $w \in (V_{t1} \cup V_{t2})$. If $S_1 \overset{*}{\Longrightarrow} w$ or $S_2 \overset{*}{\Longrightarrow} w$ and in our grammar $S \to S_1/S_2$ hence S will lead to w.

Hence G is a context free grammar, so that L(G) is a context free languages that is

$L(G) = L_1 \cup L_2$ is a CFL

Concatenation: Now, again consider a language L(G) generated by the CFG.

$G = (V_n, V_t, P, S)$

where $V_n = V_{n1} \cup V_{n2} \cup \{S\}$

$V_t = V_{t1} \cup V_{t2}$

S is the starting symbol of this grammar and production P is defined as follows

$P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$

Let w_1 be a string $w_1 \in L$ and w_2 is a string $w_2 \in L$, we know that $S_1 \overset{*}{\Longrightarrow} w_2$ and $S_2 \overset{*}{\Longrightarrow} w_2$ but in above grammar G, $S \Rightarrow S_1 S_2$. So, S will lead to w_1w_2 in broader way the language of grammar G will be L_1L_2, since L_1 is CFL and L_2 is CFL and G is context free grammar so L_1L_2 is also context free language.

Kleene star: Let L be context free language, generated by the following grammar.

Now, let kleene star of language L that is L^* is generated by a grammar G'.

$G' = (V'_n, V'_t, P', S')$

where $V'_n = V_n$

$V'_t = V_t$

'S' is the starting non-terminal and set of production is given by

$P' = P \cup \{S' \to S, S' \to \epsilon, S' \to SS\}$

When we carefully analyse P' then it is very clear that it follow all the properties of CFG since P' is production of given CFG and $S' \to S$, $S' \to \epsilon$, $S' \to SS$ also fulfill the requirement so we can say that G' is a context free grammar then L^* will be context free.

Property – 2: The family of context free language is not closed under intersection and complementation.

Proof:

We know that languages

$L_1 = \{a^m b^m c^n / n, m \geq 0\}$

$L_2 = \{a^n b^m c^m / n, m \geq 0\}$

are context free. Now let see $L_1 \cap L_2 = \{a^n b^n c^n / n \geq 0\}$ is not context free we have proven that language L (say) = $L_1 \cap L_2 = \{a^n b^n c^n / n \geq 0\}$ is not context free language using pumping lemma. So, it is proven that family of context free languages is not closed under intersection.

Complementation: Let L_1 and L_2 be two context free languages, their union $L_1 \cup L_2$ also is context free (by theorem). Now let assume that family of context free language is closed under complementation, that is complement of a context free language is also context free.

So, according to our assumption $\overline{L_1}$ and $\overline{L_2}$ are context free and their union $\overline{L_1} \cup \overline{L_2}$ is also CFL (by theorem). Again $\overline{\overline{L_1} \cup \overline{L_2}}$ will be CFL (by the assumption).

We know that Demorgan's law is

$$L_1 \cap L = 2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

We have right now proven that $\overline{\overline{L_1} \cup \overline{L_2}}$ is context free. So, $L_1 \cap L_2$ should also be context free but it is not true as we have already seen it. So, our assumption that complement of context free language is also context free is wrong.

So, we can say that, family of context free language is not closed under complementation

Property – 3: The intersection of a context free language and a regular language is a context free language.

Proof:

Here we give a proof based on finite and push-down automata; which would be simpler than one based on grammars. Let L be context free language and R is a regular set. So, there exists a PDA $A_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, S_1, F_1)$ and a finite $A_2 = (Q_2, \Sigma_2, \delta_2, S_2, F_2)$ for these languages respectively. Now we will combine these machines into a single PDA P that carries out computations by A_1 and A_2 in parallel and accepts a string if both would have accepted. Let the combined PDA be

$$A\ (Q, \Sigma, \Gamma, \delta, S, F)$$

where $Q = Q_1 \times Q_2$.

(Cartesian product of the states of A_1 and A_2)

$$\Sigma = \Sigma_1 \cup \Sigma_2$$

$$\Gamma = \Gamma_1$$

$F = F_1 \times F_2$ and δ is defined as follows.

$(((P_2, P_2), W, \alpha), ((q_1, q_2), \beta)) \in \delta$, if and only if, $((P_1, w, \alpha), (q_1, \beta)) \in \delta_1$ and $(P_2, w) \overset{*}{\underset{A_2}{|}} (q_2, \in)$.

It means that combined PDA 'P' goes from state (P_1, P_2) to (q_1, q_2) in the same way that A_1 passes from state P_1 and q_1 and also keeps track that A_2 changes its state on the same input. It is easy to see that indeed $w \in L(A)$ if and only if $W \in L(A_1) \cap L(A_2)$.

Pushdown automata (PDA)

A pushdown automata is formally defined by 7 tuples as shown below:

$$L(M) = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$$

where,

θ = a finite set of states

Σ = a finite set of input alphabet

Γ = a finite stack alphabet

δ = transition function

q_0 = the start state

Z_0 = the start stack symbol (optional)

F = set of final / accepting states

Unlike finite automata, δ in PDA takes three arguments $\delta(q, a, X)$ where

i.e. $\delta\ (q, a, X) \rightarrow (q', k)$

q = state of machine before input

a = input symbol in Σ

X = stack symbol on top of stack / symbol to be popped out.

The output of δ give pairs (q', k)

where,

q' is new state

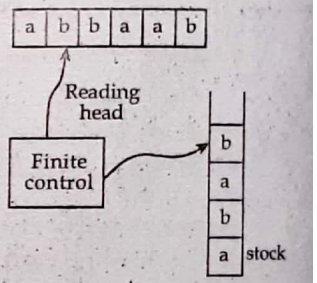
k is stack symbol that replaces

X on the top of stack.

Case I: If $k = \in$, then X is popped out.

Case II: If k = X then stack is unchanged.

Case III: If k = YX then X is push down into stack and Y is on top of stack (or X is popped out and YX is pushed down into the stack)



Pushdown automata accepts the string if

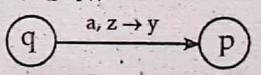
(a) It reaches final state and

(b) Stack is empty

Graphical notation of pushdown automata

Let us see some moves of PDA in the form

$((q, a, z), (p, y)) \in \delta$



Here, p and q belong to set of state 'Q' whereas z and y belong to set of stack symbol 'Γ'. And the above figure is that PDA whenever is in state q, with z on top of the stack may read 'a' from the input tape, replace z by y on top of stack and enter state p.

Note: There are two types of PDA. And they are deterministic PDA and non-deterministic PDA (NPDA). Throughout this chapter, we would mostly focus on NPDA.

Examples

1. **Design a PDA which accepts the language L = {w ∈ {a, b}* / w has the equal number of a's and b's}.**

Solution:

Let the required PDA be

$$L(M) = \{\theta, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$$

where

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

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

$\Gamma = \{a, b, c\}$

$q_0 = \{q\}$

$Z_0 = \{C\}$

$F = \{f\}$

Note that here 'C' is stack start symbol and δ is defined as follows:

δ {

1	((S, ∈, ∈), (q, c)	
2	((q, a, c), (q, ac)	
3	((q, a, a), (q, aa)	
4	((q, a, b), (q, e)	
5	((q, b, c), (q, bc)	b, a → ∈
6	((q, b, b), (q, bb)	b, b → bb
7	((q, b, a), (q, ∈)	b, c → bc
8	((q, ∈, c), (f, ∈)	a, b → ∈
	}	a, a → aa
		a, c → ac

State diagram

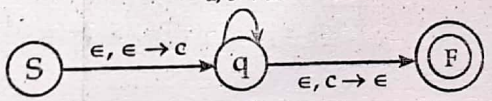


Table showing transitions and stack

Test for string 'abba'

S.N.	State	Unread input	Stack	Transition used
1.	S	abba	∈	
2.	q	abba	c	1
3.	q	bba	ac	2
4.	q	ba	c	7
5.	q	a	bc	5
6.	q	∈	c	4
7.	f	∈	∈	8

Accepted

Now, here we can see that on reading null string after reading all inputs the PDA is in final / accepting state 'f' and also the stack is empty. So, string 'abba' is accepted.

Test for string 'abbab'

S.N.	State	Unread input	Stack	Transition used
1.	S	abbab	∈	
2.	q	abbab	c	1
3.	q	bbab	ac	2
4.	q	bab	c	7
5.	q	ab	bc	5
6.	q	b	c	4
7.	q	∈	bc	8

Rejected

As we can see there is no transition for (q, ∈, b) → no transition as well as stack is not also empty. So, string abbab' is rejected.

2. Design a PDA for the following language $L = \{a^n b^n : n > 0\}$

Solution:

Let, the required PDA be

$L(M) = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$

where,

$Q = \{s, q, f\}$

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

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

$q_0 = \{q\}$

$Z_0 = \{\in\}$

$F = \{f\}$

and δ is defined as follows:

δ = {

1. ((S, a, ∈), (S, a))
2. ((S, a, a), (S, aa))
3. ((S, b, a), (q, ∈))
4. ((q, b, a), (q, ∈))
5. ((q, ∈, ∈), (f, ∈))

}

State diagram

a, a → aa

a, ∈ → a

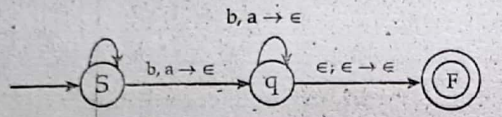


Table showing transitions and stack

Test for string 'aabb'

S.N.	State	Unread input	Stack	Transition used
1.	S	aabb	∈	
2.	S	abb	a	1
3.	S	bb	aa	2
4.	q	b	a	3
5.	q	∈	∈	4
6.	q	∈	∈	5

Here, we can see that on reading null string after reading all inputs, the PDA is in final / accepting state 'f' and also the stack is empty. So, string 'aabb' is accepted.

Test for string 'abba'

S.N.	State	Unread input	Stack	Transition used
1.	S	abba	∈	
2.	S	bba	a	1
3.	q	ba	∈	4

Rejected

We can see that there is no transition defined in δ when PDA is in state 'q' with input 'b' and empty stack. So, string 'abba' is rejected.

Equivalence of PDA and CFG

Let G = (V_n, V_t, P, S) be a context free grammar, we must construct a pushdown automata P such that L(P) = L(G). The machine we construct has only two states P and q, and remains permanently in state q after its first move. Also, P uses V_n, the set of non-terminals and V_t the set of terminals as its stack alphabet.

we let P = (θ, Σ Γ, δ, q_0, Z_0, F)

where,

θ = {p, q}

Σ = V_t

Γ = $V_n \cup V_t$

that is set of terminals and non-terminals.

q_0 = p

and transition δ is defined as follows:

1. ((p, ∈, ∈), (q, S) as S is starting non-terminal CFG)
2. ((q, ∈, A), (q, x)) for each rule A → x in CFG.
3. ((q, a, a), (q, ∈)) for each q ∈ V_t.

The pushdown automata P begins by pushing S, the starts symbol of grammas G, on its initially empty pushdown store, and entering state of (transition 1). On each subsequent step, if either replaces the topmost symbol A on the stack, provided that if is a non-terminal, by the right hand side x of some rule A → x in grammar (transition of type 2) or pops the top most symbol from the stack, provided that if it is a terminal symbol that matches the next input symbol (transition of type 3).

Examples

1. **Design a PDA for the following CFG;**

G = V_n, V_t, P, S with

V_n = {S}, V_t = {(,)} and P is defined δ as follows:

S → ∈

S → SS

S → (S)

Solution:

Let us assume corresponding PDA will be

1 = (θ, Σ, Γ, δ, q_0 Z_0, F)

where,

θ = {p, q}

Σ = {(,)}

Γ = {S, (,)}

q_0 = {P}

F = {q}

Z_0 = stack stat symbol → ∈ in this case and transition δ is defined as follows:

1. ((p, ∈, ∈), (q, S))
2. ((q, ∈, S), (q, ∈))
3. ((q, ∈, S), (q, SS))
4. ((q, ∈, S), (q, (S)))
5. ((q, c, c), (q, ∈))
6. ((q,),), (q, ∈))

Let us apply these relations on the string

w = ( ) ( )

S.N.	State	Unread input	Stack	Transition used
1.	P	( ) ( )	∈	
2.	q	( ) ( )	S	1
3.	q	( ) ( )	SS	3
4.	q	( ) ( )	(S) S	4
5.	q	) ( )	S) S	5
6.	q	) ( )	) S	2
7.	q	( )	S	6
8.	q	( )	(S)	4
9.	q	)	S)	5
10.	q	)	)	2
11.	q	∈	∈	6

2. Design a PDA for the grammar

G = (V_n, V_t, P, S)

V_n = {S}

V_t = {a, b, c}

and P is defined as

S → a Sa

S → bsb

S → c

Solution:

Let P D A be

L = (θ, Σ, Γ, δ, q0, Z0, F)

where,

θ = {p, q}

Σ = {a, b, c}

Γ = {S, a, b, c}

q_0 = {p}

Z_0 = ∈, the stack start symbol

F = {q}

and δ is defined as

1. ((p, ∈, ∈), (q, S)
2. ((q, ∈, S), (q, aSa)
3. ((q, ∈, S), (q, bSb)
4. ((q, ∈, S), (q, c)
5. ((q, a, a), (q, ∈)
6. ((q, b, b), (q, ∈)
7. ((q, c, c), (q, ∈)

Let us apply this on string

w = abbcbba

S.N.	State	Unread input	Stack	Transition used
1.	P	abbcbba	∈	
2.	q	abbcbba	S	1
3.	q	abbcbba	aSa	2
4.	q	bbcbba	Sa	5
5.	q	bbcbba	bSba	3
6.	q	bcbba	Sba	6
7.	q	bcbba	bSbba	3
8.	q	cbba	Sbba	6
9.	q	cbba	cbba	4
10.	q	bba	bba	7
11.	q	ba	ba	6
12.	q	a	a	6
13.	q	∈	∈	5

Exam Solution

1. **Convert the following CFG into CNF with explanation of each steps G = (V, Σ R, S), where**

V = {S, X, Y, a, b, c},

Σ = {a, b, c}

R = {S → a X b X

X → a Y / b Y / X Y / ∈

Y → a X / c

} **[2075 Ashwin, Back]**

The given set of production / rules

R = {S → aX bX

X → aY / bY / XY / ∈

Y → aX / a / c

Here, we have a null production X → ∈, Removing null production X ∈, we replace symbol X be ∈ and add the new production to rules / grammar i.e.

S → aX bX / abX / aXb / ab

X → aY / bY / XY / Y

Y → aX / a / c

Now, we have unit production X → Y, we add production of Y to X to remove it.

S → aX bX / abX / a Xb / ab

X → aY / bY / XY / aX / a / c

Y → aX / a / c

Now, we don't have neither null production nor unit production. Also, we don't have useless symbols.

Therefore, new set of productions

R' = {

S → aX bX / abX / aXb / ab

X → aY / bY / XY / aX / a /C

Y → aX / a / C

Now, this set is ready to be converted into Chomsky normal form

Let, $C_a \rightarrow a$

$C_b \rightarrow b$

Then S → C_aX C_bX / C_a C_bX / C_aX C_b / $C_a C_b$

X → C_aY / C_bY / XY / C_aX / a / C

Y → C_aX / a / C

Since NF is of form A → BC, A → a (i.e. expressions on right hand side of production must be two variable (non-terminals) or a single terminal, we will introduce new-variables (non-terminals) to get CNF.

For S → C_aX C_bX, replace D_1 by CAX C_b and so on

S → D_1X

D_1 → D_2 C_b

D_2 → C_aX

For S → C_a XC_b, we have

S → D_2 C_b

D_2 → C_aX

For S → C_a C_bX, we replace D_3 by C_a C_b

S → D_3 X

D_3 → C_a C_b

Now, all of the productions are of the form CNF

R' = {

S → D_1X / D_3X / D_2 C_b / C_a C_b

X → C_aY / C_bY / XY / C_aX / a / C

Y → C_aX / a / C

D_1 → D_2 C_b

D_2 → C_aX

D_3 → C_a C_b

Ca → a

Cb → b

}

This is the required CNF form

2. **Convert the following CGF into CNF with explanation of each steps G = {V, Σ, R, S) where**

V = {S, A, B, a, b}

Σ = {a, b}

R = {

S → ASB / ∈

A → aAS / a

B → AB / b / ∈

} **[2074 Ashwin, Back]**

Given sets of productions

R = {

S → ASB / ∈

A → aAS / a

B → AB / b / ∈

}

Step-I: Elimination of useless.

Here, no productions are useless. So, no need to remove useless symbols.

Step-II: Elimination of null (∈) productions.

Here, two null productions are present

S → ∈

B → ∈

First, remove S → ∈ on doing so, we replace S by ∈ and all the new productions to the grammar (i.e.)

S → ASB / AB

A → aAS / aA / a

B → AB / b / ∈

Now, we remove B → ∈. On doing so, we replace B by ∈ and add all the new productions to the grammar (i.e.)

S → ASB / AB / AS / A

A → aAS / aA / a

B → AB / b / A

Now, all null productions are removed

Step-III: Elimination of unit productions.

Here, unit productions are

S → A

B → A

First, remove S → A. On doing so, we add all production of A to the production of S

S → ASB / AB / AS / aAS/ aA / a

A → aAS / aA / a

B → AB / A / b

Secondly, remove B → A, on doing so, we add all productions of A to the production of B.

S → ASB / AB / aAS / AS / aA / a

A → aAS / aA / a

B → AB / aAS / aA / a / b

Now, we have new set of production R'

R' = {

S → ASB / AB / aAS / AS / aA / a

A → aAS / aA / a

B → AB / aAS / aA / a / b

}

For CNF, we make all productions either as combination of two non-terminals or a single terminal.

Let C_a → a then

S → ASB / AB / C_aAS / AS / C_aA / a

A → C_aAS / C_aA / a

B → AB / C_aAS / C_aA / a / b

For S → ASB, we have

S → D_1B

D_1 → AS

For S → C_aAS, we have

S → D_2S

D_2 → C_aA

For A → C_aAS, we have

A → D_2S

D_2 → C_aA

For B → C_aAS, we have

B → D_2S

D_2 → C_aA

Finally new set of production P' is

P' = {

S → D_1B / AB / D_2S / AS / C_aA / a

A → D_2S / C_aA / a

$B \rightarrow AB / D_2S / C_aA / a / b$

$Ca \rightarrow a$

$D_1 \rightarrow AS$

$D_2 \rightarrow C_aA$

}

This is the required CNF

3. Define Chomsky normal form (CNF). Convert the following CFG into CNF G = (V, Σ R, S)

where

$V = \{S, A, B, a, b\}$

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

$R = \{$

$S \rightarrow A$

$S \rightarrow B$

$A \rightarrow a\ Ba$

$A \rightarrow e$

$B \rightarrow b\ Ab$

$B \rightarrow e$

} **where e is empty symbol.** **[2073 Chaitra]**

For definition of Chomsky normal form (CNF), please see theory part.

Here, the given set of productions.

$R = \{$

$S \rightarrow A / B$

$A \rightarrow aBa / e$

$B \rightarrow b\ Ab / e$

}

Step-I: Elimination of useless symbols none of the productions / symbols are useless.

Step-II: Elimination of null (∈) production null productions are

$A \rightarrow e$

$B \rightarrow e$

First, we remove $A \rightarrow e$ on doing so, we replace A by e and add new productions to the grammar

$S \rightarrow A / B / e$

$A \rightarrow a\ Ba$

$B \rightarrow b\ Ab / bb / e$

Secondly, we remove $B \rightarrow e$, on doing so, we replace B by e and add new productions to the grammar

$S \rightarrow A / B / e$

$A \rightarrow a\ Ba / aa$

$B \rightarrow b\ Ab / bb$

Note: no need to write two 'e' in production of A.

Now, we have new null production $S \rightarrow e$, removing $S \rightarrow e$, we replace S by e and add new productions to the grammar

$S \rightarrow A / B$

$A \rightarrow a\ Ba / aa$

$B \rightarrow b\ Ab / bb$

Step-III: Elimination of unit productions.

Here, unit productions are

$S \rightarrow A$

$S \rightarrow B$

To remove unit production $S \rightarrow A$, we add every production of A to the reproduction of S. i.e.

$S \rightarrow aBa / aa / B$

$A \rightarrow aBa / aa$

$B \rightarrow bAb / bb$

To remove unit production $S \rightarrow B$, we add every production of B to the production of S.

$S \rightarrow aBa / aa / bAb / bb$

$A \rightarrow aBa / aa$

$B \rightarrow bAb / bb$

In order to convert CGE into CNF, we make each production either as combination of two non-terminals or a single terminal.

We have new set of production R' as

$R' = \{$

$S \rightarrow a\ BA / aa / bAb / bb$

$A \rightarrow a\ Ba / aa$

$B \rightarrow b\ Ab / bb$

}

Let,

$C_a \rightarrow a$

$C_b \rightarrow b$

$S \rightarrow C_a BC_a / C_a C_a / C_b A C_b / C_b C_b$

$A \rightarrow C_a BC_a / C_a C_a$

$B \rightarrow C_b A C_b / C_b C_b$

For $S \rightarrow C_a B C_a$, we have

$S \rightarrow D_1 C_a$

$D_1 \rightarrow C_a B$

For $S \rightarrow C_b A C_b$, we have

$S \rightarrow D_2 C_b$

$D_2 \rightarrow C_b A$

For $A \rightarrow C_a B C_a$, we have

$A \rightarrow D_1 C_a$

$D_1 \rightarrow C_a B$

For $B \rightarrow C_b A C_b$, we have

$B \rightarrow D_2 C_b$

$D_2 \rightarrow C_b A$

Finally, we have new set of production R' as

R' = {

$S \rightarrow D_1 C_a / C_a C_a / D_2 C_b / C_b C_b$

$A \rightarrow D_1 C_a / C_a C_a$

$B \rightarrow D_2 C_b / C_b C_b$

$C_a \rightarrow a$

$C_b \rightarrow b$

$D_1 \rightarrow C_a B$

$D_2 \rightarrow C_b A$

}

This is the required CNF form.

4. What is Chomsky normal form (CNF)?

{S, L, M, N, a, b, C}, Σ = {a, b, c}

R = {

$S \rightarrow Ma\,N / bL / bM$

$L \rightarrow ab / CN / M1 / \epsilon$

$M \rightarrow a / cM$

$N \rightarrow abN$

}

[2073 Shrawan, Back]

Ans For definition, please see theory part

Given, we have

R = {

$S \rightarrow MaN / bL / bM$

$L \rightarrow ab / cN / M1 / \epsilon$

$M \rightarrow a / cM$

$N \rightarrow abN$

}

Step – I: Elimination of useless symbol.

Here, N is useless i.e. non-generating symbol as it doesn't generate any kind of terminal symbol or symbols. So, removing all productions containing 'N', we get

$S \rightarrow bL / bM$

$L \rightarrow ab / M1 / \epsilon$

$M \rightarrow a / CM$

Step – II: Elimination of null production.

Here, null production is

$L \rightarrow \epsilon$

So, we replace L by ϵ and add these new productions to the grammars.

$S \rightarrow bL / bM / b$

$L \rightarrow ab / M1$

$M \rightarrow a / CM$

Step – III: Elimination of unit production.

Here, no unit productions are present we have new set of productions

R' = {

$S \rightarrow bL / bM / b$

$L \rightarrow ab / M1$

$M \rightarrow a / CM$

}

In order to convert CFG to CNF form, we make each production either as combination of two non-terminal symbol or a single terminal symbol.

Let $C_a \rightarrow a$
$C_b \rightarrow b$
$C_1 \rightarrow 1$
$C_c \rightarrow C$

Then, $R' = \{$
$S \rightarrow C_bL / C_bM / b$
$L \rightarrow C_aC_b / M C_1$
$M \rightarrow a / C_cM$
$C_a \rightarrow a$
$C_b \rightarrow b$
$Cc \rightarrow C$
$\}$ $C_1 \rightarrow 1$

This is the required CNF form.

5. Convert the following CFG into CNF

G = {V, T, P, S}
where V = {S, A, B, C, a, b, c}
T = {a, b, c}
P = {
S → ABA / abA / BC
A → aA / ∈
B → baB / c
C → aC
} **[2074 Chaitra]**

We have,
P = {
S → ABA / abA / BC
A → aA / ∈
B → ba B / c
C → aC
}

Step – I: Elimination of useless symbol.

Here, symbol 'c' is useless as it doesn't generate any kind of terminal symbols. So, we need to remove all productions containing C.

C → AB A / ab A
A → aA / ∈
B → ba B / c

Step-II: Elimination of null production.

Here, A → ∈ is null production, so we replace A by ∈ and add the new productions to the grammar i.e.

S → ABA / BA / AB / B / abA / ab
A → aA
B → baB / c

Step-III: Elimination of unit production we have S → B (unit production), so we add production of B to the production of S i.e.

S → ABA / BA / AB / baB / c / abA / ab
A → aA
B → baB / c

Now, we have new set of productions

P' = {
S → ABA / BA / AB / baB / c / ab A / ab
A → aA
B = baB / c
}

Let, $C_a \rightarrow a$
$C_b \rightarrow b$

Then, P' = {
S → ABA / BA / AB / C_b C_aB / C / C_a C_bA / C_a C_b
A → C_aA
B → C_bC_aB / c

For CNF, we make each productions of the grammar either as combination of two non-terminals or a single terminal i.e.

For S → ABA, we have
S → D_1 A
D_1 → AB

For S → C_b C_aB, we have
S → D_2 B
D_2 → C_b C_a

For S $\rightarrow C_a C_b A$, we have

S $\rightarrow D_3 A$

$D_3 \rightarrow C_a C_b$

For B $\rightarrow C_b C_a B$, we have

B $\rightarrow D_2 B$

$D_2 \rightarrow C_b C_a$

Now, set of production become

P' = {

S $\rightarrow D_1 A / BA / AB / D_2B / c / D_3A / C_aC_b$

B $\rightarrow C_aA$

B $\rightarrow D_2B / c$

$C_a \rightarrow a$

$C_b \rightarrow b$

$D_1 \rightarrow AB$

$D_2 \rightarrow C_b C_a$

$D_3 \rightarrow C_a C_b$

}

This is the required CNF form.

6. Define context free grammar. Convert the given context free grammar (CFG) into equivalent CNF.

S $\rightarrow$ AB

A $\rightarrow$ a AA / e

B $\rightarrow$ bBB / e **[2072 Chaitra]**

Formal definition of context free grammar (CFG), please see theory part.

Now, we have

S $\rightarrow$ AB

A $\rightarrow$ aAA / e

B $\rightarrow$ bBB / e

Step – I: Elimination of useless symbols

Here, no symbols are useless as all productions can be reached from start symbols 's'.

Step – II: Elimination of null 'e' production.

Here, null productions are

$A \rightarrow e$

$B \rightarrow e$

First removing $A \rightarrow e$, on doing so, we replace symbol A by 'e' and adding all the productions to the grammar, we get

S $\rightarrow$ AB / B

A $\rightarrow$ a AA / aA / a

B $\rightarrow$ bBB / e

Secondly removing $B \rightarrow e$, we replace symbol 'B' by 'e' and adding all the productions to the grammar

S $\rightarrow$ AB / B / A

A $\rightarrow$ aAA / aA / a

B $\rightarrow$ bBB / bB / b

Step – III: Elimination of unit production.

Here, unit productions are

$S \rightarrow A$

$S \rightarrow B$

For removing $S \rightarrow A$ unit production, we add all production of A to the production of S.

S $\rightarrow$ AB / B / aAA / aA / a

A $\rightarrow$ aAA / aA / a

B $\rightarrow$ bBB / bB / b

For removing $S \rightarrow B$, unit production, we add all production of B to the production of S

S $\rightarrow$ AB / bBB / bB / b / aAA / aA / a

A $\rightarrow$ aAA / aA / a

B $\rightarrow$ bBB / bB / a

Now, we have new set of productions or grammar

S $\rightarrow$ AB / bBB / b / aAA / aA / a / bB

A $\rightarrow$ aAA / aA / a

B $\rightarrow$ aBB / aB / b

In order to convert CFG into CNF, we write all productions on right hand side either as combination of two non-terminal symbol or a single terminal

Let, $C_a \rightarrow a$

$C_b \rightarrow b$

Then, S $\rightarrow$ AA / Ca AA / C_aA / a / C_b BB / C_bB / b

A $\rightarrow C_a$ AA / C_aA / a

B $\rightarrow C_b$ BB / C_bB / b

For S → C_a AA, we have

S → D_1A

D_1 → C_a A

For S → C_b BB, we have

S → D_2B

D_2 → C_b B

For A → C_a AA, we have

A → D_1A

D_1 → C_a A

For B → C_b BB, we have

B → D_2 B

D_2 → C_bB

New set of productions become

S → AB / D_1A / a / D_2B / C_bB / b

A → D_1A / C_a A / a

B → D_2B / C_b B / b

C_a → a

C_b → b

D_1 → C_a A

D_2 → C_b B

This is the required CNF from

7. Convert the following CFG to CNF

G = (V, Σ, R, S) where

V = {S, A, B, a, b}

Σ = {a, b}

R = {

S → aAb / Ba / A

A → SS / e

B → e

} **[2071 Chaitra]**

The given set of production / rules

R = {

S → aAb / Ba / A

A → SS / e

B → e

}

Step – I: Elimination of useless symbols.

Here, no symbols are useless as all symbols can be reached from start symbol 'S'

Step – II: Elimination of null (e) production

Here, null productions are

A → e

B → e

First, removing A → e, we replace A by e and add all the new productions to grammar i.e.

S → aAb / ab / Ba / e

A → SS

B → e

Secondly, removing B → e, we replace B by e and add the new productions to grammar i.e.

S → aAb / ab / Ba / a / e

A → SS

Now, we have new null production S → e, so removing S → e, we get

S → aAb / ab / Ba / a

A → SS / S

Note: no need to write A → e as we have already removed A → e.

Step – III: Elimination of unit productions.

Here, unit production is

A → S

So, removing A → S unit production, we add all production of S to the productions of A

S → aAb / ab / Ba / a

A → SS / aAb / ab / Ba / a

Therefore, new set of productions

R' = {

S → aAb / ab / Ba / a

A → SS / aAb / ab / Ba / a

}

For CNF, we make all productions either as combination of two non-terminals or a single terminal

Let, $C_a \rightarrow a$

$C_b \rightarrow b$, then

$S \rightarrow C_a AC_b / C_a C_b / BC_a / a$

$A \rightarrow SS / C_a AC_b / C_aC_b / BC_a / a$

For $S \rightarrow C_a A C_b$, we have

$S \rightarrow D_1 C_b$

$D_1 \rightarrow C_a A$

For $A \rightarrow C_a A C_b$, we have,

$A \rightarrow D_1 C_b$

$D_1 \rightarrow C_a A$

Finally, new set of productions R' is

R' = {

$S \rightarrow D_1 C_b / C_a C_b / BC_a / a$

$A \rightarrow SS / D_1 C_b / C_a C_b / BC_a / a$

$C_a \rightarrow a$

$C_b \rightarrow b$

$D_1 \rightarrow C_a A$

}

This is the required CNF form.

8. Convert the following CFG to Chomsky normal form (CNF)

G = {V, Σ, R, S) where V = {S, A, B, a, b}

Σ = {a, b}

R = {

$S \rightarrow aB\ bB$

$A \rightarrow aA,$

$A \rightarrow a$

$B \rightarrow bB$

} **[2072 Kartik, Back]**

Given, set of productional rules is

R = {

$S \rightarrow aA\ bB$

$A \rightarrow aA / a$

$B \rightarrow aB$

}

Step – I: Elimination of useless symbols.

Here, symbol B is useless symbol as it doesn't generate any kind of terminal symbol. So, we need to remove all production containing B. i.e.

$R = \{A \rightarrow aA / a\}$

Step – II: Elimination of null production.

No null productions

Step – III: Elimination of unit production.

No unit productions.

Now, we have new set of productions

$R' = \{A \rightarrow aA / a\}$

For CNF, we make each production of the grammar either as combination of two non-terminal or a single terminal i.e.

Let, $C_a \rightarrow a$

Then,

R' = {$S \rightarrow C_a A / a$

$C_a \rightarrow a$

}

This is the required CNF form.

9. Define a context free grammar convert the following productions into Chomsky normal form.

$S \rightarrow ab\ AB$

$A \rightarrow b\ AB / \epsilon$

$B \rightarrow BA\ a / A / \epsilon$ **[2073 Shrawan, Back]**

For definition of context free grammar, please see theory part.

Given, the set of productions

R = { $S \rightarrow ab\ AB$

$A \rightarrow b\ AB / \epsilon$

$B \rightarrow BA\ a / A / \epsilon$

}

Step I: Elimination of useless symbols.

Here, no symbols are useless

Step II: Elimination of null (ϵ) productions

Here, null productions are

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

First, we remove $A \to \epsilon$, on doing so, we replace symbol A by ϵ and add all the new productions to the grammar.

$S \to abAB / abB$

$A \to b\,AB / bB$

$B \to BA\,a / Ba / \epsilon$

Secondly, we remove $B \to \epsilon$, on doing so, we replace symbol B by ϵ and add all the new productions to the grammar

$S \to abAB / abA / abB$

$A \to bAB / bA / bB$

$B \to BA\,a / Ba / Aa$

Step III: Elimination of unit production

Here, no unit productions are present

Therefore, we have new set of productions

$R' = \{$

$S \to abAB / abA / abB$

$A \to bAB / bA / bB$

$B \to BAa / Ba / Aa$

$\}$

For CNF, we write all productions either as combination of two non-terminals or a single terminal.

Let, $C_a \to a$

$C_b \to b$

Then,

$S \to C_a C_b AB / C_a C_b A / C_a C_b B$

$A \to C_b AB / C_b A / C_b B$

$B \to BA\,C_a / B\,C_a / A\,C_a$

For $S \to C_a C_b AB$, we have

$S \to D_1 B$

$D_1 \to D_2 A$

$D_2 \to C_a C_b$

For $S \to C_a C_b A$, we have

$D \to D_2 A$

$D_2 \to C_a C_b$

For $S \to C_a C_b B$, we have

$S \to D_2 B$

$D_2 \to C_a C_b$

For $A \to C_b AB$, we have

$A \to D_3 B$

$D_3 \to C_b A$

For $A \to BA\,C_a$, we have

$A \to D_4 C_a$

$D_4 \to BA$

Finally new set of production

$R' = \{$

$S \to D_1 B / D_2 A / D_2 B$

$A \to D_3 B / C_b A / C_b B$

$B \to D_4 C_a / B\,C_a / A\,C_a$

$C_a \to a$

$C_b \to b$

$D_1 \to D_2 A$

$D_2 \to C_a C_b$

$D_3 \to C_b A$

$D_4 \to BA$

$\}$

This is the required CNF form.

10. Convert the following CFG into CNF with explanation of each step G = (V, Σ, R, S) where

V = {S, X, Y, Z, a, b, c}

Σ = {a, b, c}

R = {

S → XYZ / XY / aZ

X → abX / ∈

Y → bY / cZ / ab

Z → aXZ

} **[2071 Shrawn, Back]**

We have given set of productions

R = {

S → XYZ / XY / aZ

X → abX / ∈

Y → bY / cZ / ab

Z → aXZ

}

Step I: Elimination of useless symbols.

Here, Z is useless symbol as it doesn't generate any kind of terminal symbol. So, we need to remove all these productions containing Z i.e.

$S \rightarrow XY$

$X \rightarrow ab\,X / \epsilon$

$Y \rightarrow aY / ab$

Step II: Elimination of null (ϵ) productions.

Here, null production is

$X \rightarrow \epsilon$

On removing $X \rightarrow \epsilon$, we replace symbol X by ϵ and add all the new productions to the grammar.

$S \rightarrow XY / Y$

$X \rightarrow abX / ab$

$Y \rightarrow bY / ab$

Step III: Elimination of unit production

Here, unit production is

$S \rightarrow Y$

On removing $S \rightarrow Y$, we add all productions of Y to the production of S i.e.

$S \rightarrow XY / bY / ab$

$X \rightarrow abX / ab$

$Y \rightarrow bY / ab$

Therefore, we have new set of productions

$R' = \{$

$S \rightarrow XY / bY / ab$

$X \rightarrow abX / ab$

$Y \rightarrow bY / ab$

$\}$

For CNF, we write all productions either as combination of two terminals or a single terminal.

Let, $C_a \rightarrow a$

$C_b \rightarrow b$

Then, $S \rightarrow XY / C_b Y / C_a C_b$

$X \rightarrow C_a C_b X / C_a C_b$

$Y \rightarrow C_b Y / C_a C_b$

For $X \rightarrow C_a C_b X$, we have

$X \rightarrow D_1 X$

$D_1 \rightarrow C_a C_b$

Finally, we have new set of productions

$R' = \{$

$S \rightarrow XY / C_b Y / C_a C_b$

$X \rightarrow D_1 X / C_a C_b$

$Y \rightarrow C_b Y / C_a C_b$

$C_a \rightarrow a$

$C_b \rightarrow b$

$D_1 \rightarrow C_a C_b$

$\}$

This is the required CNF form.

11. What is additional feature PDA has when compared with finite automata? Explain. Design a pushdown automata (PDA) which accepts all the strings of language

$L = \{a^n b^m c^{2n} : n, m > 0\}$ **[2075, Ashwin, Back]**

The additional feature PDA has got when compared with finite automata is a storage device for inputs called stack. A stack is a "first in last out" or "last in first out" (LIFO) that is, symbols can be entered or removed only at the top of the list. When a new symbol is entered at the top, the symbol previously at the top becomes second and so on. Due to the stack, PDA can store the input it has read and process i.e. POP out and keep the input symbols.

Let, the required PDA be

$L(M) = (\theta, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where,

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{a, b, c, Z_o\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_5\}$

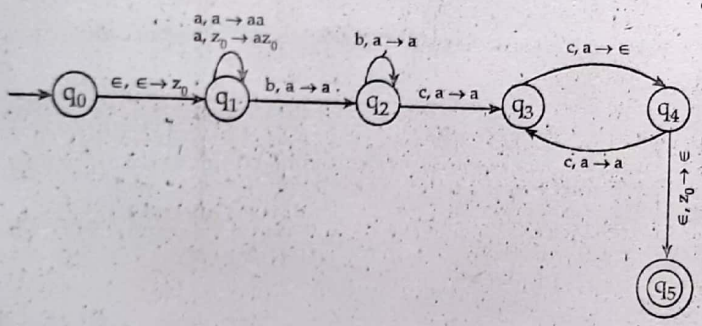
and δ is defined as follows:

$\delta = \{$

1. $((q_0, \in, \in), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_1, aZ_0))$
3. $((q_1, a, a), \quad (q_1, aa))$
4. $((q_1, b, a), \quad (q_2, a))$
5. $((q_2, b, a), \quad (q_2, a))$
6. $((q_2, c, a), \quad (q_3, a))$
7. $((q_3, c, a), \quad (q_4, \in))$
8. $((q_4, c, a), \quad (q_3, a))$
9. $((q_4, \in, Z_0), \quad (q_5, \in))$

$\}$

States diagram



Test for input string 'abbcc'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	abbcc	$\in$	–
2.	q_1	abbcc	Z_0	1
3.	q_1	bbcc	a Z_0	2
4.	q_2	bcc	a Z_0	4
5.	q_2	cc	a Z_0	4
6.	q_3	c	a Z_0	6
7.	q_4	$\in$	Z_0	7
8.	q_5	$\in$	$\in$	9

Accepted

Since, stack is empty and the final state is accepting state, so string 'abbcc' is accepted.

12. Mention role of parse tree in context free grammar Design a PDA that accepts $L = \{a^n b^{2n+1}, n > 0\}$ and check it for string aabbbbb.

[2074 Ashwin, Back]

Ans The role of parse tree in context free grammar are as follows:

(a) From parse tree, it helps us to generate intermediate strings at any part of derivation.

(b) It detects and reports any syntax errors in deviation part.

(c) We can analyse the useless symbols in derivation of a particular string.

(d) From parse tree, we can determine whether the derivation of string is either leftmost or rightmost.

(e) It helps to remove ambiguity in context free grammar.

Let, the required PDA be

$$L(M) = (\theta, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where

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

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_0\}$

$F = \{q_4\}$

$Z_0 = \{Z_0\}$

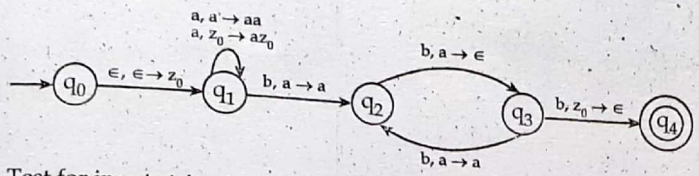
and δ is defined as follows:

$\delta = \{$

1. $((q_0, \in, \in), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_1, aZ_0))$
3. $((q_1, a, a), \quad (q_1, aa))$
4. $((q_1, b, a), \quad (q_2, a))$
5. $((q_2, b, a), \quad (q_3, \in))$
6. $((q_3, b, a), \quad (q_2, a))$
7. $((q_3, b, Z_0), \quad (q_4, \in))$

$\}$

States diagram



Test for input string 'aabbbbb'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	aabbbbb	ϵ	–
2.	q_1	aabbbbb	Z_0	1
3.	q_1	abbbbb	a Z_0	2
4.	q_1	bbbbb	aa Z_0	3
5.	q_2	bbbb	aa Z_0	4
6.	q_3	bbb	a Z_0	5
7.	q_2	bb	a Z_0	6
8.	q_3	b	Z_0	5
9.	q_4	ϵ	ϵ	7

Accepted

Since, stack is empty and the final state is accepting state i.e. q_4, string 'aabbbbb' is accpeted.

13. Design a pushdown automata (PDA) for $L = \{a^n b^{2n} : n \geq 1\}$. Hence test for "aaabbb" and "aabbbb". [2074 Chaitra]

Let, the required PDA be

$$L(M) = (\theta, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where

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

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

$\Gamma = \{a, b, Z_o\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_4\}$

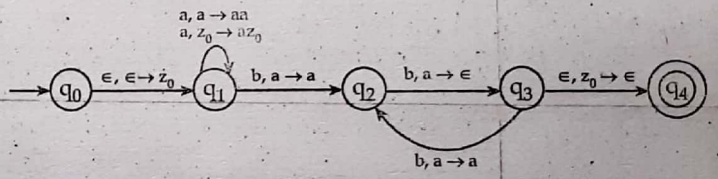
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_0, \epsilon, \epsilon), \quad (q_1, Z_o))$
2. $((q_1, a, Z_0), \quad (q_1, aZ_o))$
3. $((q_1, a, a), \quad (q_1, aa))$
4. $((q_1, b, a), \quad (q_2, a))$
5. $((q_2, b, a), \quad (q_3, \epsilon))$
6. $((q_3, b, a), \quad (q_2, a))$
7. $((q_3, \epsilon, Z_0), \quad (q_4, \epsilon))$

$\}$

States diagram



Test for input string 'aaabbb'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	aaabbb	ϵ	–
2.	q_1	aaabbb	Z_0	1
3.	q_1	aabbb	a Z_0	2
4.	q_1	abbb	aa Z_0	3
5.	q_1	bbb	aaa Z_0	3
6.	q_2	bb	aaa Z_0	4
7.	q_3	b	aa Z_0	5
8.	q_2	ϵ	aa Z_0	6

Accepted

Since, there is no transition defined for state q_2 on input 'ϵ' i.e. $\delta(q_2, \epsilon, a) \rightarrow \phi$, so string 'aaabbb' is rejected.

Test for input string 'aabbbb'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	aabbbb	$\in$	–
2.	q_1	aabbbb	Z_0	1
3.	q_1	abbbb	a Z_0	2
4.	q_1	bbbb	aa Z_0	3
5.	q_2	bbb	aa Z_0	4
6.	q_3	bb	a Z_0	5
7.	q_2	b	a Z_0	6
8.	q_3	$\in$	Z_0	5
9.	q_4	$\in$	$\in$	7

Accepted

Since, stack is empty and final state is also accepting state. So, string 'aabbbb' is accepted.

14. **Design a PDA that accepts those strings "having total number of 'a' equal to the sum of number of 'b' and 'c' with sequence of a, b, c (i.e. $a^i\ b^j\ c^k : i = j + k$). Hence, test your design for the string "aaaabbcc".** **[2073 Chaitra]**

Let, the required PDA be

$L(M) = (\theta, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where

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

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_4\}$

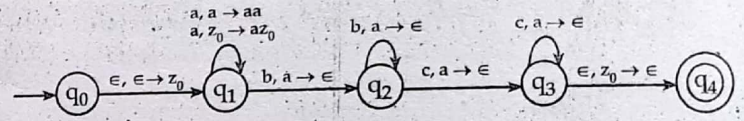
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_0, \in, \in), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_1, aZ_0))$
3. $((q_1, a, a), \quad (q_1, aa))$
4. $((q_1, b, a), \quad (q_2, \in))$
5. $((q_2, b, a), \quad (q_2, \in))$
6. $((q_2, c, a), \quad (q_3, \in))$
7. $((q_3, c, a), \quad (q_3, \in))$
8. $((q_3, \in, Z_0), \quad (q_4, \in))$

$\}$

States diagram



Test for input string 'aaaabbcc'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	aaaabbcc	$\in$	–
2.	q_1	aaaabbcc	Z_0	1
3.	q_1	aaabbcc	a Z_0	2
4.	q_1	aabbcc	aa Z_0	3
5.	q_1	abbcc	aaa Z_0	3
6.	q_1	bbcc	aaaa Z_0	3
7.	q_2	bcc	aaa Z_0	4
8.	q_2	cc	aa Z_0	5
9.	q_3	c	a Z_0	6
10.	q_3	$\in$	Z_0	7
11.	q_4	$\in$	$\in$	8

Accepted

Since, stack is empty and final state is also accepting state so string 'aaaabbcc' is accepted

15. **Construct a PDA which accepts the language $L = \{a^n b^{n+m} c^m : n, m \geq 1\}$. Verify your design by taking s string "abbcc" as example.** [2073 Shrawan, Back]

Let, the required PDA be

$L(M) = (\theta, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{a, b, c, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_5\}$

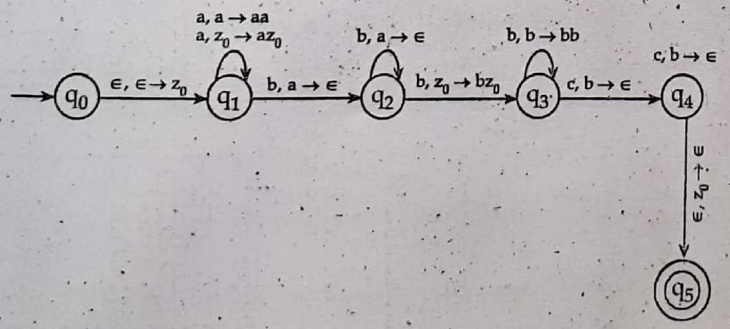
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_0, \epsilon, \epsilon), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_1, aZ_0))$
3. $((q_1, a, a), \quad (q_1, aa))$
4. $((q_1, b, a), \quad (q_2, \epsilon))$
5. $((q_2, b, a), \quad (q_2, \epsilon))$
6. $((q_2, b, Z_0), \quad (q_3, bZ_0))$
7. $((q_3, b, b), \quad (q_3, bb))$
8. $((q_3, c, b), \quad (q_4, \epsilon))$
9. $((q_4, c, b), \quad (q_4, \epsilon))$
10. $((q_4, \epsilon, Z_0), \quad (q_5, \epsilon))$

$\}$

States diagram



Test for input string 'abbbcc'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	abbbcc	ϵ	–
2.	q_1	abbbcc	Z_0	1
3.	q_1	bbbcc	a Z_0	2
4.	q_2	bbcc	Z_0	4
5.	q_3	bcc	b Z_0	6
6.	q_3	cc	bb Z_0	7
7.	q_4	c	b Z_0	8
8.	q_4	ϵ	Z_0	9
9.	q_5	ϵ	ϵ	10

Accepted

Since, stack is empty and final state is also accepting state. So, string 'abbbcc' is accepted.

16. **Define configuration of PDA. Design a PDA that accepts $L = \{a^{3n} b^n, n > 0\}$ and check the string aaaaaabb.** [2072 Chaitra]

The configuration of PDA is

$\delta(q, a, X) \rightarrow (q', k)$

where

- $q \rightarrow$ state of machine before input
- $a \rightarrow$ input symbol in Σ
- $X \rightarrow$ stack symbol on top of stack / symbol to be popped out.

The output of δ give pairs (q', k), where q' is new state

- q' is new state
- k is stack symbol that replaces X on the top of stack

Case I: If $k = \epsilon$, then X is popped out

Case II: If $k : X$, then stack is unchanged

Case III: If $k = YX$, then X is pushdown into stack and Y is on top of stack (or X is popped out and YX is pushed down into the stack)

Let, the required PDA be

$L(M) = (\theta, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_5\}$

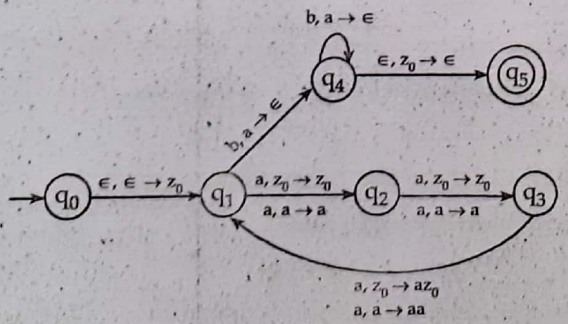
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_0, \in, \in), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_2, Z_0))$
3. $((q_2, a, Z_0), \quad (q_3, Z_0))$
4. $((q_3, a, Z_0), \quad (q_1, aZ_0))$
5. $((q_3, a, a), \quad (q_1, aa))$
6. $((q_1, b, a), \quad (q_4, \in))$
7. $((q_4, b, a), \quad (q_4, \in))$
8. $((q_4, \in, Z_0), \quad (q_5, \in))$
9. $((q_1, a, a), \quad (q_2, a))$
10. $((q_2, a, a), \quad (q_3, a))$

$\}$

States diagram



Test for input string 'abbbcc'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	aaaaaabb	$\in$	–
2.	q_1	aaaaaabb	Z_0	1
3.	q_2	aaaaabb	Z_0	2
4.	q_3	aaaabb	Z_0	3
5.	q_1	aaabb	a Z_0	4
6.	q_2	aabb	a Z_0	9
7.	q_3	abb	a Z_0	10
8.	q_1	bb	aa Z_0	5
9.	q_4	b	a Z_0	6
10.	q_4	$\in$	Z_0	7
11.	q_5	$\in$	$\in$	8

Accepted

Since, stack is empty and final state is also accepting state. So, string 'aaaaaabb' is accepted

17. **Design a pushdown automaton to accept $L = \{ww^R \in \{a, b\}^*$. Show how it accepts the string 'abbbba". [2072 Kartik, Back]**

Let, the required PDA be

$L(M) = (\theta, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where

$\theta = \{q_1, q_2, q_3, q_4\}$

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_1\}$

$Z_0 = \{Z_0\}$

$F = \{q_4\}$

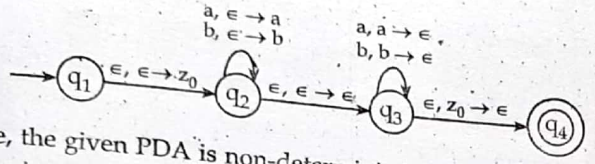
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_1, \in, \in), \quad (q_2, Z_0))$
2. $((q_2, a, \in), \quad (q_2, a))$
3. $((q_2, b, \in), \quad (q_2, b))$
4. $((q_2, \in, \in), \quad (q_3, \in))$
5. $((q_3, a, a), \quad (q_3, \in))$
6. $((q_3, b, b), \quad (q_3, \in))$
7. $((q_3, \in, Z_0), \quad (q_4, \in))$

$\}$

States diagram



Here, the given PDA is non-deterministic PDA so if any of the final states is accepting state, then string will be accepted. In above state diagram, we can see there is empty transition on input empty '∈' symbol between states q_2 and q_3. Any string can be resolved into strings like there exists empty symbol '∈' between two consecutive alphabet in string i.e. abba → ∈ a ∈ b ∈ b ∈ a ∈.

In many of the final state of given non-deterministic PDA, final state will be accepting state if empty symbol '∈' is taken as input exactly in middle or centre of the string. And final state will be rejecting state of any input empty symbol '∈' is taken before or after that except the beginning and the end. And after that, if matches the input with previously stored element in stack and POPS out it. And finally stack becomes empty and string is accepted if it follows ww^R.

Test for input string 'abbbba'

S.No.	State	Unread input	Stack	Transition used
1.	q_1	abbbba	∈	–
2.	q_2	abbbba	Z_0	1
3.	q_2	bbbba	a Z_0	2
4.	q_2	bbba	ba Z_0	3
5.	q_2	bba	bba Z_0	3
6.	q_3	bba	bba Z_0	4
7.	q_3	ba	ba Z_0	6
8.	q_3	a	a Z_0	6
9.	q_3	∈	Z_0	5
10.	q_4	∈	∈	7

Accepted

Note: In S. No. 5, transition $\delta(q_2, \in, \in) \rightarrow (q_3, \in)$ is used as centre of string is reached.

Since, stack is empty and the final state is also accepting state. So, string 'abbbba' is accepted.

18. Describe the transition function of push-down automata.

[2071 Shrawan, Back]

Please see 2072 Chaitra

19. Design a PDA that accepts language, $L = \{a^n b^{3n} : n \geq 1\}$. Test your design for string "abbb".

[2071 Chaitra]

Let, the required PDA be

$$L(M) = (\theta, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_5\}$

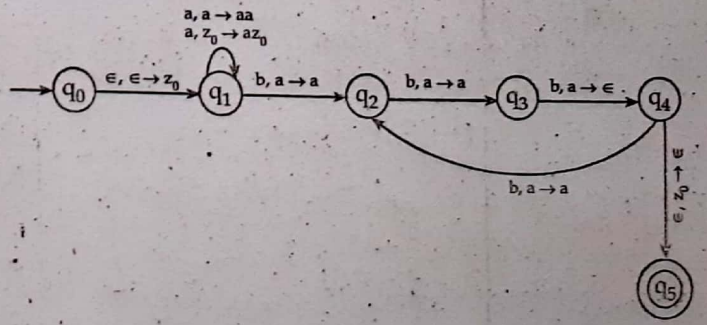
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_0, \in, \in), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_1, aZ_0))$
3. $((q_1, a, a), \quad (q_1, aa))$
4. $((q_1, b, a), \quad (q_2, a))$
5. $((q_2, b, a), \quad (q_3, a))$
6. $((q_3, b, a), \quad (q_4, a))$
7. $((q_4, b, a), \quad (q_2, a))$
8. $((q_4, \in, Z_0), \quad (q_5, \in))$

$\}$

States diagram



Test for input string 'abbb'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	abbb	$\in$	–
2.	q_1	abbb	Z_0	1
3.	q_1	bbb	a Z_0	2
4.	q_2	bb	a Z_0	4
5.	q_3	b	a Z_0	5
6.	q_4	$\in$	Z_0	6
7.	q_5	$\in$	$\in$	8

Accepted

Since, stack is empty and final state is also accepting state, so bb accepted.

20. Design a PDA that accepts all the palindromes defined over {a, b}*. Your design should accept strings like $\in$, a, b, aba, bab, abba, babab, etc. **[2070 Chaitra]**

Let, the required PDA (non-deterministic) be

$$L(M) = (\theta, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where

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

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_3\}$

and transition (δ) is defined as follows:

$\delta = \{$

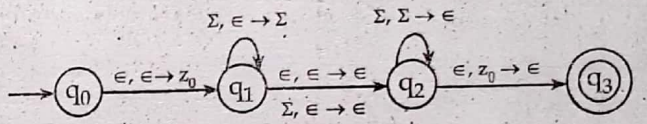
1. $((q_0, \in, \in), \quad (q_1, Z_0))$
2. $((q_1, \Sigma, \in), \quad (q_1, \Sigma))$
3. $((q_1, \in, \in), \quad (q_2, \in))$
4. $((q_1, \Sigma, \in), \quad (q_2, \in))$
5. $((q_2, \Sigma, \Sigma), \quad (q_2, \in))$
6. $((q_2, \in, Z_0), \quad (q_3, \in))$

$\}$

Here, transition $((q_1, \Sigma, \in), (q_1, \Sigma))$ means that on every input alphabet to state q_1, PDA doesn't POP out anything but it pushes that read input into stack and stays in q_1. All the other transitions including 'Σ' symbol can be understood in similar manner.

Here, the given PDA is non-deterministic. So, if any of the final states is accepting state, then string will be accepted. In between q_1 and q_2, PDA chooses transition $\in, \in \to \in$ for even-palindromes whereas it chooses transition $\Sigma, \in \to \in$ for odd palindromes non-deterministically which leads to accepting state. It is guaranteed that all the possible transitions between q_1 and q_2 except the mentioned above leads to rejecting state.

States diagram



Test for input string '$\in$'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	$\in$	$\in$	–
2.	q_1	$\in$	Z_0	1
3.	q_2	$\in$	Z_0	3
4.	q_3	$\in$	$\in$	6

Accepted

Test for input string 'a'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	a	$\in$	–
2.	q_1	a	Z_0	1
3.	q_2	$\in$	Z_0	4
4.	q_3	$\in$	$\in$	6

Accepted

Test for input string 'aba'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	aba	$\in$	-
2.	q_1	aba	Z_0	1
3.	q_1	ba	a Z_0	2
4.	q_2	a	a Z_0	4
5.	q_2	$\in$	Z_0	5
6.	q_3	$\in$	$\in$	6

Accepted

We can see string 'a' and 'aba' are odd palindromes, so PDA chooses transition 4 non-deterministically for accepting state.

Test for input string 'abba'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	abba	$\in$	-
2.	q_1	abba	Z_0	1
3.	q_1	bba	a Z_0	2
4.	q_1	ba	ba Z_0	2
5.	q_2	ba	ba Z_0	3
6.	q_3	a	a Z_0	5
7.	q_2	$\in$	Z_0	5
8.	q_3	$\in$	$\in$	6

Accepted

Here, string 'abba' is even palindrome. So, PDA chooses transition 3 non-deterministically for accepting states.

In all of the cases, stack is empty and accepting state is also reached, so, they are accepted.

21. Write context free grammar for the language $L = \{a^i b^j c^i, i, j > 0\}$ over the alphabet $\Sigma = \{a, b, c\}$. Use leftmost, rightmost derivation to generate strings "aabbb cc". Also draw parse three for the same.

[2075 Ashwin, Back]

Let, G (V, Σ, R, S)

where $V = \{S, a, b, c, B\}$

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

$R = \{$

$S \rightarrow aSc$

$S \rightarrow B$

$B \rightarrow bB / Bb / b$

$\}$

Now, let us generate string aabbbcc

Leftmost derivation

$S \xRightarrow[lm]{} aSc$

$\xRightarrow[lm]{} aaScc$

$\xRightarrow[lm]{} aaBcc$

$\xRightarrow[lm]{} aabBcc$

$\xRightarrow[lm]{} aabbBcc$

$\xRightarrow[lm]{} aabbbcc$

Parse tree

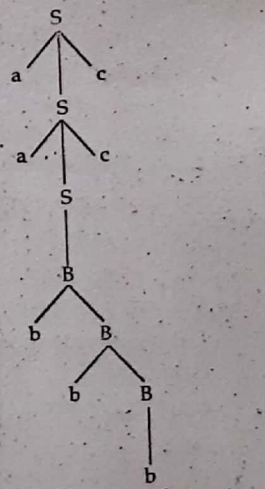


Fig. Leftmost derivation

Rightmost derivation

$S \xRightarrow[rm]{} aSc$

$\xRightarrow[rm]{} aaScc$

$\xRightarrow[rm]{} aaBcc$

$\xRightarrow[rm]{} aaBbcc$

$\xRightarrow[rm]{} aaBbbcc$

$\xRightarrow[rm]{} aabbbcc$

Parse tree

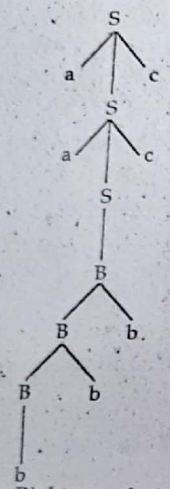


Fig. Rightmost derivation

22. Define pumping lemma for context free language prove that language L = {ww / w∈ {1, 0}* is not context free. [2070 Ashwin, Back]

Please see theory part and its example -2 for proof.

23. **Define context free grammar (CFG). Show that L = {$a^n b^{2n} c^{3n}$: n > 0} is not context free language by using pumping lemma for CFL.**

[2074 Chaitra]

For formal definition of CFG, please see theory part.

Pumping lemma states that, "if a language A is context free language, then it must have a pumping length 'p' such that any string 's' ∈ A having length is $S \geq p$ can be divided into 5 parts uvxyz such that following conditions must be true.

(1) $uv^i xy^i z \in A$ for every $i \geq 0$.

(2) $|vy| \geq 1$

(3) $|vxy| \leq p$

We will prove it by contradiction using pumping lemma.

Let us assume language L = {$a^n b^{2n} c^{3n}$: n > 0} be CFL (context free language)

Let, $S \in L = a^p b^{2p} c^{3p}$

Let p = 3

Then, $S = a^3 b^6 c^9$

$= aaa\ bbbbbb\ ccccccccc$

Dividing S into u, v, x, y and z, we get

Case I: When v and y contain only one kind of symbol is

$S = \underbrace{a}_{u}\ \underbrace{aa}_{v}\ \underbrace{bbbbbbccccc}_{x}\ \underbrace{cc}_{y}\ \underbrace{cc}_{z}$

checking for condition 1

Let, i = 2.

$S = uv^2 xy^2 z$

$= a\ aaaa\ bbbbbb\ ccccccccccc$

$= a^5 b^6 c^{11} \notin L$

Here, condition 1 fails we can see that 'b' is not twice as many as 'a' and 'c' is not thrice as many as 'a'.

Case II: when either v or y contains two different kind of symbol.

$S = \underbrace{ab}_{u}\ \underbrace{ab}_{v}\ \underbrace{bbbbb\ ccccc}_{x}\ \underbrace{cc}_{y}\ \underbrace{cc}_{z}$

checking for condition 1

Let, i = 2

$S = uv^2 xy^2 z$

$= aa\ abab\ bbbbb\ cccccccc \notin L$

Here, again condition 1 fails as we can see pattern is not followed.

So, none of the cases follow all three conditions of context free language as stated by pumping lemma. Therefore language L is not context free language (CLF).

24. **Define ambiguity in CFG write CFG for L = {w {a, b}* : w is a palindrome} and also draw parse tree for the derivation of any two strings of length even and odd.** **[2073, Chaitra]**

For ambiguity in CFG, please see 'ambiguous grammars in theory part.

Let G = (V, Σ, R, S) be the required CFG

where,

V = {S, a, b}

Σ = {a, b}

R = {

S → a Sa / bSb / a / b / ∈

(For derivation of string of even length, we put S → ∈ while terminating symbol 's' whereas we put S → a or b according to our requirement in case of string of odd length).

Let two strings of even length be 'baab' and 'abaaba'

Derivation of string 'baab'

$S \Rightarrow bSb$

$\Rightarrow baSab$

$\Rightarrow baab$

Parse tree

Derivation of string 'abaaba'

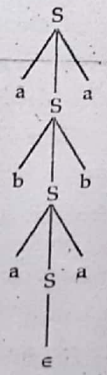
$S \Rightarrow aSa$

$\Rightarrow abSba$

$\Rightarrow abaSaba$

$\Rightarrow abaaba$

Parse tree



Let two strings of odd length be 'baaab'

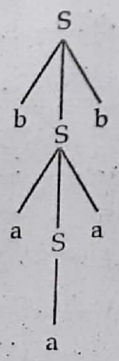
Derivation of string 'baaab'

$S \Rightarrow bSb$

$\Rightarrow baSab$

$\Rightarrow baaab$

Parse tree



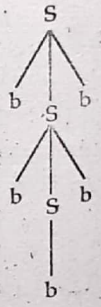
Derivation of string 'bbb bb'

$S \Rightarrow bSb$

$\Rightarrow bbSbb$

$\Rightarrow bbbbb$

Parse tree



25. Construct a CFG for the language $L = a^n b^{2n}$, $n > 0$ and use this grammar to generate string aabbbb. Also, construct the parse tree.

[2073 Shrawan, Back]

Let, $G = \{V, \Sigma, R, S\}$ be the required CFG for $L = a^n b^{2n} : n > 0$.

where,

$V = \{S, a, b\}$

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

and production R is defined as

$R = \{$

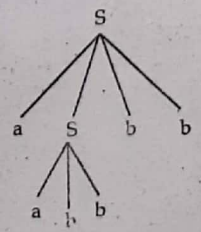
$S \rightarrow aSbb / abb$

Let us derive string aabbbb

$S \rightarrow aSbb$

$\rightarrow aabbbb$

Parse tree



26. Write a CFG for the regular expression $R = 0^* 1 (0 \cup 1)^*$ [2072 Chaitra]

Let, $G = (V, \Sigma, R, S)$ be the required CFG.

where

$V = \{S, A, B, 0, 1\}$

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

Production R is defined as

$R = \{$

$S \rightarrow A\,1\,B$

$A \rightarrow OA / \in$

$B \rightarrow OB / 1B / \in$

$\}$

Let, us derive string '01001'

$S \Rightarrow A\,1\,B$

$\Rightarrow 0\,A\,1\,B$

$\Rightarrow 0\,1\,B$

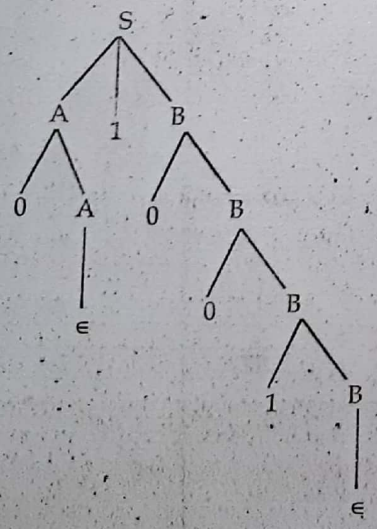
$\Rightarrow 0\,1\,0\,B$

$\Rightarrow 0\,1\,0\,0\,B$

$\Rightarrow 0\,1\,0\,0\,1\,B$

$\Rightarrow 0\,1\,0\,0\,1$

Parse tree



Let, us derive string '1001'

$S \Rightarrow A\,1\,B$

$\Rightarrow 1\,B$

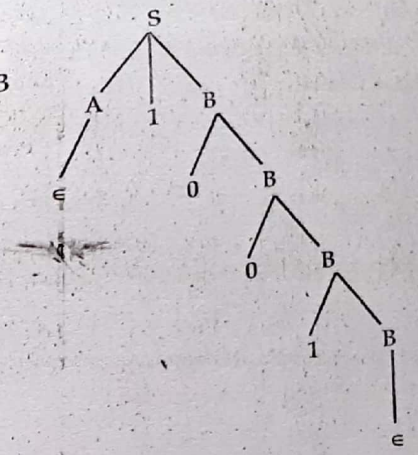
$\Rightarrow 1\,0\,B$

$\Rightarrow 1\,0\,0\,B$

$\Rightarrow 1\,0\,0\,1\,B$

$\Rightarrow 1\,0\,0\,1$

Parse tree



27. Use concept of closure property to prove that intersection of context free language is not context free. **[2072 Chaitra]**

We know that languages

$L_1 = \{ a^n b^m c^m / n, m \geq 0 \}$

and $L_2 = \{a^m b^m c^n / n, m \geq 0\}$

are context free. Now let see L, $n < 2 = \{a^n b^n c^n / n \geq 0\}$ is context free. We have proven that language $L = L_1 \cap L_2 = \{a^n b^n c^n / n \geq 0\}$ is not context free using pumping lemma. So, it is proven that intersection of context free languages is not context free.

28. Define the term ambiguity and inherent ambiguity in parse tree for a CFG given by G = (V, Σ, R, S) with V = {S}, Σ = {a} and production rules R is defined as

$S \rightarrow S\,S$

$S \rightarrow a$

Obtain the language L(G) generated by this grammar. **[2071 Chaitra]**

The term ambiguity refers to a grammar $G = (V, \Sigma, R, S)$ which is said to be ambiguous if any string $S \in L(G)$ has two or more than two leftmost derivation, rightmost derivation or parse trees for its derivation.

A "inherently ambiguous language" is a language for which no unambiguous grammar exists.

A language 'L' is said to be "inherently ambiguous in parse tree if every construction of parse tree that generates language L is ambiguous." Otherwise it is said to be unambiguous in parse tree.

Given, CFG

$G = (V, \Sigma, R, S)$

where $V = \{S\}$

$\Sigma = \{a\}$

$R = \{$

$S \rightarrow SS$

$S \rightarrow a$

$\}$

Let us start with start symbol 'S'

$S \Longrightarrow SS$

$\Longrightarrow aS$

$\Longrightarrow aSS$

$\Longrightarrow aaS$

$\Longrightarrow :$

$\Longrightarrow .$

$\Longrightarrow aa \ldots a^{n-1}S$

$\Longrightarrow aa \ldots a^{n-1}a$

$\Longrightarrow a^n$

Therefore, language L(G) generated by this grammar is

$L(G) = \{a^n : n \geq 2\}$

29. Using the pumping theorem for context free languages show that $L = \{a^n b^n c^n : n \geq 0\}$ is not context free. [2072 Kartik, Back]

Please see the example 1 of pumping lemma in theory part.

30. Write context free grammars (CFG) for the language $L_1 = \{a^m b^n c^n : m \geq 1, n \geq 1\}$ the $L_2 = \{a^n b^n c^m : m \geq 1, n \geq 1\}$. Do you think that $L = (L_1 \cap L_2)$ is also context free? If not prove that the language thus obtained is not context free by using pumping lemma for context free languages. [2070 Chaitra]

Given, languages L_1 and L_2 are

$L_1 = \{a^m b^n c^n : m \geq 1, n \geq 1\}$

$L_2 = \{a^n b^n c^m : m \geq 1, n \geq 1\}$

Let, $G_1 = (V, \Sigma, R, S)$ be the required CFG for language L_1 where

$V = \{S, A, B, a, b, c\}$

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

$R = \{$

$S \rightarrow AB$

$A \rightarrow aA / a$

$B \rightarrow b\,BC / bc$

$\}$

Let us generate string 'aaabbcc'

$S \Longrightarrow AB$

$\Longrightarrow a\,AB$

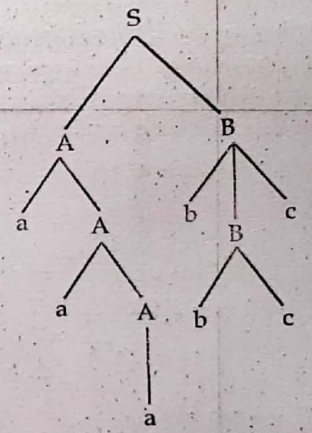
$\Longrightarrow aa\,AB$

$\Longrightarrow aaa\,B$

$\Longrightarrow aaa\,b\,Bc$

$\Longrightarrow aaa\,b\,bc\,c$

Parse tree



Let $G_2 = (V, \Sigma, R, S)$ be the required CFG for the language L_2 where

$V = \{S, M, N, a, b, c\}$

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

$R = \{$

$S \rightarrow MN$

$M \rightarrow a\,M\,b / a\,b$

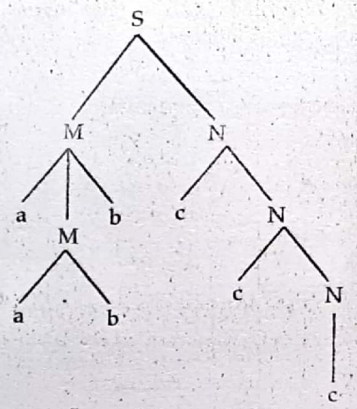
$N \rightarrow c\,N / c$

$\}$

Let us generate string 'aabbccc'

$S \Rightarrow MN$
$\Rightarrow aMbN$
$\Rightarrow aabbN$
$\Rightarrow aabbcN$
$\Rightarrow aabbccN$
$\Rightarrow aabbccc$

Parse tree



Let L be the intersection of the language L_1 and L_2 i.e. $L = L_1 \cap L_2 = \{a^n b^n c^n : n \geq 1\}$. Here, the language L is not context free language (CFL) we will prove it using pumping lemma.

Pumping lemma states that, "if a language L is context free language, then it must have a pumping length 'p' such that any string 's' $\in$ L having length $|S| \geq p$ can be divided into S parts uvxyz such that following conditions must be true.

(1) $uv^i xy^i z \in L$ for every $i \geq 0$
(2) $|vy| \geq 1$
(3) $|vxy| \leq p$

To prove: L is not CFL where $L = \{a^n b^n c^n : n \geq 1\}$

Proof: Let, $S \in L = a^p b^p c^p$

Let, $p = 3$

Then $S = a^3 b^3 c^3$
$= aaa\ bbb\ ccc$

Dividing S into u, v, x, y and z, we get.

Case I: When 'v' any 'y' contain only one type of symbol.

$S = \underbrace{a}_{u}\ \underbrace{a}_{v}\ \underbrace{abbbc}_{x}\ \underbrace{c}_{y}\ \underbrace{c}_{z}$

checking for condition 1,

Let, i = 2

$S = uv^2 xy^2 z$
$= a\ aa\ abbbc\ cc\ c$
$= a^4 b^3 c^4 \notin L$

Here, condition 1 fails.

Case II: Either 'v' or 'y' contains more than one kind of symbol.

$S = \underbrace{aa}_{u}\ \underbrace{ab}_{v}\ \underbrace{bbc}_{x}\ \underbrace{c}_{y}\ \underbrace{c}_{z}$

Checking for condition 1,

Let, i = 2

$S = uv^2 xy^2 z$
$= a\ a\ a\ b\ ab\ bbc\ ccc \notin L$

Here, it doesn't follow the pattern $a^n b^n c^n$. So, condition 1 fails. So, none of the cases satisfy all the three conditions of context free language as stated by pumping lemma. Therefore, language L which is intersection of two CFL L_1 and L_2 and is assumed to be CFL is not context free, language CFL using pumping lumping lemma. Hence, proved.

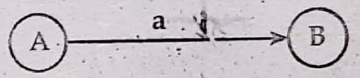
31. Consider the regular grammar G = (V, Σ, R, S) where,

$V = \{S, A, B, a, b\}, \Sigma = \{a, b\}$

$R = \{$
$S \rightarrow abA / B / baB / e$
$A \rightarrow bS / a$
$B \rightarrow aS$
$\}$ **[2070 Chaitra]**

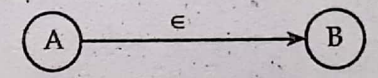
The rules for constructing equivalent finite automata for regular grammar

(a) For every production, $A \rightarrow aB$, we construct an edge.



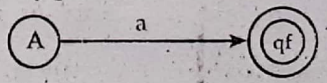
where $\delta(A, a) = B$

(b) For every production, $A \rightarrow B$ we construct an edge



where $\delta(A, \in) = B$

(c) For every production $A \rightarrow a$, we construct an edge



where qf is final state.

(d) For every production, $A \rightarrow e$ or $\in$, we make state the final state i.e.

Note that the number of states is equal to the number of non-terminals plus one

Given,

$R = \{$

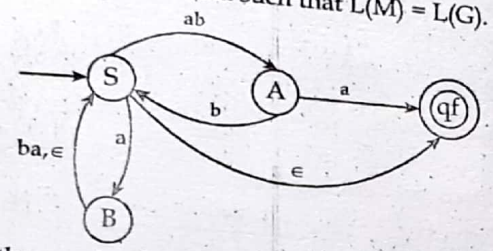
$S \rightarrow abA / B / baB / e$

$A \rightarrow bS / a$

$B \rightarrow aS$

$\}$

Equivalent finite automata such that L(M) = L(G).



This is the required equivalent finite automata.

32. Describe the transition function of push-down automata. **[2072 Shrawan, Back]**

The transition function (δ) of push-down automata (non-deterministic) is given as

$$\delta : (\theta \times \Sigma^* \times \Gamma^*) \rightarrow (\theta' \times x)$$

where

- θ → current state of the control unit
- Σ^* → current input symbol
- Γ^* → current symbol on top of the stack.
- θ' → next state
- x → string that is put on top of the stack in place of single symbol there before.

For example;

(a) The interpretation of $((q, a, z), (p, y)) \in \delta$ where $p, q \in \theta$, a is an alphabet, z and y in Γ^*) is that PDA whenever is in state q, with z on the top of the stack may read 'a' from the input tape, replace z by y on the top of the stack and enter state p.

(b) To push a symbol on the stack, just add it on the top of the stack. This can be achieved by the transition $((q, a, \in), (p, a))$.

(c) Similarly, to POP a symbol is to remove it from the top of the stack. The transition $((q, a, z), (p, \in))$ pops z from the stack.

(d) Suppose PDA is in state 'p', reads input 'a' with 'z' on top of stack. And PDA wants to stay in same state and do nothing neither POP nor push. Then, it can be achieved by following transition

$((p, q, z), (p, z))$

Chapter - 4

Turing Machine

Introduction

A Turing machine is like a pushdown automata with infinite input tape. It is the next model of computation under machine based approach and context sensitive or phase structure grammar in the corresponding model under grammatical approach.

A Turing machine consists of a finite control, a tape and a head that can be used for reading or writing on that tape. The tape consists of a infinite long series of "squares" each of which can hold a single symbol. The 'tape head' or 'read write head' can read a symbol from the tape, write a symbol to the tape and more one square in either direction. There are two types of Turing machines.

(a) Deterministic Turing machine.

(b) Non-deterministic Turing machine.

Here, we will discuss about deterministic Turing machines only.

Mathematical definition of Turing machine

A Turing machine M is a 7-tuple

$(\theta, \Sigma, \Gamma, b, \delta, q_0, F)$

where, θ is a finite set of states

Σ is a finite set of symbols "input alphabets"

Γ is a finite set of symbols "tape alphabets"

δ is the partial transition function $\delta : \theta \times \Gamma \rightarrow \theta \times \Gamma' \times (L, R, N)$

b is a symbol called 'blank symbol'

$q_0 \in \theta$ is the initial state.

$F \subseteq \theta$ is a set of final states.

Operation of Turing machine

In essence a Turing machine consists of a finite state control unit and a tape communication between two is provided by a single head which

reads symbols from the tape and is also used to change the symbols on the tape. The control unit operates in discrete steps. At each step, it performs three functions in a way that depends on its current state and the tape symbol currently scanned by the read/write head.

(a) Put the control unit in a new state.

(b) Write a symbol in the tape square currently scanned, replacing the one already there.

(c) Move the read/write head one tape square to the left or right.

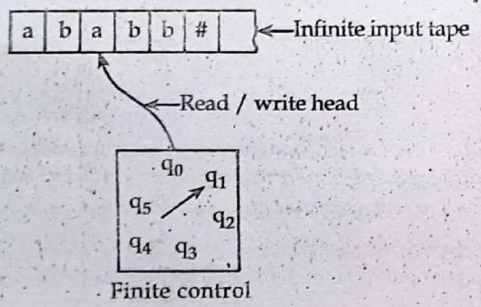


Fig. Pictorial representation of Turing machine

Note that all Turing machines discussed here are the deterministic ones. Standard Turing machine is a special case of non-deterministic Turing machine in which for each (q, a) the set $\{(q_i, a_i, M_i)\}$ in next moves is a singleton set or empty. In case of empty, Turing machine rejects or halts.

Examples

1. **Design a Turing machine that accepts the language L = {(a + b)* ab (a + b)* i.e. strings containing 'ab' as substring.**

Let the required Turing machine be

$$T(M) = (\theta, \Sigma, \Gamma, b, \delta, q_0, F)$$

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

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

$\Gamma = \{a, b, \#\}$

$b = \{\#\}$

$q_0 = \{q_0\}$

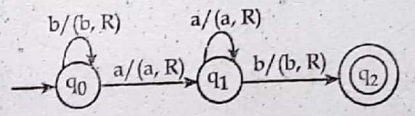
$F = \{q_2\}$

and transition (δ) is defined as follows:

States / Inputs	a	b	#
q_0	(q_1, a, R)	(q_0, b, R)	–
q_1	(q_1, a, R)	(q_2, b, R)	–
q_2	–	–	–

Note: '–' is for undefined move / rejected state.

States diagram



Let us process string # aaa #

$(q_0, \# \underline{a}aa\ \#) \vdash M (q_1, \# a \underline{a} a\ \#)$

$\vdash M (q_1, \# a a \underline{a}\ \#)$

$\vdash M (q_1, \# a a a \underline{\#})$ Rejected

Since, q_1 is not accepting state, so string 'aaa' is rejected.

Let us process string # bab #

$(q_0, \# \underline{b} a b\ \#) \vdash M (q_0, \# b \underline{a} b\ \#)$

$\vdash M (q_1, \# b a \underline{b}\ \#)$

$\vdash M (q_2, \# b a b\ \#)$ Accepted

Since, q_2 is accepting state after reading all inputs, so string 'bab' is accepted.

Note: All the moves which aren't shown leads towards not accepting state.

2. **Design a Turing machine that accepts all the palindrome of strings over alphabet {a, b}.**

There are various steps involved in processing palindromes. The Turing machine TM scans the first symbol of input tape (a or b), erases it and change state (q_1 or q_2). TM scans the remaining part without changing tape symbol it encounters '#'. The read / write head moves to the left. If the rightmost symbol tallies with the leftmost symbol (which can be erased but remembered), the rightmost symbol is erased otherwise TM halts. The read / write head moves to the left until '#' is encountered. The above steps are repeated after changing the states suitably. The transition table is shown below.

Let, the required Turing machine be

$T(M) = (\theta, \Sigma, \Gamma, b, \delta, q_0, F)$

where, $\theta = \{A, B, C, D, E, F, h\}$

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

$\Gamma = \{a, b, \#\}$

$b = \{\#\}$

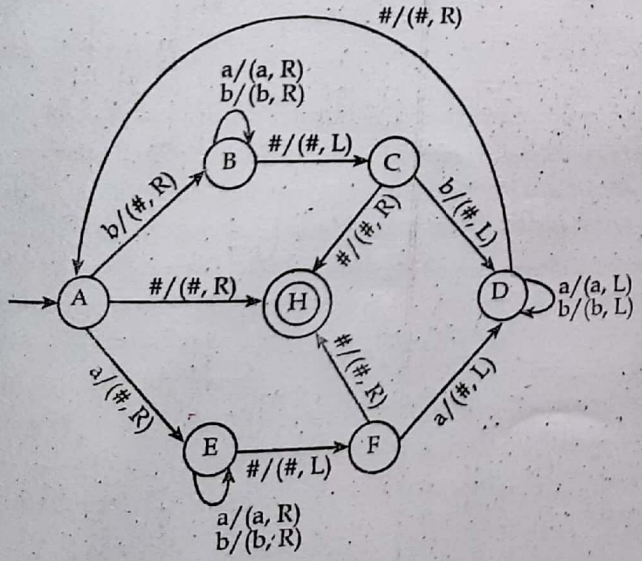
$q_0 = \{A\}$

$F = \{h\}$

and transition (δ) is defined as

States / Inputs	a	b	#
A	(E, #, R)	(B, #, R)	(H, #, R)
B	(B, a R)	(B, b, R)	(C, #, L)
C	-	(D, #, L)	(H, #, R)
D	(D, a, L)	(D, b, L)	(A, #, L)
E	(E, a, R)	(E, b, R)	(F, #, L)
F	(D, #, L)	-	(H, #, R)
H	-	-	-

States diagram



Test for string # ababa #

(A, # a b aba #) ⊢M (E, # # b a b a #)
⊢M (E, # # b a b a #)
⊢M (E, # # b a b a #)
⊢M (E, # # b a b a #)
⊢M (E, # # b a b a #)
⊢M (F, # # b a b a #)
⊢M (D, # # b a b # # #)
⊢M (D, # # b a b # # #)
⊢M (D, # # b a b # # #)
⊢M (D, # # b a b # # #)
⊢M (A, # # b a b # # #)
⊢M (B, # # # a b # # #)
⊢M (B, # # # a b # # #)
⊢M (B, # # # a b # # #)
⊢M (C, # # # a b # # #)
⊢M (D, # # # a # # # #)
⊢M (D, # # # a # # # #)
⊢M (A, # # # a # # # #)
⊢M (E, # # # # # # # #)
⊢M (F, # # # # # # # #)
⊢M (H, # # # # # # # #)

Since, it reaches final state H, string 'ababa' is accepted.

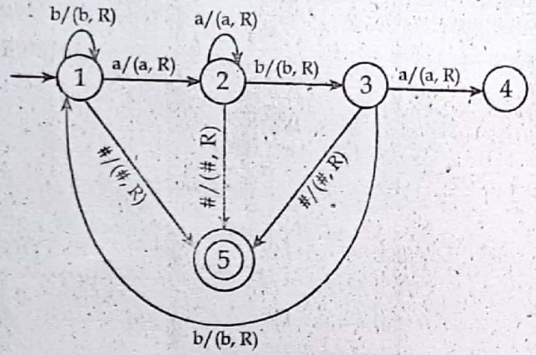
Let us process string # abab #

(A, #, a b a b #) ⊢M (E, # # b a b #)
⊢M (E, # # b a b #)
⊢M (E, # # b a b #)
⊢M (E, # # b a b #)
⊢M (F, # # b a b #)

Since, there is no defined move on input to F, it is one of the rejecting state so, string 'abab' is rejected.

3. Turing machine that accepts input over Σ = {a, b} that doesn't contain 'aba' as substring.

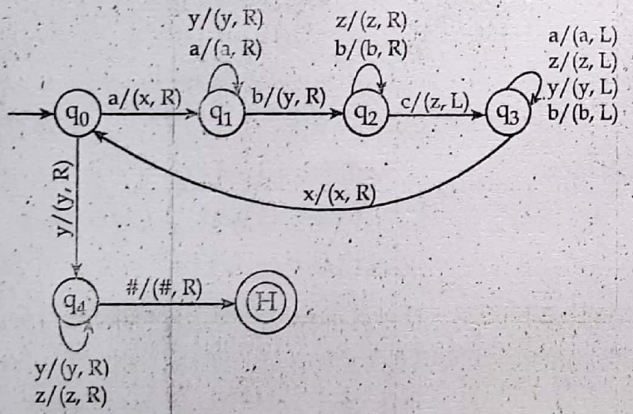
States, diagram



Note: Readers are requrested to define Turing machine and transition (δ) themselves.

4. Turing machine that accepts language L = {$a^n b^n c^n / x \geq 1$} over the alphabet {a, b, c}

States diagram



Note: Readers are requested to define Turing machine and transistion (δ) themselves.

5. Design a truing machine that recognizes L + {$0^n 1^n$, n ≥ 0}

Let, the required Turing machine be

T(M) = (θ, Σ, Γ, b, δ, q_0, F)

where, θ = {A, B, C, D, H}

Σ = {0, 1}

Γ = {0, 1, x, y, #}

q_0 = {A}

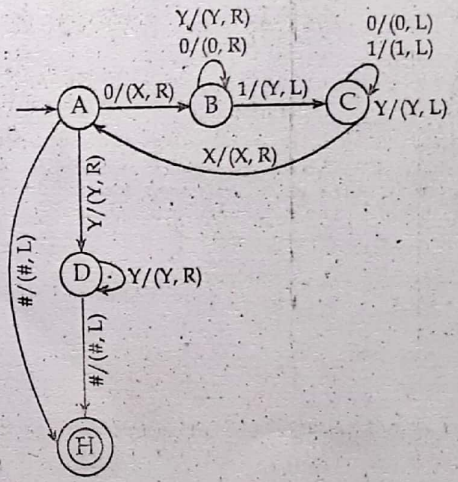
F = {H}

b = {#}

and δ is defined as follows:

States/Inputs	0	1	#	X	Y
A	(B, X, R)	-	(H, #, L)	-	(D, Y, R)
B	(B, 0, R)	(C, y, L)	-	-	(B, Y, R)
C	(C, 0, L)	-	-	(A, X, R)	(C, Y, L)
D	-	-	(H, #, L)	-	(D, Y, R)
H	-	-	-	-	-

States diagram



Let us process string #0011#

(A, #0011#) ⊢M (B, #X011#)

⊢M (B, #X011#)

⊢M (C, #X0Y1#)

⊢M (C, #X0Y1#)

⊢M (A, #X0Y1#)

$\vdash$ M (B, # X X Y 1 #)
$\vdash$ M (B, # X X Y 1 #)
$\vdash$ M (C, # X X Y Y #)
$\vdash$ M (C, # X X Y y #)
$\vdash$ M (A, # X X Y Y #)
$\vdash$ M (D, # X X Y Y #)
$\vdash$ M (D, # X X Y Y #)
$\vdash$ M (H, # X X Y Y #) Accepted

Since, H is accepting state, so string '0011' is accepted

Let us process string # 0 0 1 #

(A, # 0 0 1 #) $\vdash$ M (B, # X 0 1 #)
$\vdash$ M (B # X 0 1 #)
$\vdash$ M (C, # X 0 Y #)
$\vdash$ M (C, # X 0 Y #)
$\vdash$ M (A, # X 0 Y #)
$\vdash$ M (B, # X X Y #)
$\vdash$ M (B, # X X Y #) Rejected

Since, there is no transition state for input '#' on state B, string '001' is rejected.

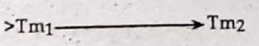
Combining turing machine

We can build complex Turing machines by combining simple ones. In this respect, a simple Turing machine can work as Turing machine. Tm_1 is a Turing machine that is not known to hang. Then Tm_1 can be made part of a large machine Tm as follows. Tm prepares some string as input to Tm_1, placing it near the right end of non-blank portion of the tape, passes control to Tm_1 with read/write head just beyond the end of that string and finally retrieves control from Tm_1 when Tm_1 has finished its computing. It is guaranteed that Tm_1 will never interfere with the operational computation of Tm. This is like a function, when we call a function (say Tm_1). In the main function of a simple (c) program the control simply returns to main program after function Tm_1, performs its calculations.

Rules for combining Turing machine

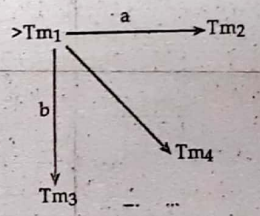
Now we adopt a graphical representation to show the combining of Turing machines. In fact this representation is the same as that of a finite automata. Individual machines are like the states of the automata and these machines are connected in the same way as the states of automata.

The connection from one machine to another will not be pursued until the first machine halts, the other machine is then started from its initial state with the tape one head position as they were left by the first machine. This can be shown by the figure.



Here, Tm_1 machine starts computing (that is initial state in finite automata) and when Tm_1 halts, control, now goes to machine Tm_2. Now consider the following machine.

This machine starts computing with machine Tm_1. Now when Tm_1 halts the symbol on the tape under the current head position can be a, b or #. Then fig (i) shows when Tm_1 halts, the control goes to machine Tm_2, Tm_3 or Tm_4 respectively, according to symbol under current head position which is a or b or #.



Here we are going to discuss certain symbols to represent simple machines. These symbols will be used further.

(a) Fig (i) is the basic machine which moves, towards left by one cell during scanning any input (6) symbol. State change is also possible.

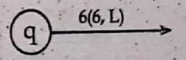


Fig. (i)

(b) Fig (ii) is the basic machine moves towards right by one cell, when current input symbol is blank or non-blank (6).

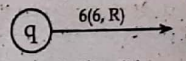


Fig. (ii)

(c) Fig (iii) is the basic machine, which moves towards left till the first blank symbol is encountered.

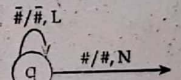


Fig. (iii)

(d) Fig. (iv) is the basic machine, which moves towards right until first blank symbol is encountered.

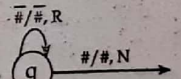


Fig. (iv)

(e) Fig. (v) is the basic machine, which moves towards left until non-blank symbol is encountered.

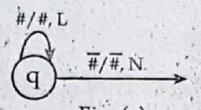


Fig. (v)

(f) Fig (vi) is the basic machine, which moves towards right until first non-blank symbol is encountered.

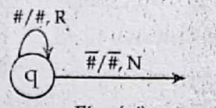


Fig. (vi)

6. Design a Turing machine which works as eraser.

Let us first analyse the problem, we want to design a Turing machine which works as eraser, that is if we pass input # abb # then output should be ##, that is null string.

Let us Turing machine be

$Tm = (\theta, \Sigma, \Gamma, b, \delta, q_0, F)$

where, $\theta = \{q_0, q_1, h\}$

$\Sigma = \bar{\#}$ = non-blank symbols

$\Gamma = \{\bar{\#}, \#\}$

$b = \#$ = blank-symbol

$q_0 = \{q_0\}$

$T = \{h\}$

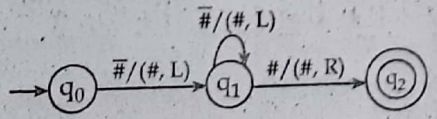
and transition (δ) is defined as

States/Inputs	$\bar{\#}$	#
q_0	$(q_1, \#, L)$	-
q_1	$(q_1, \#, L)$	$(h, \#, R)$

Note: The moves like $(q_0, \#)$ etc which are unaddressed leads towards rejecting state.

And $(q_0, \bar{\#})$ means non-blank symbol input to the state 'q_0'.

State diagram



Here, it is preassumed that there is at least one non-blank symbol in the string.

Lets, process the string # ababa #

$(q_0, \# a b a b \underline{a} \#) \vdash M (q_1, \# abab \underline{\#} \#)$

$\vdash M (q_1, \# aba \underline{\#} \#\#\#)$

$\vdash M (q_1, \# ab \underline{\#} \#\#\#\#)$

$\vdash M (q_1, \# \underline{a} \#\#\#\#\#)$

$\vdash M (q_1, \underline{\#} \#\#\#\#\#\#)$

$\vdash M (h_1, \underline{\#}\#\#\#\#\#\#)$

Let's process the string # 011 #

$(q_0 011 \underline{\#}) \vdash M (q_1, \# 01 \underline{\#} \#)$

$\vdash M (q_1, \# 0 \underline{\#} \#\#)$

$\vdash M (q_1, \# \underline{\#} \#\#\#)$

$\vdash M (h_1, \#\underline{\#}\#\#\#)$

Note: (Note: we can process string from any (end is leftmost and rightmost, it only depends on our transition (δ), we have constructed.) Here, it is for rightmost end.

7. Design a Turing machine that computes the following f(n, m) = n+m.

Let us decide the policy to show input numbers # I # represents 1, # I I # represents 2 and so on we assume that input numbers (i.e. n and m) are written on tape separated by #. that is, if we want to add 2 and 3, the input string will be # II # III # which gives output as follows:

$f(2, 3) = 2 + 3 = 5 =$ # IIIII #

When we analyse the output, then it becomes clear that for constructing Turing machine f(n, m) = n + m, we have to replace blank symbol '#' between the inputs by I and the last I by #.

Let the required Turing machine be

$T(M) = (\theta, \Sigma, \Gamma, b, \delta, q_0, F)$

where $\theta = \{q_0, q_1, q_2, q_3, h\}$

$\Sigma = \{I\}$

$\Gamma = \{I, \#\}$

$b = \{\#\}$

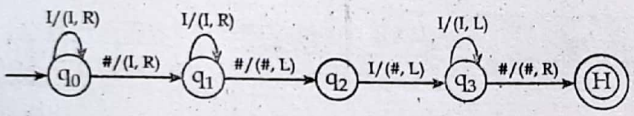
$q_0 = \{q_0\}$

$F = \{h\}$

and δ is defined as follows:

States/Inputs	I	#
q_0	(q_0, I, R)	(q_1, I, R)
q_1	(q_1, I, R)	$(q_2, \#, L)$
q_2	$(q_3, \#, L)$	–
q_3	(q_3, I, L)	$(h, \#, R)$
h	–	–

Here, undefined moves leads to rejecting states or halting condition of Turing machine.



Let us, process string # II # III#

$(q_0, \# \underline{I}I \# III \#)$ $\vdash\!\!-M\ (q_0, \# I\underline{I} \# III \#)$

$\vdash\!\!-M\ (q_0, \# II \underline{\#} III \#)$

$\vdash\!\!-M\ (q_1, \# III\underline{I}II \#)$

$\vdash\!\!-M\ (q_1, \# IIII\underline{I}I \#)$

$\vdash\!\!-M\ (q_1, \# IIIII\underline{I} \#)$

$\vdash\!\!-M\ (q_1, \# IIIIII \underline{\#})$

$\vdash\!\!-M\ (q_2, \# IIIII\underline{I} \#)$

$\vdash\!\!-M\ (q_3, \# IIII\underline{I} \# \#)$

$\vdash\!\!-M\ (q_3, \# III\underline{I}I \# \#)$

$\vdash\!\!-M\ (q_3, \# II\underline{I}II \# \#)$

$\vdash\!\!-M\ (q_3, \# I\underline{I}III \# \#)$

$\vdash\!\!-M\ (q_3, \# \underline{I}IIII \# \#)$

$\vdash\!\!-M\ (q_3, \underline{\#} IIIII \# \#)$

$\vdash\!\!-M\ (q_3, \# \underline{I}IIII \# \#)$

Let us process string # # # (i.e. m = 0, n = 0)

$(q_0, \# \underline{\#} \#)$ $\vdash\!\!-M\ (q_1, \# I \underline{\#})$

$\vdash\!\!-M\ (q_2, \# \underline{I} \#)$

$\vdash\!\!-M\ (q_3, \underline{\#} \# \#)$

$\vdash\!\!-M\ (h, \# \underline{\#} \#)$

So, output is 0 as m = 0, n = 0.

8. Construct q Turing machine that performs subtraction operation.

$$f(m, n) = \begin{cases} m - n, & \text{if } m \geq n \\ 0 & \text{if } m \leq n \end{cases}$$

Let the required Turing machine be

$T(M) = (\theta, \Sigma, \Gamma, b, \delta, q_0, F)$

where $\theta = \{q_0, q_1, q_2, q_3, q_4, q_F, h\}$

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

$\Gamma = \{0, 1, \#\}$

$b = \{\#\}$

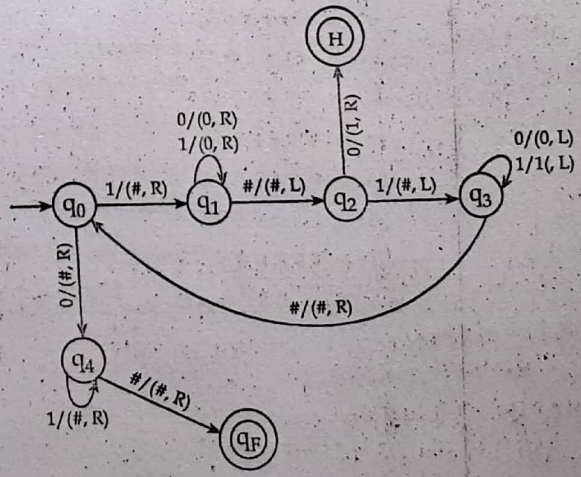
$q_0 = \{q_0\}$

$F = \{q_F, h\}$

and transition (δ) is defined as follows:

States/Inputs	0	1	#
q_0	$(q_4, \#, R)$	$(q_1, \#, R)$	–
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	$(q_2, \#, L)$
q_2	$(H, 1, R)$	(q_3, H, L)	–
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_0, \#, R)$
q_4	–	$(q_4, \#, R)$	$(q_F, \#, R)$
H	–	–	–
qf	–	–	–

States diagram



Let us process string # 1101 #. Hence, we use 11 to represent 2, 1 to represent 1 and so on. And 0 is used to separate input. Algorithm is we cancel on 1 one RHS. for one '1' on LHS separated by 0.

$(q_0, \#, \underline{1}\,1\,0\,1\,\#) \vdash_M (q_1, \#\,\#\,\underline{1}\,0\,1\,\#)$
$\vdash_M (q_1, \#\,\#\,1\,\underline{0}\,1\,\#)$
$\vdash_M (q_1, \#\,\#\,1\,0\,\underline{1}\,\#)$
$\vdash_M (q_1, \#\,\#\,1\,0\,1\,\underline{\#})$
$\vdash_M (q_2, \#\,\#\,1\,0\,\underline{1}\,\#)$
$\vdash_M (q_3, \#\,\#\,1\,\underline{0}\,\#\,\#)$
$\vdash_M (q_3, \#\,\#\,\underline{1}\,0\,\#\,\#)$
$\vdash_M (q_3, \#\,\#\,\underline{\#}\,1\,0\,\#\,\#)$
$\vdash_M (q_0, \#\,\#\,\underline{1}\,0\,\#\,\#)$
$\vdash_M (q_1, \#\,\#\,\#\,\underline{0}\,\#\,\#)$
$\vdash_M (q_1, \#\,\#\,\#\,0\,\underline{\#}\,\#)$
$\vdash_M (q_2, \#\,\#\,\#\,\underline{0}\,\#\,\#)$
$\vdash_M (H, \#\,\#\,\#\,1\,\underline{\#}\,\#)$

Let us process string # 1011 #

$(q_0, \#\,\underline{1}\,0\,1\,1\,\#) \vdash_M (q_1, \#\,\#\,\underline{0}\,1\,1\,\#)$
$\vdash_M (q_1, \#\,\#\,0\,\underline{1}\,1\,\#)$
$\vdash_M (q_1, \#\,\#\,0\,1\,\underline{1}\,\#)$
$\vdash_M (q_1, \#\,\#\,0\,1\,1\,\underline{\#})$
$\vdash_M (q_2, \#\,\#\,0\,1\,\underline{1}\,\#)$
$\vdash_M (q_3, \#\,\#\,0\,\underline{1}\,\#\,\#)$
$\vdash_M (q_3, \#\,\#\,\underline{0}\,1\,\#\,\#)$
$\vdash_M (q_3, \#\,\#\,\underline{\#}\,0\,1\,\#\,\#)$
$\vdash_M (q_0, \#\,\#\,\underline{0}\,1\,\#\,\#)$
$\vdash_M (q_4, \#\,\#\,\#\,\underline{1}\,\#\,\#)$
$\vdash_M (q_4, \#\,\#\,\#\,\#\,\underline{\#}\,\#)$
$\vdash_M (q_F, \#\,\#\,\#\,\#\,\#\,\underline{\#})$

9. **Construct q Turing machine that performs multiplication operation f(m, n) = m × n.**

Let the required Turing machine be

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

where $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, h\}$

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

$\Gamma = \{0, 1, Y, \#\}$

$b = \{\#\}$

$q_0 = \{q_0\}$

$F = \{h\}$

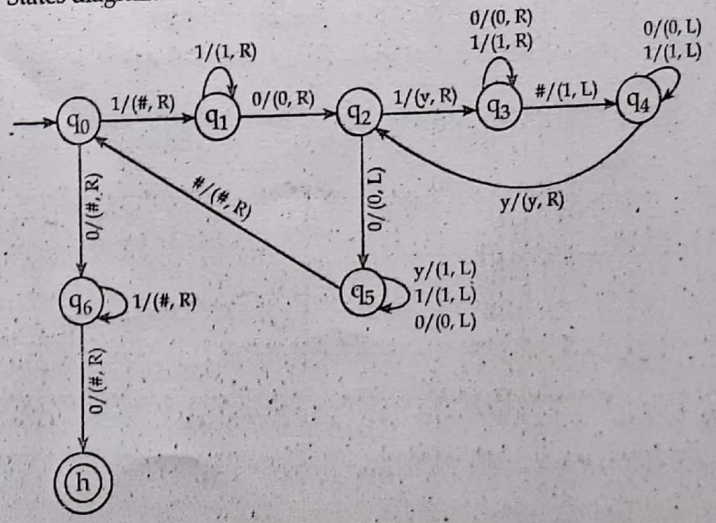
and transition δ is defined as follows:

States/Inputs	0	1	#	Y
q_0	$(q_6, \#, R)$	$(q_1, \#, R)$	–	–
q_1	$(q_2, 0, R)$	$(q_1, 1, R)$	–	–
q_2	$(q_5, 0, L)$	(q_3, Y, R)	–	–
q_3	$(q_3, 0, R)$	$(q_3, 1, R)$	$(q_4, 1, L)$	–
q_4	$(q_4, 0, L)$	$(q_4, 1, L)$	–	(q_2, Y, R)
q_5	$(q_5, 0, L)$	$(q_5, 1, L)$	$(q_0, \#, R)$	$(q_5, 1, L)$
q_6	$(h, \#, R)$	$(q_6, \#, R)$	–	–
h	–	–	–	–

Here input on tape will be like #1101110# for m = 2 and n = 3

And a zero '0' is added at the end of input

States diagram



Note: In some transition '*' after closing paranthesis ') ' denotes as extra blank symbol. On right hand side of infinite input tape is used.

And

$\vdash M(q_3, \#\# 10\underline{4}10\ \#)$

$\vdash M(q_3, \#\# 104\underline{1}0\ \#)$

$\vdash M(q_3, \#\# 10410\ \underline{\#})$

is equivalent to writing

$\vdash M(q_3, \#\# 10\underline{4}10\ \#)$

$\vdash^{*} M(q_3, \#\# 10410\ \underline{\#})$

Let us process string # 110110 # i.e. m = 2 and n = 2

$(q_0, \#, \underline{1}\,1\,0\,1\,1\,0\ \#)$ $\vdash M(q_1, \#\#\underline{1}\,0\,1\,1\,0\ \#)$

$\vdash M(q_1, \#\#1\underline{0}\,1\,1\,0\ \#)$

$\vdash M(q_2, \#\#10\underline{1}\,1\,0\ \#)$

$\vdash M(q_3, \#\#10Y\underline{1}\,0\ \#)$

$\vdash^{*} M(q_3, \#\#10Y10\ \underline{\#})$

$\vdash M(q_4, \#\#10Y1\underline{0}\,1\ \#)^{*}$

$\vdash^{*} M(q_4, \#\#10\underline{Y}\,1\,0\,1\ \#)$

$\vdash M(q_2, \#\#10Y\underline{1}\,0\,1\ \#)$

$\vdash M(q_3, \#\#10YY\underline{0}\,1\ \#)$

$\vdash^{*} M(q_3, \#\#10YY01\ \underline{\#})$

$\vdash M(q_4, \#\#10YY0\underline{1}\,1\ \#)^{*}$

$\vdash^{*} M(q_4, \#\#10Y\underline{Y}\,0\,1\,1\ \#)$

$\vdash M(q_2, \#\#10YY\underline{0}\,1\,1\ \#)$

$\vdash M(q_5, \#\#10Y\underline{Y}\,0\,1\,1\ \#)$

$\vdash M(q_5, \#\#10\underline{Y}\,1\,0\,1\,1\ \#)$

$\vdash M(q_5, \#\#1\underline{0}\,1\,1\,0\,1\,1\ \#)$

$\vdash M(q_5, \#\#\underline{1}\,0\,1\,1\,0\,1\,1\ \#)$

$\vdash M(q_5, \#\#\underline{\#}\,1\,0\,1\,1\,0\,1\,1\ \#)$

$\vdash M(q_0, \#\#\underline{1}\,0\,1\,1\,0\,1\,1\ \#)$

$\vdash M(q_1, \#\#\underline{0}\,1\,1\,0\,1\,1\ \#)$

$\vdash M(q_2, \#\#\#0\underline{1}\,1\,0\,1\,1\ \#)$

$\vdash M(q_3, \#\#\#0Y\underline{1}\,0\,1\,1\ \#)$

$\vdash^{*} M(q_3, \#\#\#0Y10\underline{1}\,1\ \#)$

$\vdash M(q_4, \#\#\#0Y10\underline{1}\,1\ \#)^{*}$

$\vdash^{*} M(q_4, \#\#\#0\underline{Y}\,1\,0\,1\,1\ \#)$

$\vdash M(q_2, \#\#\#0Y\underline{1}\,0\,1\,1\ \#)$

$\vdash M(q_3, \#\#\#0YY\underline{0}\,1\,1\ \#)$

$\vdash^{*} M(q_3, \#\#\#0YY01\underline{1}\ \#)$

$\vdash M(q_4, \#\#\#0YY0\underline{1}\,1\,1\ \#)^{*}$

$\vdash M(q_4, \#\#\#0Y\underline{Y}\,0\,1\,1\,1\ \#)$

$\vdash M(q_2, \#\#\#0YY\underline{0}\,1\,1\,1\ \#)$

$\vdash M(q_5, \#\#\#0Y\underline{Y}\,0\,1\,1\,1\ \#)$

$\vdash M(q_5, \#\#\#0\underline{Y}\,1\,0\,1\,1\,1\ \#)$

$\vdash M(q_5, \#\#\#\underline{0}\,1\,1\,0\,1\,1\,1\ \#)$

$\vdash M(q_5, \#\#\underline{\#}\,0\,1\,1\,0\,1\,1\,1\ \#)$

$\vdash M(q_0, \#\#\#\underline{0}\,1\,1\,0\,1\,1\,1\ \#)$

$\vdash M(q_6, \#\#\#\#\underline{1}\,1\,0\,1\,1\,1\ \#)$

$\vdash M(q_6, \#\#\#\#\#\underline{1}\,0\,1\,1\,1\ \#)$

$\vdash M(q_6, \#\#\#\#\#\#\underline{0}\,1\,1\,1\ \#)$

$\vdash M(h, \#\#\#\#\#\#\#\underline{1}\,1\,1\ \#)$

10. Design a Turing machine which accepts the language L = {w ∈ {a, b)*, w has equal number of a's and b's.

Here, the algorithm is to delete in pairs (a, b) or (b, a) iteratively.

Let, the required Turing machine be

$T(M) = (\theta, \Sigma, \Gamma, b, \delta, q_0, F)$

where $\theta = \{q_0, q_1, q_2, q_3, q_4\}$

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

$T = \{a, b, \#, X\}$

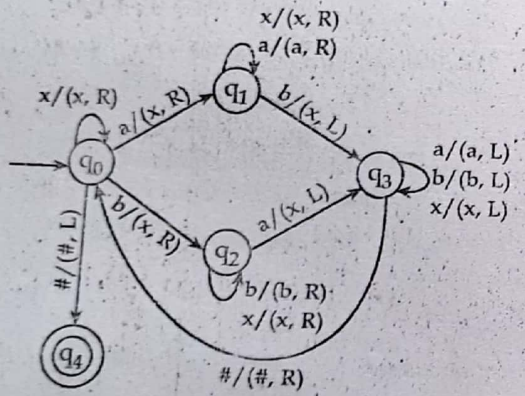
$q_0 = \{q_0\}$

$F = \{q_4\}$

and transition δ is defined as follows:

States/Inputs	a	b	#	X
q_0	(q_1, X, R)	(q_2, X, R)	$(q_4, \#, L)$	(q_0, X, R)
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	$(q_3, 0, L)$	(q_3, b, L)	$(q_0, \#, R)$	(q_3, X, L)
q_4	–	–	–	–

States diagram



Let us process string # babaab #

$(q_0, \#\,\underline{b}\,a\,b\,a\,a\,b\,\#)$ $\vdash M\,(q_2, \#\,X\,\underline{a}\,b\,a\,a\,b\,\#)$

$\vdash M\,(q_3, \#\,\underline{X}\,X\,b\,a\,a\,b\,\#)$

$\vdash M\,(q_3, \underline{\#}\,X\,X\,b\,a\,a\,b\,\#)$

$\vdash M\,(q_0, \#\,\underline{X}\,X\,b\,a\,a\,b\,\#)$

$\vdash M\,(q_0, \#\,X\,\underline{X}\,b\,a\,a\,b\,\#)$

$\vdash M\,(q_0, \#\,X\,X\,\underline{b}\,a\,a\,b\,\#)$

$\vdash M\,(q_2, \#\,X\,X\,X\,\underline{a}\,a\,b\,\#)$

$\vdash M\,(q_3, \#\,X\,X\,\underline{X}\,X\,a\,b\,\#)$

$\vdash M\,(q_3, \#\,X\,\underline{X}\,X\,X\,a\,b\,\#)$

$\vdash M\,(q_3, \#\,\underline{X}\,X\,X\,X\,a\,b\,\#)$

$\vdash M\,(q_3, \underline{\#}\,X\,X\,X\,X\,a\,b\,\#)$

$\vdash M\,(q_0, \#\,\underline{X}\,X\,X\,X\,a\,b\,\#)$

$\vdash M\,(q_0, \#\,X\,\underline{X}\,X\,X\,a\,b\,\#)$

$\vdash M\,(q_0, \#\,X\,X\,\underline{X}\,X\,a\,b\,\#)$

$\vdash M\,(q_0, \#\,X\,X\,X\,\underline{X}\,a\,b\,\#)$

$\vdash M\,(q_0, \#\,X\,X\,X\,X\,\underline{a}\,b\,\#)$

$\vdash M\,(q_1, \#\,X\,X\,X\,X\,X\,\underline{b}\,\#)$

$\vdash M\,(q_3, \#\,X\,X\,X\,X\,\underline{X}\,X\,\#)$

$\vdash M\,(q_3, \#\,X\,X\,X\,\underline{X}\,X\,X\,\#)$

$\vdash^{*} M\,(q_3, \underline{\#}\,X\,X\,X\,X\,X\,X\,\#)$

$\vdash M\,(q_0, \#\,\underline{X}\,X\,X\,X\,X\,X\,\#)$

$\vdash^{*} M\,(q_0, \#\,X\,X\,X\,X\,X\,X\,\underline{\#})$

$\vdash M\,(q_4, \#\,X\,X\,X\,X\,X\,\underline{X}\,\#)$ Accepted

Since, q_4 is final state, so string 'babaab' is accepted.

Extensions / variations of Turing machine

Now, it becomes clear that Turing machine can perform fairly powerful computation in order to better understand their surprising extending the Turing machine in various directions. The "new", improved models "of the Turing machine can in each instance be simulated by the standard model. Such results increase our confidence that the Turing machine is indeed the ultimate computational device, the end of our progression to move to powerful automata. The various extension of Turing machine are explained in detail as follows:

Extension / variations of Turing machine (conted)

Multiple tapes Turing machine

One can think of Turing machine that have several tapes (see fig. (i)). Each tape is connected to the finite control by means of a read/write head (one on each tape).

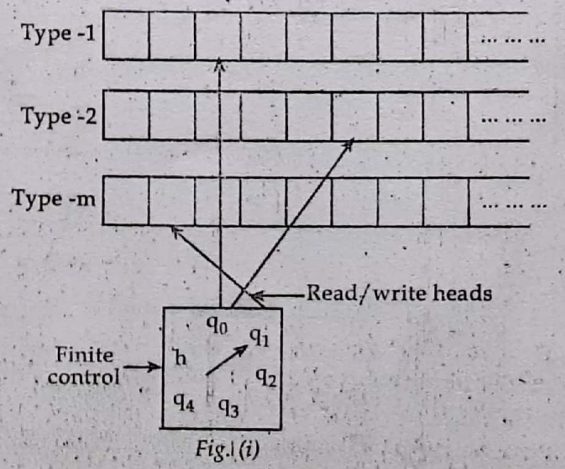


Fig.1 (i)

The machine can in one step read the symbols scanned by all its heads and then, depending on these symbols in current state, rewrite some of those scanned squares and move some of the heads to the left or right, in addition to changing the state.

For any fixed integer $m \geq 1$, an m-tape Turing machine is a Turing machine equipped with m-tapes and corresponding heads. Thus, standard Turing machine studied so far in this chapter is just an m-tape Turing machine, with $m = 1$.

$\Rightarrow$ Let $m \geq 1$ be an integer

An m-tape Turing machine is a six tuple machine as follows:

$Tm = (Q, \Sigma, \Gamma, \delta, q_0, h)$

Where, Q = The finite set of states of the finite control

Σ = The finite set of input symbols

Γ = The tape symbol where $\Gamma = \Sigma \cup \#$

h = Halt state $h \in Q$

q_0 = The initial state, $q_0 \in Q$

δ = The transition function is a function which maps

$(Q \times \Sigma^m) \rightarrow Q \times (\Sigma \cup \{L, R, N\}^m)$

That is, for each state q and each m-tapes of tape symbols $(a_1, a_2, ..., a_k)$, δ $(a_1, a_2, ..., a_k) = (P, (b_1, b_2, ..., b_k)$ where P is, as before, the new state, and b_j is the action taken by Tm at tape j.

Computation takes place in all m-tapes of a m-taps Turing machine. Accordingly, a configuration of such a machine must include information about all tapes.

Example 1: Design an m-tape Turing machine which work as copying machine.

Solution

Now, we are free to use m-tape in the designing of Turing machine.

We will design a 2-tape Turing machine, which takes input # w # and gives output as #w#w#.

Let us clear notations for the tapes first.

For the first tape machines notations used will be written as R' and L' input symbol from tape-1 is read-out as σ'. Similarly for the tape-2 notations of machines will R^2 and L^2 and so on. Input which is read-out from tape-2 will be shown as σ^2.

This Turing machine can be accomplished as follows:

(a) Move the leads on both tapes to the right, copying each symbol on the first tape on to the second tape, until a blank is found on the first tape. The first square of the second tape should be left blank.

(b) Move the lead on second tape to the left until a blank is found.

(c) Again move the heads on both tapes to the right this time copying symbols from the second tape on to the first tape. Halt when a blank is found on the second tape.

The required 2-tape Turing machine is shown in figure as follows:

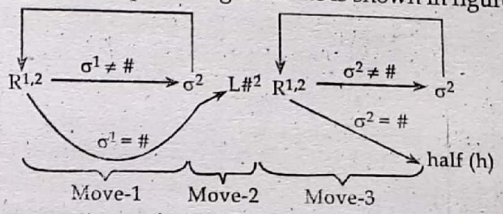
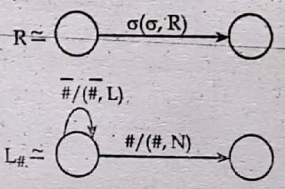


Fig. (ii)

Here,



and

The sequence of actions can be pictured as follows:

First tape # w #

Second tape # #

After (move –1): First tape # w #

Second tape # w #

After (move – 2): First tape # w #

Second tape # w #

After (move – 3): First tape # w # w #

Second tape # w #

Two-way infinite tape

The extensions are distinguished from each other and from the standard Tm through different definitions of next-move relation δ and of configurations for each of the extension. So here we discuss the extension only in terms of definitions of δ and of configuration.

Like standard Tm, in this case also, the next-move is given by δ as a partial function from

$$(Q \times \Gamma) \text{ to } (Q \times \Gamma \times \{L, R, N\})$$

To following three points need to be noted in respect of configurations of two-way Turing machine.

(a) Configuration/instantaneous description. In standard Tm, if there are a number or left-most positions which contain blanks, then those are included in the configuration that is if the one-way configuration tape is of the form

```
# ab # cd ef ## ... #
  ↑
  q2
```

then the configuration in the standard Tm is written as:

$(q_2, \#\ ab\ \#\ cdef\ \#\#\ ...\ \#)$

However, in the two-way infinite tape Tm, both left-hand and right-hand parts of the tape are symmetrical in the sense that there is an infinite continuous sequence of blanks on each of the right hand and left hand of the sequence of non-blanks. Therefore, in case of two-way infinite tape, if the above string is on the tape then it will be in the form.

$(q_2, \#\ ...\ \#\#\ ab\ \#\ cdef\ \#\#\ ...\ \#)$

(b) No (easing of operation without halting. In this case, as there is no left end of the tape, therefore, there is no possibility of jumping off the left-end of the tape.

(c) The empty tape configuration. When at some point of time all the cells of the tape are #'s and the state is say q, then configuration in two-way tape may be dented as:

$(q, \#)$

Where only the current cell containing # is shown in figuration

TIPS

Despite the fact that, it is possible in the new model of computer to move left as far as required. The model does not provide any additional computational capability.

Multiple heads Turing machine

In order to simplify the discussion, we assume that there are only two heads on the tape. The tape is assumed to be one-way infinite. We explain

the involved concepts with the help of an example. Let the content of the tape and the position of the two heads that is H_1 and H_2, be as given below:

```
## ab c # d e f ## ...
   ↑     ↑
   H2    H1
```

Further, let the state of the Tm be q.

The move function of the two-head one-way Turing may be defined as

δ (state, symbol under head 1, symbol under head 2)

= (new state, (S_1, M_1), (S_2, M_2)

where S_i is the symbol to be written in the cell under H, (the i^{th} head) and M_i denotes the movement the movement of H_i where the movement may be L, R or N and further L denotes movement to the left, R denotes movement to the right of the current cell and N denotes 'no movement of the head'.

Two special cases of the S function defined above, need to be considered.

(a) What should be written in the current cell when both head are scanning the same cell at a particular time and the next moves (S_1, M_1), (S_2, M_2) for the two heads are such that $S_1 \neq S_2$ that is symbol to be written is current cell by H_1 is not same as symbol be written in current cell by H_2.

In such a situation, a general rule may be defined, say as whatever is to be done by H_1 will take precedence over whatever is to be done by H_2.

(b) For two-head one way tape, a configuration shall be called hanging if δ(q, symbol under H_1, symbol under H_2) = $(q_1, (S_1, M_1), (S_2, M_2)$ is such that either.

(i) symbol under H_1 is in the left-most cell and M_1 is L_1 that is movement of H_1 is to be to the left,

(ii) Symbol under H_2 is in the left-most cell and M_2 is L_1 that is movement of H_2 is to be to the left.

TIPS

The above discussion can be further extended easily to the case when number of heads is move the two. the power of Tm is not enhanced by the use of extra heads.

K-dimensional Turing machine

Again to facilitate the understanding of the basic ideas involved, let us discuss initially only two-dimensional Turning machine. Then these ideas can be easily generated to k-dimensional case, where $k > 2$.

In the case of two-dimensional tape as shown below, we assume that the tape is bounded on the left and the bottom.

Each cell, is given an address say (i, j) where i is the row-number of the cell and j is the column number of the cell.

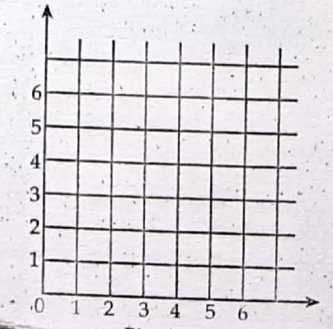


Fig. (iii)

A configuration of a two-dimensional Tm at a particular time may be described in terms of finitely many of the triplets of the form (i_1, i_2, c) where for each such triplets of the form, (i_1, i_2, C) where for each such triplet (i_1, i_2) in the address of a cell and c denotes the contents of the cell. Only these cells are included in on ID, for which, the contents are non-blank symbols.

In the configuration or ID, order of the cell which are included in on ID. Row Major Ordering is to be followed that is first all the elements in the row with least index are included in the ID, followed by the elements of the row with next least index and so on. Within cells of each row, the cell with non # contents and having least column number is included first followed by the non blank cell with next least column number and so on.

Let $q \in Q$, $C_k \in \Gamma - \{\#\}$, that is C_k is a non-blank tape symbol.

The configuration at a particular instant is denoted by $(q_1 (H_1, H_2 (i_1, j_1), (i_2, j_2, C_{i_2} j_2) \ldots (i_k, j_k, C_{i_k} k_k \ldots))$ where each of $C_{i_1} j_1, C_{i_2} j_2 \ldots$ is non-blank and these are the only non-blank on the tape.

Also (H_1, H_2) denotes the location of the cell currently being seconded that is the cell under the head.

Non-deterministic Turing machine

We have already seen that when finite automata are allowed to act non0deterministically no increase in the computational power, but that non-determine in push down automata are more powerful that deterministic ones.

We can also imagine Turing machine that acts non-deterministically. In standard Tm, to each point of current state (except the half-0state) and the symbol being scanned, there is a unique triplet comprising of the next state, unique action in terms of writing a symbol in the cell being scanned and the motion, if any to the right or left. However, in case of NDTM (non deterministic Turing machine), to each pair (q, a) with q as current state and 'a' as symbol being scanned, there may be a finite set of triplets $\{q_i, a_i, m_i); i = 1, 2, \ldots\}$ of possible moves. The set of triplets may be empty that is for some particular (q, a) the Tm may not have any next move. Or alternatively the set $\{(q_i, a_i, m_i)\}$ may have more than one triplet, meaning thereby that the NDTM in the state q and scanning symbol 'a' and can choose next move from the set $\{(q_i, a_i, m_i)\}$ of next moves.

From the above discussion it can be easily understood that standard Tm is a special case of the NDTM in which for each (q, a) the set $\{(q_i, a_i, m_i)\}$ in next moves is a singleton set or empty.

In order to define the concept of NDTM and a configuration in NDTM, we assume that the tape is one-way infinite.

TIPS

Proper non-determinism means that at some stage, there are at least two next possible moves. Now, if we engage two different machines to work out further possible moves according to each of these two moves the two can work independent of each other. This means non-determinism allows parallel computation.

A non-0deterministic Turing machine is a six tuple $(Q, \Sigma, \Gamma, \delta, q_0, h)$ where

Q : set of states

Σ : set of input smbols

Γ : set of tape symbols

$\delta : (Q \times \Gamma) \rightarrow$ power set of $(Q \times \Gamma \times \{L, R, N\})$

q_0: initial state and

h : halt state, $h \in Q$

The concept of a configuration is same as in case of standard Tm. But the concept of 'yield in one step' denoted by Tm has different meaning.

Here one configuration may yield more that the language one configuration.

Example: Construct an NDTM which accepts $\{a^n b^m : n \geq 1, m \geq 1\}$, that is language of all strings over {a, b}, in which there is at least one 'a' and one 'b' and all a's precede all b's.

Solution:

The diagrammatic representation of the required NDTM is as given below: (initially tape head is at leftmost symbol that is 'a')

In the proposed NDTM, as the motion of the head is always to the right except in the halt sate, therefore, R is not mentioned in the labels in the given diagram.

Where the label x/y on denotes that if symbol in the current cell is 'x' then contents of the cell are to be replaced by 'y'. Formally the proposed NDTM may be defined as

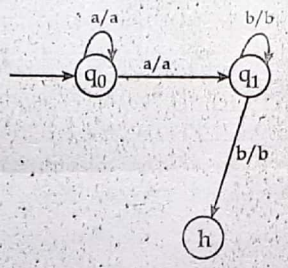


Fig. (iv)

$$M = \{\{q_0, q_1\}, \{a, b\}, \{a, b, \#\}, \delta, q_0, h\}$$

where δ is defined as follows:

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

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

$\delta(q_1, a) = \text{empty}$

$\delta(q_1, b) = \{(a, b, R), (h_1, b, N)\}$

Unrestricted grammars

Before understating unrestricted grammars, let's first understand phrase structure grammars:

A phrase structure grammar is a collection of three things.

(a) A finite alphabet Σ of letters called terminals.

(b) A finite set of symbols called non-terminals that includes the start symbol s.

(c) A finite set of production of the form.

String 1 ⟶ string 2

Where string 1 can be any string of terminals and non-terminals that contains at least one non-terminals and where string 2 is any string of terminals and non-terminals what so ever. The language generated by a phrase structure grammar is the set of all string of terminals that can be derived starting at s.

Till now, we have discussed several types of languages and the relationship between them are as follows:

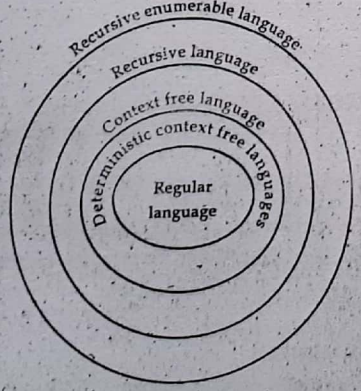


Fig. Chomsky hierarchy

S.No.	Name of language generated	Productions A → B	Acceptor
1	Regular	A = one non-terminal B = aX or B = a, where a is a terminal and X is a non-terminal	Finite automata
2.	Context free	A = one non-terminal B = any string	Pushdown automata
3.	Unrestricted (recursively enumerable)	A = any string with non terminal B = any string	Turing machine

Unrestricted grammars accepts recursively enumerable language. the unrestricted grammar is defined as

$$G = (V_n V_t, P, S)$$

where V_n = a finite set of non-terminals

V_t = a finite set of terminals

S = starting non-terminal, $S \in V_n$

and P is set of productions of the following form

$$\alpha \rightarrow \beta$$

where α and β are arbitrary string of grammar symbols with $\alpha \neq \epsilon$. These grammars are also known as phase structure or unrestricted grammars.

Example – 1: Construct a grammar (unrestricted) for language $L = \{a^n b^n c^n / n > 0\}$

Solution:

We know that if is difficult to write grammar for $L = \{a^n b^n c^n / n > 0\}$ as it is neither regular nor context-free. So we try to write grammar in two parts

(a) First we construct grammar for $a^n x^n$.

(b) Then convert x^n into $b^n c^n$.

Let grammar be $G = (V^n, V_t, P, S)$

where, $V_n = \{S, B, C\}$

$V_t = \{a, b, c\}$

$S = \{s\}$

and P is defined as follows:

S → a SBC / a BC

CB → BC

aB → ab

bB → bb

bC → bc

cC → cc

Now let us derive $a^n b^n c^n$ from above discussed grammar.

S $\Rightarrow a^{n-1} S (BC)^{n-1}$ (use S → a SBC (n – 1) times)

$\Rightarrow a^{n-1} a BC (BC)^{n-1}$ (use s → a BC)

$\Rightarrow a^n (BC)^n$

$\Rightarrow a^n B^n C^n$ (By applying CB → BC several times)

$\Rightarrow a^{n-1} ab B^{n-1} C^n$ (By applying aB → ab once)

$\Rightarrow a^n b^n C^n$ (By applying bB → bb several times)

$\Rightarrow a^n b^{n-1} bc C^{n-1}$ (By applying bC → bc once)

$\Rightarrow a^n b^n c^n$ (By applying cC → cc several times)

For example; lets derive s = aaabbbccc

S $\Rightarrow a \underline{S} BC$

$\Rightarrow a a \underline{S} B C B C$

$\Rightarrow a a a B \underline{CB} C B C$

$\Rightarrow a a a B B C \underline{CB} C$

$\Rightarrow a a a B B \underline{CB} C C$

$\Rightarrow a a \underline{aB} B B C C C$

$\Rightarrow a a a \underline{bB} B C C C$

$\Rightarrow a a a b \underline{bB} C C C$

$\Rightarrow a a a b b \underline{bC} C C$

$\Rightarrow a a a b b b \underline{cC} C$

$\Rightarrow a a a b b b c \underline{cC}$

$\Rightarrow a a a b b b c c c$

Example 2: Write a grammar generating {a^i / i is a positive power of 2}.

Solution:

Let grammar be G = (V_n, V_t, P, S)

where, V_n {S, A, C, B, D, E}

V_t = {a, ∈}

S = {s}

and P is defined as follows:

S → A C a B (P_1)

Ca → aaC (P_2)

CB → DB (P_3)

CB → E (P_4)

aD → Da (P_5)

AD → AC (P_6)

aE → Ea (P_7)

AE → ∈ (P_8)

A and B serve as left and right and marks for sentential forms, c is a marker that moves through the string of a's between A and B, doubling their number by production P_2 when C hits the right and marker B, it becomes a D or E by production P_3 and P_4. If a D is chosen, that D migrates left by production P_5 until the left end marker A is reached. At the point, the D becomes a again by production P_6 and the process starts over. It an E is chosen the right and marker is consumed. The E migrates left by production P_7 and consumer the left end marker, leaving a string of 2^i a's for some i > 0.

Let us derive string S = aa for a^i where i = 1.

S $\Rightarrow A \underline{Ca} B$

$\Rightarrow A a a \underline{CB}$ (P_2)

$\Rightarrow A a \underline{a E}$ (P_4)

$\Rightarrow A \underline{a E} a$ (P_7)

$\Rightarrow A E a a$ (P_7)

$\Rightarrow a a$ (P_8)

Example 3: Let G = {V_n, V_t, P S) where

V_n = {S, X, Y}

V_t = {a, b}

S = {S}

and P is defined as follows:

$S \rightarrow a\,XYa$

$X \rightarrow ba\,XYb$

$Y \rightarrow X\,ab$

$aX \rightarrow baa$

$bYb \rightarrow abab$

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

Solution:

Here I have underlined the substring to be replaced by the use of a production.

$S \Rightarrow \underline{aX}\,Ya$

$\Rightarrow baa\,\underline{Y}a$

$\Rightarrow baa\,\underline{X}aba$

$\Rightarrow baab\underline{aX}Ybaba$

$\Rightarrow baabbaa\,\underline{Y}baba$

$\Rightarrow baabba\underline{aX}abbaba$

$\Rightarrow baabbabaaabbaba$.

$\Rightarrow w$

So, $w \in L(G)$.

Recursive function theory

In this unit, we will be concerned with recursive function theory, which is a functional or declarative approach to computation under this approach, computation is described in terms of 'what is to be accomplished' instead of 'how to accomplish'.

In the previous units, we have discussed the automata or machine models of the computation phenomenon. The automata approach to computation is operational is nature i.e. automata approach is concerned with the computational aspect of 'how the computation is to be performed.'

In automata theory, the concepts like 'state', 'initial state', 'final state' and 'input', etc are assumed to be understood, without any elaboration. Further, the capabilities of an automata to accept an input from the environment; to change its state on some, or even on no input i to give signal about acceptability unacceptability of a string i are assumed.

In recursive function theory, to begin with, it is assumed that three types of functions and three types of functions and three structuring rules (that is combination, composition and primitive recursion) for constructing more complex functions out of the already constructed or assumed to be constructible functions are so simple that our ability to construct machines to realize these functions and the structuring rules is taken as acceptable without an argument.

Recursion

We know that recursion is widely used term in computer science. "A fucntion which calls itself directly or indirectly and terminates after finite number of steps is known as recursive function." In recursive functions, terminating point is also known as base point.

Initial functions for natural numbers

It is clear that all elementary functions are all functions of natural numbers hence, they may take zero as input, but not a negative number and not any rational or irrational fraction.

Let N = {0, 1, 2, ...} be a set of natural numbers. We have three initial functions over N defined below.

(a) Zero function: It is denoted by Z and defined as follows:

$Z(x) = 0$ for $\forall$ (every) $n \in N$

Clearly the zero function returns zero regardless of its argument.

(b) Successor function (s) is defined as follows:

$S(n) = n + 1$ for $\forall\, n \in N$

Clearly successor function takes one argument and returns the succeeding number. For example,

$S(2) = 2 + 1 = 3$ where, $2, 3 \in N$

$Z(0) = 0$

$Z(2) = 0$

(c) Projection function (P_1^n) is defined as follows: $P_i^n (q_1, q_2, \ldots q_n) = q_i$, where $q_i \in N$ for $l = 1, 2, \ldots, n$ and $i \leq n$. Here projection function takes n arguments and returns their i^{th} argument. When it takes one argument it returns its argument as its value.

Initial functions for symbols

We know that set of alphabets is represented by Σ. Two initial functions are defined over Σ

(a) **Null function:** It is defined as follows:

NULL (n) = ϵ for $\forall$ n $\in \Sigma^*$

For example, If Σ = {a, b}

Null (a) = ϵ

Null (b) = ϵ

(b) **Concatenation function:** It is defined as follows:

CONCATE W(W_1) = WW_1 for W, $W_1 \in \Sigma^*$ and $|W_1| \geq 0$

For example, CONCATE K(ITE) = KITE

CONCATE a (CONCATE b(NULL (ABC)))

= CONCATE a (CONCATE b(ϵ))

= CONCATE a (b)

= ab

Composition of functions

We can define a new function by the combination of two or more functions. Such defined functions are called composition function.

For example, S(Z(a)) = S(0) = 1

and S(S(Z(n))) = 2 and so on

Here, S is successor function and Z is null function.

Types of recursive function/partial recursive function

A function is called partial recursive if it is defined for some of its arguments. Let $f(a_1, a_2, \ldots a_n)$ be a function and defined on function g $(b_1, b_2, \ldots b_n)$ then f is partial function if some elements of f is assigned to almost one element of function g.

A partial function is recursive if

(a) It is an initial function over N, or

(b) It is obtained by applying recursion or composition on initial function over N.

Total recursive function

A total function is a subclass of partial function A function is called total function if it is defined for all its arguments. Let f $(a_1, a_2, \ldots, a_n)$ be a function and defined on function g $(b_1, b_2, \ldots, b_m)$ then f is total function if every element of f is assigned to some unique element of function g.

Example – 1: Addition of two positive integer is a total function.

Solution:

Let f(m, n) = m + n be the addition function from N × N to N of two positive integers m and n. This function can be defined recursively as f(m, 0) = 1, f(m, n + 1) = f (m, n) + 1.

Since, f is defined for all values of m, n $\in$ N. So f is a total function.

Example – 2: Multiplication of two positive integers is a total function.

Solution:

Let g(m, n) be multiplication function from N × N to N of two positive integers m and n, defined recursively as g(m, 0) = 0, g(m, n + 1) = m + g(m, n). Since g is defined for all m, n $\in$ N. So g is a total recursive function.

Primitive recursive function

A function is primitive if it follows the condition.

(a) It is an initial function, or

(b) It is obtained from recursion or composition of initial functions.

Most popular total functions are primitive recursive functions, however some total functions are not. For example the Ackermann function is total function but not primitive recursive.

Example – 1: Show that addition of two positive, integers is primitive recursive.

Solution:

Let f is the addition function of two positive integers, so

f(m, n) = m + n

We define of to increment the argument m, n times one at a time

f(m, 0) = m

f(m, n + 1) = f(m, n) + 1

Now, we define f using the initial functions as follows:

For example, let us consider f(7, 2)

f(7, 2) = s(f(7, 1)) (successor function)

= f(7, 1) + 1

= s(f(7, 0)) + 1

= f(7, 0) + 1 + 1

= 7 + 1 + 1 = 9

Hence, f is a primitive recursive function.

Exam Solution

1. Design a Turing machine that increments any binary strings by one with Σ = {0, 1, #}. Hence test your design for ## 11 # to # 100 #.

[2075 Ashwin, Back]

Here, while we feed input on input tape, we add extra blank symbol '#' on the leftmost side of the string. For example, input # 11 # becomes ## 11 # for overflow (carry)

Let the required Turing machine be

$$T(M) = (\theta, \Sigma, \Gamma, b, \delta, q_0 F).$$

where

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

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

$\Gamma = \{0, 1, \#\}$

$b = \{\#\}$

$q_0 = \{q_0\}$

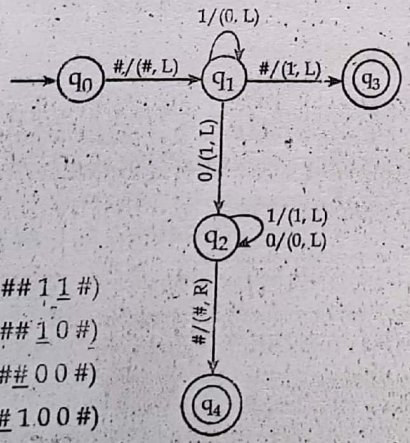
$F = \{q_3, q_4\}$

and transition δ is defined as follows:

State/Input	0	1	#
q_0	–	–	$(q_1, \#, L)$
q_1	$(q_2, 1, L)$	$(q_1, 0, L)$	$(q_3, 1, L)$
q_2	$(q_2, 0, L)$	$(q_2, 1, L)$	$(q_4, \#, R)$
q_3	–	–	–
q_4	–	–	–

Note: '–' is for undefined move

States diagram



Let us process string ## 11 #

$(q_0, \#\# 11 \#) \vdash M (q_1, \#\# 1 1 \#)$

$\vdash M (q_1, \#\# 1 0 \#)$

$\vdash M (q_1, \#\# 0 0 \#)$

$\vdash M (q_3, \# 1 0 0 \#)$

2. Design a multi-tape Turing machine which act as copying machine over the alphabets Σ = {0, 1} that transforms string of the form "# 10#' into # 10# 10#. **[2074 Ashwin, Back]**

Let us clear notations for m multi tape Turing machine where m = 2 i.e. 2 tapes Turing machine.

For the first tape machine notations used will be written as R' and L'. Input symbol from tape – 1 is read out as σ^1. Similarly for the tape – 2 notations of machines will be R^2 and L^2 and so on. Input which is read out from tape – 2 will be shown as σ^2.

This Turing machine can be accomplished as follows:

(a) Move the heads on both tapes to the right, copying each symbol on the first tape to the second tape, until a blank is found on the first tape. The first square of the second tape should be left blank.

(b) Move the head or the second tape to the left until a blank is found.

(c) Again move the heads on both tapes to the right, this time copying symbols from the second tape on to the first tape. Halt when a blank is found on the second tape.

The required 2-tape Turing machine is shown below:

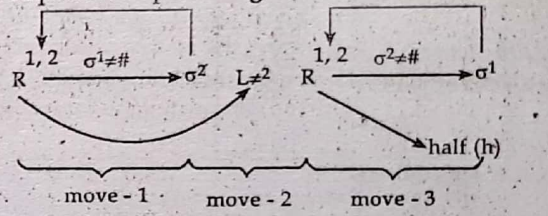
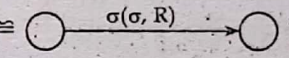
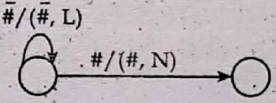


Fig. 2 tape Turing machine

Here, R ≅ 

L ≃ 

and the square sequence of actions can be pictured as follows:

Move – 1 →	Firs tape	# 10 #
	Second tape	# #
Move – 2 →	Firs tape	# 10 #
	Second tape	# 10 #
Move – 3 →	Firs tape	# 10 #
	Second tape	# 10 #
Move – 4 →	Firs tape	# 10 # 10 #
	Second tape	# 10 #

This is the required multi tape Truing machine.

3. **Design a single tape deterministic Turing machine which accepts all strings defined for the language $L = \{a^n cb^n, n \geq 0\}$ over the alphabet $\Sigma = \{a, b, c\}$.** **[2074 Ashwing, Back]**

Ans Let, the required Turing machine be

$$T(M) = (\theta, \Sigma, \Gamma, b', \delta, q_0, F)$$

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{a, b, c, X, Y, \#\}$

$b' = \{\#\}$

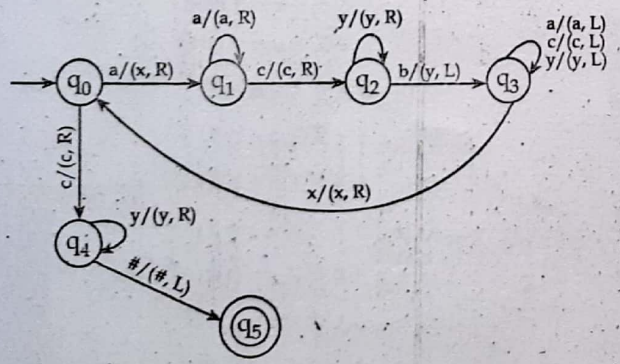
$q_0 = \{q_0\}$

$F = \{q_5\}$

and translation (δ) is defined as follows:

State/Input	a	b	c	X	Y	#
q_0	(q_2, X, R)	–	(q_4, c, R)	–	–	–
q_1	(q_1, a, R)	–	(q_2, c, R)	–	–	–
q_2	–	$(q_3, 4, L)$	–	–	(q_2, Y, R)	–
q_3	(q_3, a, L)	–	(q_3, c, L)	(q_0, X, R)	(q_3, Y, L)	–
q_4	–	–	–	–	(q_4, Y, R)	$(q_5, \#, L)$
q_5	–	–	–	–	–	–

States diagram



Note: '–' is for undefined move i.e. Turing machine to halt. And this Turing machine is deterministic one as there is only one transition state for every input on every state i.e. either to halt or to process more but not both at the same time.

Let us process string # aaa c bbb #

$(q_0, \# \underline{a}\, a\, a\, c\, b\, b\, b\, \#)$ $\vdash_M (q_1, \# X \underline{a}\, a\, c\, b\, b\, b\, \#)$

$\vdash_M (q_1, \# X\, a\, \underline{a}\, c\, b\, b\, b\, \#)$

$\vdash_M (q_1, \# X\, a\, a\, \underline{c}\, b\, b\, b\, \#)$

$\vdash_M (q_2, \# X\, a\, a\, c\, \underline{b}\, b\, b\, \#)$

$\vdash_M (q_3, \# X\, a\, a\, \underline{c}\, Y\, b\, b\, \#)$

$\vdash_M (q_3, \# X\, a\, \underline{a}\, c\, Y\, b\, b\, \#)$

$\vdash_M (q_3, \# X\, \underline{a}\, a\, c\, Y\, b\, b\, \#)$

$\vdash_M (q_3, \# \underline{X}\, a\, a\, c\, Y\, b\, b\, \#)$

$\vdash_M (q_0, \# X \underline{a}\, a\, c\, Y\, b\, b\, \#)$

$\vdash_M (q_1, \# X X \underline{a}\, c\, Y\, b\, b\, \#)$

$\vdash_M (q_1, \# X X\, a\, \underline{c}\, Y\, b\, b\, \#)$

$\vdash_M (q_2, \# X X\, a\, c\, \underline{Y}\, b\, b\, \#)$

$\vdash_M (q_2, \# X X\, a\, c\, Y\, \underline{b}\, b\, \#)$

$\vdash_M (q_3, \# X X\, a\, c\, \underline{Y}\, Y\, b\, \#)$

$\vdash_M (q_3, \# X X\, a\, \underline{c}\, Y\, Y\, b\, \#)$

$\vdash_M (q_3, \# X X\, \underline{a}\, c\, Y\, Y\, b\, \#)$

$\vdash_M (q_0, \# X X \underline{a}\, c\, Y\, Y\, b\, \#)$

$\vdash_M (q_1, \# X X X \underline{c}\, Y\, Y\, b\, \#)$

$\vdash_M (q_2, \# X X X\, c\, \underline{Y}\, Y\, b\, \#)$

$\vdash_M (q_2, \# X X X\, c\, Y\, \underline{Y}\, b\, \#)$

$\vdash_M (q_2, \# X X X\, c\, Y\, Y\, \underline{b}\, \#)$

$\vdash_M (q_3, \# X X X\, c\, Y\, \underline{Y}\, Y\, \#)$

$\vdash_M (q_3, \# X X X\, c\, \underline{Y}\, Y\, Y\, \#)$

$\vdash_M (q_3, \# X X X\, \underline{c}\, Y\, Y\, Y\, \#)$

$\vdash_M (q_3, \# X X \underline{X}\, c\, Y\, Y\, Y\, \#)$

$\vdash_M (q_0, \# X X X \underline{c}\, Y\, Y\, Y\, \#)$

$\vdash_M (q_4, \# X X X\, c\, \underline{Y}\, Y\, Y\, \#)$

$\vdash_M (q_4, \# X X X\, c\, Y\, \underline{Y}\, Y\, \#)$

$\vdash_M (q_4, \# X X X\, c\, Y\, Y\, \underline{Y}\, \#)$

$\vdash_M (q_4, \# X X X\, c\, Y\, Y\, Y\, \underline{\#})$

$\vdash_M (q_5, \# X X X\, c\, Y\, Y\, \underline{Y}\, \#)$ Accepted

Since, q_5 is accepting state, so string 'aaacbbb' is accepted.

4. Define Turing machine. Design a single tape deterministic Turing machine which reverses the given string w, over alphabet $\Sigma = \{a, b\}$.

[2074 Chaitra]

For definition of Turing Machine, please see theory part.

Let, the required Turing machine be

$$T(M) = (\theta, \Sigma, \Gamma, b', \delta, q_0, F)$$

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

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

$\Gamma = \{a, b, X, \#\}$

$b' = \{\#\}$

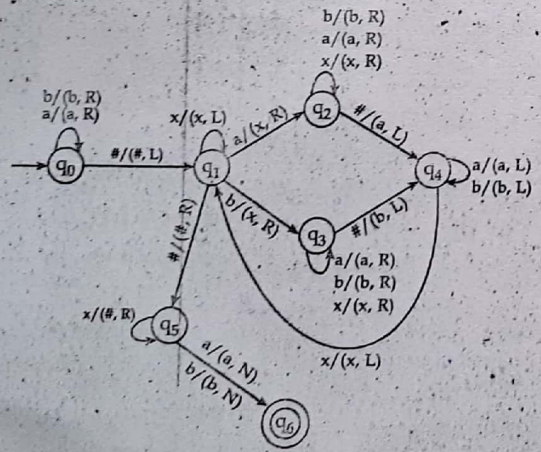
$q_0 = \{q_0\}$

$F = \{q_6\}$

and transition δ is defined as follows:

State/Input	a	b	X	#
q_0	(q_0, a, R)	(q_0, b, R)	–	$(q_1, \#, L)$
q_1	(q_2, X, R)	(q_3, X, R)	(q_1, X, L)	$(q_5, \#, R)$
q_2	(q_2, a, R)	(q_2, b, R)	(q_2, X, R)	(q_4, a, L)
q_3	(q_3, a, R)	(q_3, b, R)	(q_3, X, R)	(q_4, b, L)
q_4	(q_4, a, L)	(q_4, b, L)	(q_1, X, L)	–
q_5	(q_6, a, N)	(q_6, b, N)	$(q_5, \#, R)$	–
q_6	–	–	–	–

States diagram



Let us process string # b aa # ###

$(q_0, \#\ \underline{b}aa\ \#\#\#)$ $\vdash\!\!\!-$ M $(q_0, \#\ b\ \underline{a}\ a\ \#\ \#\ \#)$

$\vdash\!\!\!-$ M $(q_0, \#\ b\ a\ \underline{a}\ \#\ \#\ \#)$

$\vdash\!\!\!-$ M $(q_0, \#\ b\ a\ a\ \underline{\#}\ \#\ \#)$

$\vdash\!\!\!-$ M $(q_1, \#\ b\ a\ \underline{a}\ \#\ \#\ \#)$

$\vdash\!\!\!-$ M $(q_2, \#\ b\ a\ X\ \underline{\#}\ \#\ \#)$

$\vdash\!\!\!-$ M $(q_4, \#\ b\ a\ \underline{X}\ a\ \#\ \#)$

$\vdash\!\!\!-$ M $(q_1, \#\ b\ \underline{a}\ X\ a\ \#\ \#)$

$\vdash\!\!\!-$ M $(q_2, \#\ b\ X\ \underline{X}\ a\ \#\ \#)$

$\vdash\!\!\!-$ M $(q_2, \#\ b\ X\ X\ \underline{a}\ \#\ \#)$

$\vdash\!\!\!-$ M $(q_2, \#\ b\ X\ X\ a\ \underline{\#}\ \#)$

$\vdash\!\!\!-$ M $(q_4, \#\ b\ X\ X\ \underline{a}\ a\ \#)$

$\vdash\!\!\!-$ M $(q_4, \#\ b\ X\ \underline{X}\ a\ a\ \#)$

$\vdash\!\!\!-$ M $(q_1, \#\ b\ \underline{X}\ X\ a\ a\ \#)$

$\vdash\!\!\!-$ M $(q_1, \#\ \underline{b}\ X\ X\ a\ a\ \#)$

$\vdash\!\!\!-$ M $(q_3, \#\ X\ \underline{X}\ X\ a\ a\ \#)$

$\vdash\!\!\!-$ M $(q_3, \#\ X\ X\ \underline{X}\ a\ a\ \#)$

$\vdash\!\!\!-$ M $(q_3, \#\ X\ X\ X\ \underline{a}\ a\ \#)$

$\vdash\!\!\!-$ M $(q_3, \#\ X\ X\ X\ a\ \underline{a}\ \#)$

$\vdash\!\!\!-$ M $(q_4, \#\ X\ X\ X\ a\ \underline{\#}\ \#)$

$\vdash\!\!\!-$ M $(q_4, \#\ X\ X\ X\ a\ \underline{a}\ b\ \#)$

$\vdash\!\!\!-$ M $(q_4, \#\ X\ X\ \underline{X}\ a\ a\ b\ \#)$

$\vdash\!\!\!-$ M $(q_4, \#\ X\ \underline{X}\ X\ a\ a\ b\ \#)$

$\vdash\!\!\!-$ M $(q_1, \#\ X\ \underline{X}\ X\ a\ a\ b\ \#)$

$\vdash\!\!\!-$ M $(q_1, \#\ \underline{X}\ X\ X\ a\ a\ b\ \#)$

$\vdash\!\!\!-$ M $(q_1, \underline{\#}\ X\ X\ X\ a\ a\ b\ \#)$

$\vdash\!\!\!-$ M $(q_5, \#\ \underline{X}\ X\ X\ a\ a\ b\ \#)$

$\vdash\!\!\!-$ M $(q_5, \#\ \#\ \underline{X}\ X\ a\ a\ b\ \#)$

$\vdash\!\!\!-$ M $(q_5, \#\ \#\ \#\ \underline{X}\ a\ a\ b\ \#)$

$\vdash\!\!\!-$ M $(q_5, \#\ \#\ \#\ \#\ \underline{a}\ a\ b\ \#)$

$\vdash\!\!\!-$ M $(q_6, \#\ \#\ \#\ \#\ \underline{a}\ a\ b\ \#)$

5. Design a Turing machine to accept language $L = \{ww^R \mid w \in \{0, 1\}^*\}$. Show processing for the string 101101. [2073 Chaitra]

Let, the required Turing machine be

$$T(M) = (\theta, \Sigma, \Gamma, b', \delta, q_0, F)$$

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

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

$\Gamma = \{0, 1, \#\}$

$b' = \{\#\}$

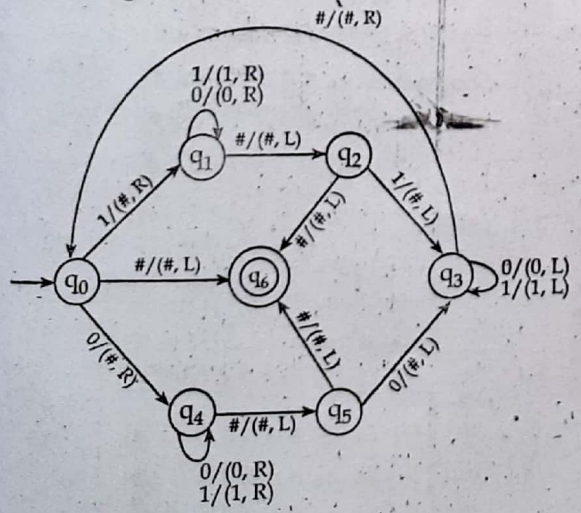
$q_0 = \{q_0\}$

$F = \{q_6\}$

and transition δ is defined as follows:

State/Input	0	1	#
q_0	$(q_4, \#, R)$	$(q_1, \#, R)$	$(q_6, \#, L)$
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	$(q_2, \#, L)$
q_2	–	$(q_3, \#, L)$	$(q_6, \#, L)$
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_0, \#, R)$
q_4	$(q_4, 0, R)$	$(q_4, 1, R)$	$(q_5, \#, L)$
q_5	$(q_3, \#, L)$	–	$(q_6, \#, L)$
q_6	–	–	–

States diagram



Note: Same as that of palindrome.

Let us process string # 101101#

$(q_0, \# 101101\#) \vdash M (q_1, \# \# \underline{0} 1 1 0 1 \#)$

$\vdash M (q_1, \# \# 0 \underline{1} 1 0 1 \#)$

$\vdash M (q_1, \# \# 0 1 \underline{1} 0 1 \#)$

$\vdash M (q_1, \# \# 0 1 1 \underline{0} 1 \#)$

$\vdash M (q_1, \# \# 0 1 1 0 \underline{1} \#)$

$\vdash M (q_1, \# \# 0 1 1 0 1 \underline{\#})$

$\vdash M (q_2, \# \# 0 1 1 0 \underline{1} \#)$

$\vdash M (q_3, \# \# 0 1 1 \underline{0} \# \#)$

$\vdash M (q_3, \# \# 0 1 \underline{1} 0 \# \#)$

$\vdash M (q_3, \# \# 0 \underline{1} 1 0 \# \#)$

$\vdash M (q_3, \# \# \underline{0} 1 1 0 \# \#)$

$\vdash M (q_3, \# \underline{\#} 0 1 1 0 \# \#)$

$\vdash M (q_0, \# \# \underline{0} 1 1 0 \# \#)$

$\vdash M (q_4, \# \# \# \underline{1} 1 0 \# \#)$

$\vdash M (q_4, \# \# \# 1 \underline{1} 0 \# \#)$

$\vdash M (q_4, \# \# \# \underline{1} 1 0 \# \#)$

$\vdash M (q_4, \# \# \# 1 1 \underline{0} \# \#)$

$\vdash M (q_4, \# \# \# 1 1 0 \underline{\#} \#)$

$\vdash M (q_5, \# \# \# 1 1 \underline{0} \# \#)$

$\vdash M (q_3, \# \# \# 1 \underline{1} \# \# \#)$

$\vdash M (q_3, \# \# \# \underline{1} 1 \# \# \#)$

$\vdash M (q_3, \# \# \underline{\#} 1 1 \# \# \#)$

$\vdash M (q_0, \# \# \# \underline{1} 1 \# \# \#)$

$\vdash M (q_1, \# \# \# \# \underline{1} \# \# \#)$

$\vdash M (q_1, \# \# \# \# 1 \underline{\#} \# \#)$

$\vdash M (q_2, \# \# \# \# \underline{1} \# \# \#)$

$\vdash M (q_3, \# \# \# \underline{\#} \# \# \# \#)$

$\vdash M (q_0, \# \# \# \# \underline{\#} \# \# \#)$

$\vdash M (q_6, \# \# \# \underline{\#} \# \# \# \#)$

Accepted

Since, q_6 is accepting state, so string '101101' is accepted.

6. Design a Turing machine (TM) which accepts the following language L = {w ∈ {x, y, z}*: w has equal no. of x's y's and z's}. Verify your design for the string '# xy xy zz #. [2073 Shrawan, Back]

Let, the required Turing machine be

$$T(M) = (\theta, \Sigma, \Gamma, b', \delta, q_0, F)$$

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

$\Sigma = \{x, y, z\}$

$\Gamma = \{x, y, z, \#, d\}$

$b' = \{\#\}$

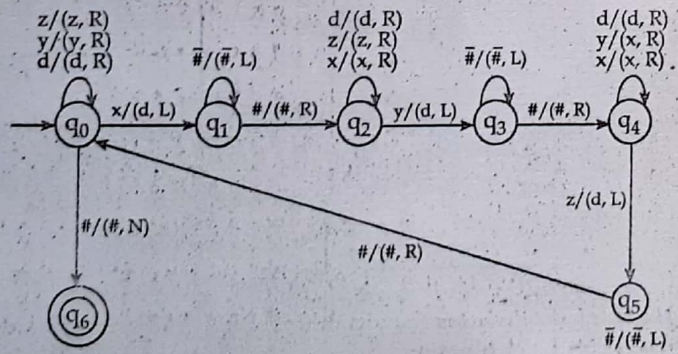
$q_0 = \{q_0\}$

$F = \{q_6\}$

and transition (δ) is defined as follows:

State/Input	x	y	z	d	#
q_0	(q_1, d, L)	(q_0, y, R)	(q_0, z, R)	(q_0, d, R)	$(q_6, \#, N)$
q_1	(q_1, x, L)	(q_1, y, L)	(q_1, z, L)	(q_1, d, L)	$(q_2, \#, R)$
q_2	(q_2, x, R)	(q_3, d, L)	(q_2, z, R)	(q_2, d, R)	–
q_3	(q_3, x, L)	(q_3, y, L)	(q_3, z, L)	(q_3, d, L)	$(q_4, \#, R)$
q_4	(q_4, x, R)	(q_4, y, R)	(q_5, d, L)	(q_4, d, R)	–
q_5	(q_5, x, L)	(q_5, y, L)	(q_5, z, L)	(q_5, d, L)	$(q_0, \#, R)$
q_6	–	–	–	–	–

States diagram



Here, represents non-empty symbols like x, y, z and d.

Let us process string # xyxyzz#

$(q_0, \# \underline{x}yxyzz\#) \vdash M(q_1, \# \underline{d}\, y\, x\, y\, z\, z\, \#)$

$\vdash M(q_2, \# d\, \underline{y}\, x\, y\, z\, z\, \#)$

$\vdash M(q_3, \# \underline{d}\, d\, x\, y\, z\, z\, \#)$

$\vdash M(q_3, \# d\, d\, x\, y\, z\, z\, \#)$

$\vdash M(q_4, \# \underline{d}\, d\, x\, y\, z\, z\, \#)$

$\vdash M(q_4, \# d\, \underline{d}\, x\, y\, z\, z\, \#)$

$\vdash M(q_4, \# d\, d\, \underline{x}\, y\, z\, z\, \#)$

$\vdash M(q_4, \# d\, d\, x\, \underline{y}\, z\, z\, \#)$

$\vdash M(q_4, \# d\, d\, x\, y\, \underline{z}\, z\, \#)$

$\vdash M(q_5, \# d\, d\, x\, \underline{y}\, d\, z\, \#)$

$\vdash M(q_5, \# d\, d\, \underline{x}\, y\, d\, z\, \#)$

$\vdash M(q_5, \# d\, \underline{d}\, x\, y\, d\, z\, \#)$

$\vdash M(q_5, \# \underline{d}\, d\, x\, y\, d\, z\, \#)$

$\vdash M(q_5, \underline{\#}\, d\, d\, x\, y\, d\, z\, \#)$

$\vdash M(q_0, \# \underline{d}\, d\, x\, y\, d\, z\, \#)$

$\vdash M(q_0, \# d\, \underline{d}\, x\, y\, d\, z\, \#)$

$\vdash M(q_0, \# d\, d\, \underline{x}\, y\, d\, z\, \#)$

$\vdash M(q_1, \# d\, \underline{d}\, d\, y\, d\, z\, \#)$

$\vdash M(q_1, \# \underline{d}\, d\, d\, y\, d\, z\, \#)$

$\vdash M(q_1, \underline{\#}\, d\, d\, d\, y\, d\, z\, \#)$

$\vdash M(q_2, \# \underline{d}\, d\, d\, y\, d\, z\, \#)$

$\vdash M(q_2, \# d\, \underline{d}\, d\, y\, d\, z\, \#)$

$\vdash M(q_2, \# d\, d\, \underline{d}\, y\, d\, z\, \#)$

$\vdash M(q_2, \# d\, d\, d\, \underline{y}\, d\, z\, \#)$

$\vdash M(q_3, \# d\, d\, \underline{d}\, d\, d\, z\, \#)$

$\vdash M(q_3, \# d\, \underline{d}\, d\, d\, d\, z\, \#)$

$\vdash M(q_3, \# \underline{d}\, d\, d\, d\, d\, z\, \#)$

$\vdash M(q_3, \underline{\#}\, d\, d\, d\, d\, d\, z\, \#)$

$\vdash M(q_4, \# \underline{d}\, d\, d\, d\, d\, z\, \#)$

$\vdash M(q_4, \# d\, \underline{d}\, d\, d\, d\, z\, \#)$

$\vdash M(q_4, \# d\, d\, \underline{d}\, d\, d\, z\, \#)$

$\vdash M(q_4, \# d d d \underline{d} d z \#)$
$\vdash M(q_4, \# d d d d \underline{d} z \#)$
$\vdash M(q_4, \# d d d d d \underline{z} \#)$
$\vdash M(q_5, \# d d d d \underline{d} d \#)$
$\vdash M(q_5, \# d d d \underline{d} d d \#)$
$\vdash M(q_5, \# d d \underline{d} d d d \#)$
$\vdash M(q_5, \# d \underline{d} d d d d \#)$
$\vdash M(q_5, \# \underline{d} d d d d d \#)$
$\vdash M(q_5, \underline{\#} d d d d d d \#)$
$\vdash M(q_0, \# \underline{d} d d d d d \#)$
$\vdash M(q_0, \# d \underline{d} d d d d \#)$
$\vdash M(q_0, \# d d \underline{d} d d d \#)$
$\vdash M(q_0, \# d d d \underline{d} d d \#)$
$\vdash M(q_0, \# d d d d \underline{d} d \#)$
$\vdash M(q_0, \# d d d d d \underline{d} \#)$
$\vdash M(q_0, \# d d d d d d \underline{\#})$
$\vdash M(q_6, \# d d d d d d \#)$ Accepted

Since, q_6 is final state, so string 'xyxyzz' is accepted.

7. Design a Turing machine to compute the function f(n) = n + 1, where n be a binary string show the processing for the string 10111. [2072 Chaitra][

Please see 2075 Ashwin, Back.

8. Construct a Turing machine to transform ⊔ w ⊔ into ⊔⊔w⊔, where w is a string containing no blanks and ⊔ represents blank.

Let, the required Turing machine be

$T(M) = (\theta, \Sigma, \Gamma, b', \delta, q_0, F)$

where

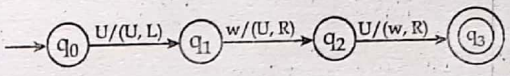
$\theta = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{\sqcup, w\}$
$\Gamma = \{\sqcup, w\}$
$b' = \{\sqcup\}$
$q_0 = \{q_0\}$
$F = \{q_3\}$

and transition (δ) is defined as follows:

State/Input	⊔	w
q_0	$(q_1, \sqcup, L)$	–
q_1	–	$(q_2, \sqcup, R)$
q_2	(q_3, w, R)	–
q_3	–	–

Note: The moves which are not shown are considered as rejecting states

States diagram



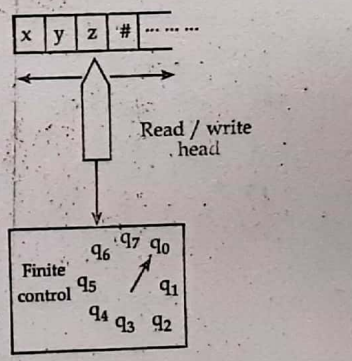
Let us process string ⊔w⊔⊔

(we can add blank symbol on rightmost side of string as many as we want because input tape contains infinite blank symbol on rightmost

$(q_0, \sqcup w \sqcup \sqcup) \vdash M(q_1, \sqcup \underline{w} \sqcup \sqcup)$
$\vdash M(q_2, \sqcup \sqcup \underline{\sqcup} \sqcup)$
$\vdash M(q_3, \sqcup \sqcup w \underline{\sqcup})$

9. Define head shifting and symbol writing Turing machines. Design a Turing machine (TM) which computes following function f(w) = ww^R, where w^R is the reverse of a string and w ∈ {0, 1}*. If your input string is # 01# then TM should given the output string as # 0110 #. [2071 Chaitra]

As we know, Turing machine consists of read / write head which performs its head shifting operation and symbol writing operation. While tape head is stationed at one of the tape cells, it scans or reads symbol from the cell and then moves in a particular direction. This course of action of move by tap3e head while scanning any input symbol is called head shifting. It's defined by following function



$(\theta \times \Gamma) \rightarrow (\theta \times \Gamma \times \{L, R, N\})$

where $\Gamma = \Sigma \cup \#$, 'L' denotes the tape head moves to the left adjacent cell, 'R' denotes tape head moves to the right adjacent cell and 'N' denotes head doesn't move, that is continuously scanning the same cell.

And when tape head moves to its adjacent cell tape head may or may not change currently scanned symbol. This is called symbol writing Turing machines.

Left the required Turing machine be

$$T(M) = (\theta, \Sigma, \Gamma, b, \delta, q_0, F)$$

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{0, 1, X, Y, \#\}$

$b = \{q_0\}$

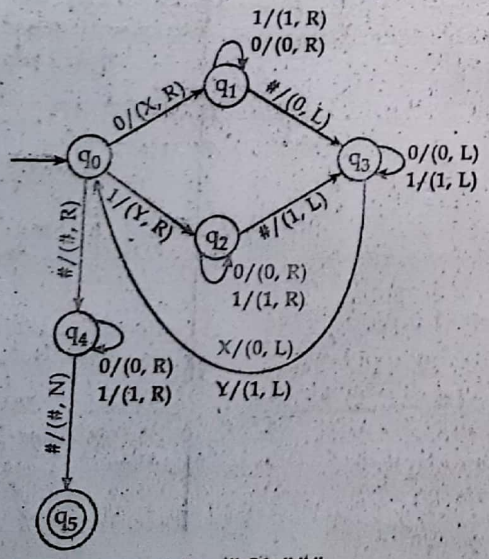
$q_0 = \{\#\}$

$F = \{q_5\}$

and transition (δ) is defined as follows:

State/Input	0	1	X	Y	#
q_0	(q_1, X, R)	(q_2, Y, R)	–	–	$(q_4, \#, R)$
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	–	–	$(q_3, 0, L)$
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	–	–	$(q_3, 1, L)$
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_0, 0, L)$	$(q_0, 1, L)$	–
q_4	$(q_4, 0, R)$	$(q_4, 1, R)$	–	–	$(q_5, \#, N)$
q_5	–	–	–	–	–

States diagram



Let us process string # 01 ###

$(q_0, \#\,\underline{0}1\,\#\#\#)$ $\vdash\!\!\!-$ M $(q_2, \#\,0\,Y\,\underline{\#}\,\#\,\#)$

$\vdash\!\!\!-$ M $(q_3, \#\,0\,Y\,1\,\#\,\#)$

$\vdash\!\!\!-$ M $(q_0, \#\,\underline{0}\,1\,1\,\#\,\#)$

$\vdash\!\!\!-$ M $(q_1, \#\,x\,\underline{1}\,1\,\#\,\#)$

$\vdash\!\!\!-$ M $(q_1, \#\,x\,1\,\underline{1}\,\#\,\#)$

$\vdash\!\!\!-$ M $(q_1, \#\,x\,1\,1\,\underline{\#}\,\#)$

$\vdash\!\!\!-$ M $(q_3, \#\,x\,1\,\underline{1}\,0\,\#)$

$\vdash\!\!\!-$ M $(q_3, \#\,x\,\underline{1}\,1\,0\,\#)$

$\vdash\!\!\!-$ M $(q_3, \#\,\underline{x}\,1\,1\,0\,\#)$

$\vdash\!\!\!-$ M $(q_0, \underline{\#}\,0\,1\,1\,0\,\#)$

$\vdash\!\!\!-$ M $(q_4, \#\,\underline{0}\,1\,1\,0\,\#)$

$\vdash\!\!\!-$ M $(q_4, \#\,0\,\underline{1}\,1\,0\,\#)$

$\vdash\!\!\!-$ M $(q_4, \#\,0\,1\,\underline{1}\,0\,\#)$

$\vdash\!\!\!-$ M $(q_4, \#\,0\,1\,1\,\underline{0}\,\#)$

$\vdash\!\!\!-$ M $(q_4, \#\,0\,1\,1\,0\,\underline{\#})$

$\vdash\!\!\!-$ M $(q_5, \#\,0\,1\,1\,0\,\underline{\#})$

Accepted

10. Define the term configuration of Turing machine. Design a Turing machine which accepts the set of all palindromes over alphabet {0, 1}.

[2070 Chaitra]

The configuration of Turing machine is defined below:

$$\theta X \Gamma \rightarrow (\theta' X \Gamma X \{L \mid R \mid N\})$$

where

θ → initial state

θ' → final state (may be equal to initial state)

Γ → $\Sigma \cup \#$ 'L' denotes the tape head moves to the left adjacent cell, 'R' denotes tape head moves to the right adjacent cells and 'N' denotes head doesn't move, that is continuously scanning the same cell.

Please see 2073 Chaitra Turing machine

11. How multiple-tape Turing machine is different from multi-track Turing machine? Does any variation of Turing machine have more computational power than standard Turing machine?

[2075 Ashwin, Back]

Multiple tape Turing machine has multiple read/write head on multiple tape (i.e. separate head for separate tape) whereas multi-track Turing machine has single read/write on multiple track. All the separate heads in multiple tape Turing machine can work independently whereas it can't do so in the case of multi-track Turing machine we can write different symbols in multi-cell of multiple tape Turing machine but in the case of multi-track Turing machine we can write same symbols in all tracks.

Any variation of Turing machine like (multiple tape Turing machine, multi-track Turing machine, etc and so on) doesn't contribute in having more computational power than standard Turing machine. The reason is all of them are equivalent and can be simulated on each other. For example; standard Turing machine can be simulated on two-way infinite tape. The pictorial representations of two way infinite tape and standard Turing machine are shown below:

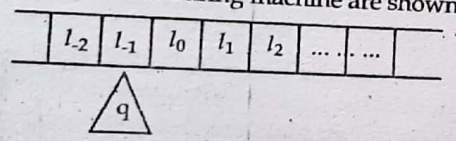


Fig. (i) Two way infinite tape

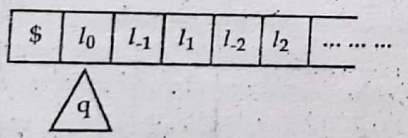


Fig. (ii) Standard Turing machine

Let's see how can two way infinite tape be converted on standard Turing machine to do so, we simply don't use the negative index of the two-way infinite tape as shown in above fig. (i). If the Turing machine with a single side infinite tapes relies on hanging for its behaviour by moving one step after the leftmost tape location, we can still model this one our double sided (two-way) infinite tape Turing machine. We simple use a new symbol ($ say(as a part of our alphabet as shown in fig. (ii) and place it at location $l - 1$ to delimit the end of the tape. We then add a "hanging transition rule" to our Turing machine (i.e. we transition to a struck state) whenever we hit this special symbol is as to model hanging in a single sided (standard) Turing machine.

Similarly, multiple tape Turing machine can be simulated on standard Turing machine or vice-versa. And the key to this problem is, if we use only one tape of multiple tape, it can be simulated on standard Turing machine. And if we combine many standard Turing machine, it could act as multiple tape Turing machine. So, all the variations of Turing machine (M') and standard Turing machine (m) accepts the same language i.e.

$$L(M') = L(M)$$

Therefore, Turing machine with variations doesn't increase the class of functions that can be computed by standard Turing machine and have the same computational power as that of standard Turing machine.

12. Compare Turing machine with finite automata and push down automata (PDA). [2073 Chaitra]

The comparison between Turing machine and finite automata are as follows:

Turing machine	Finite automata
(a) Turing machine is the generalization of finite automata.	(a) Finite automata is a kind of restricted Turing machine.
(b) It recognizes the recursively enumerable languages which is superset of regular language.	(b) It recognizes the regular language which is subset of recursively enumerable language.
(c) It can perform read and write operations i.e. it can change the inputs.	(c) It can't change the inputs.
(d) It has more memory than finite automata.	(d) It has limited memory.
(e) It has more computation power.	(e) It has less computational power.
(f) It has infinite tape as its storing element.	(f) It has current state for its storage.

The comparison between Turing machine and push-down automata (PDA) are as follows:

Turing machine	Push-down automata (PDA)
(a) It can be called as push down automata with infinite input tape.	(a) It can be described as older version of Turing machine.
(b) It is composed of infinite single or many tape.	(b) It has single stack as its strong element.
(c) It can access any input or alphabet from any position on an infinite tape.	(c) It can access the alphabet at the top of stack only in LIFO sequence (Last in first out)
(d) Turing machine recognizes recursively enumerable language.	(d) PDA recognizes the context free languages which is subset of recursively enumerable languages.
(e) It has more computation power than PDA.	(e) It has less computational power than Turing machine.
(f) It can perform almost all type of computation like addition, subtraction, multiplication, search, etc.	(f) It can't perform all types of computation.

13. **Define multiple Turing machine.** [2072 Chaitra]

A multiple Turing machine is like an ordinary Turing machine with several tapes. Each tape has its own head for reading and writing. Initially the input appears on tape and other start out blank. Multiple tape machines can't calculate any more functions than single tape machines.

A multiple tape Turing machine can be described as 6-tuple

$M = (\theta, \Gamma, S, b, F, \delta)$ where

$\theta \rightarrow$ finite set of states

$\Gamma \rightarrow$ finite set of the tape alphabet

$S \rightarrow$ initial state, $S \in \theta$

$d \rightarrow$ blank symbol, $b \rightarrow \Gamma$

$F \subseteq \theta$ is the set of final or accepting states

$\delta : \theta X \Gamma^k \rightarrow (\theta X \Gamma X \{L, N, R\})^K$ is a partial function called transition function, where K is the number of tapes, L is left shift, R is right shift and S is no shift.

14. **What is Turing machine? Describe its operation.** [2072 Kartick, Back]

A Turing machine is a pushdown automata with infinite input tape. It consists of a finite control, read/write head and a infinite input tape as shown in fig (i) below. The tape has series of squares to head a single symbol and head moves one square in particular direction.

A Turing machine is a 7 tuple described as described as

$$M = (\theta, \Sigma, \Gamma, b, \delta, q_0, F)$$

where

$\theta \rightarrow$ finite set of states

$\Sigma \rightarrow$ input alphabets

$\Gamma \rightarrow$ tape alphabets including blank symbol 'b'

$b \rightarrow$ blanks symbol

$\delta \rightarrow$ transition function; $\theta X \Gamma \rightarrow \theta X \Gamma X \{L, R, N\}$

$q_0 \rightarrow$ initial state

$F \rightarrow$ finial state

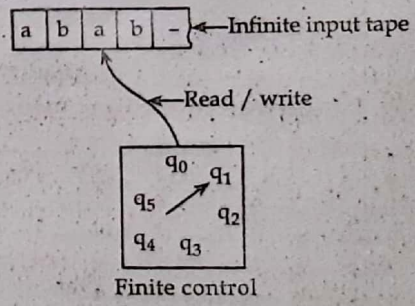
Operation of Turing machine

In essence, a Turing machine consists of a finite state control and a tape communication between two is provided by a single head which reads symbols from the tape and is used to change the symbols on the tape control unit operates in discrete steps. At each steps, it performs two functions in a way that depends on its current state and the tape symbol currently scanned by the read/write head.

(a) Put the control unit in a new state.

(b) (i) Write a symbol in the tape square currently scanned, replacing the one already there

(ii) More the read/write head one tape square to the left or right or non move at all.



15. Design a Turing machine that reads binary string and doubles the number represented by that string. A binary number is doubled if a '0' is added on the right end of the number. [2071 Chaitra]

Let, the required Turing machine be

$T(M) = (\theta, \Sigma, \Gamma, b, \delta, q_0, F)$

where

$\theta = (q_0, q_1)$

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

$\Gamma = \{0, 1, \#\}$

$b = \{\#\}$

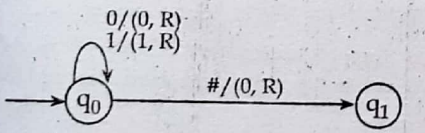
$q_0 = \{q_0\}$

$F = \{q_1\}$

and transition (δ) is defined as follows:

State/Input	0	1	#
q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	$(q_1, 0, R)$
q_1	–	–	–

States diagram



Let us process string # 100 ##

$(q_0, \# \underline{1}00\ \#\#) \vdash M (q_0, \# 1\underline{0}0\ \#\#)$

$\vdash M (q_0, \# 10\underline{0}\ \#\#)$

$\vdash M (q_0, \# 100\underline{\#}\ \#)$

$\vdash M (q_1, \# 100\ \#\underline{\#})$

16. Is Turing machine a complete computer, support your answer in reference to different roles of Turing machines? [2070 Chaitra]]

It is a theoretical and thought model of a computer for computation. It has never had physical existence though some scientists designed it somehow later it has infinite memory. But its memory can be made limited to perform all computation that a real computer (or RAM more specifically).

Turing machine can simulate any type of subroutine found in programming languages including recursive procedures and any of the known parameter passing mechanisms it could perform all kind of mathematical computation like addition, subtraction, multiplication, etc. Turing machines are equivalent to RAM. It may look as though RAM model is more efficient. Compared to TMS when such operations like addition or subtraction are considered. But for string operation like concatenating two strings x and y, TM may look more efficient.

According to church Turing thesis, what can be computed by an algorithm or real computer can also be computed by Turing machine. And no one has yet been able to suggest a problem solvable by what we consider an algorithm, for which a Turing machine can't be designed. Therefore, Turing machine is capable of simulating complete computer.

17. Define multiple tapes Turing machine. With reference to linage they accept, compare multiple tapes Turing machine with single tape Turing machine. [2070 Chaitra]

For definition of multi-tapes Turing machine, please see 2072 Chaitra.

The language that is accepted by both multi tape (or multiple tape) Turing machine is decidable (or recursive) language and the comparison between multiple tapes Turing machine and single tape Turing machine are as follows:

Multiple tape Turing machine	Single tape Turing machine
(a) This type of Turing machine has m number of tapes with m tape heads.	(a) This type of Turing machine has single tape with single head.
(b) The time complexity of this type of Turing machine is comparatively faster than single tape Turing machine.	(b) The time complexity of this type of Turing machine is comparatively slower than multiple tape Turing machine.
(c) It is usually defined as six-tuples i.e. $M = (Q, \Sigma\, \Gamma, \delta, q_0, h)$	(c) It is usually defined as 7-tuples $M = (Q, \Sigma, \Gamma, b, \delta, q_0, h)$
(d) The transition function of this type of Turing machine is $(Q \times \Sigma^m) \rightarrow (Q \times \Sigma \times \{L, R, N\}^m)$ where m is the number of tape	(d) The transition function of this type of Turing machine is $Q \times \Sigma \rightarrow (Q \times \Sigma \times \{L, R, N\})$
(e) In this type of Turing machine, one tape is usually used for input and other tapes are used for output.	(e) In this type of Turing machine, the only one same tape is used for both reading (input) and writing (output)

18. Define unrestricted grammar. Explain possible extensions of Turing machine in brief. [2074 Ashwin]

Unrestricted grammar is defined as

$G = \{V_n, V_t, P, S\}$

where

V_n = a finite set of non-terminals

V_t = a finite set of terminals

S = starting non-terminal, $S \in V_n$

and P is set of productions of the following form.

$\alpha \rightarrow \beta$

where α and β are arbitrary strings of grammar symbols with $\alpha \neq \in \alpha$ must consists of a non-terminal or more with any number of terminals. And β is any string of terminals and non-terminals what so ever.

The possible extensions of Turing machine are described below:

(a) **Multi tape Turing machine:** The Turing machine that have several tapes (as shown in fig below). Each tape is connected to the finite control by means of their separate read/write head. The machine can in one step reads the symbol scanned by all its heads and then, depending on these symbols in current state, rewrite some of those scanned squares and move some of the heads to the left or right in addition to changing the state. Multi tape Turing machine have same computational power as that of a standard Turing machine. Having many tapes in such machines doesn't increase the class of computation it can solve.

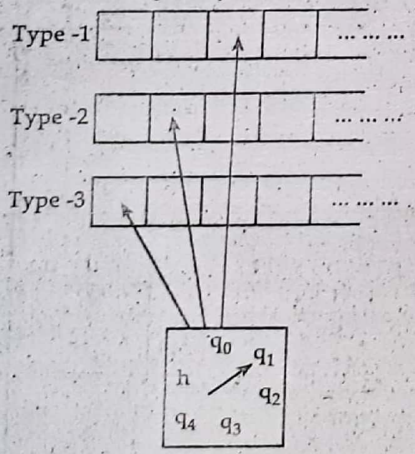


Fig. Multitape Turing machine

(b) **Multi-track Turing machine:** A multi track Turing machine is a specific type of multi tape Turing machine. In a standard n-tape Turing machine (multi-track Turing machine) in heads move independently along n tracks. In a n-track Turing machine, one head reads and writes on all tracks simultaneously. A tape position in a n-track Turing machine contains n symbols from the tape alphabet. It is equivalent to standard Turing machine.

(c) **Multidimensional (or K-dimensional) Turing machine:** A "multidimensional Turing machine" has a multidimensional "tape". For example, a two-dimensional Turing machine would read and write on an infinite plane divided into squares, like a checker board. Possible directions that the tape head could move might be labeled {N, E, S, W}.

(d) **Non-deterministic Turing machine:** In a non-deterministic Turing machine, for every state and symbol, there are group of actions the TM can have. The computation of a non-deterministic Turing machine is a tree of configurations that can be reached from the start configuration. Standard Turing machine is a special case of the NDTM (non-deterministic Turing machine) in which for each (q, a) the set {(qi, ai, mi)} in next moves is a singleton set or empty.

(e) **Semi-infinite tape:** The Turing that has non-blank input at the extreme left and of the tape.

(f) **Two-way infinite tape Turing machine:** It has continuous sequence of blanks on each of the right hand and left hand of the sequence of non-blanks.

(g) **Offline Turing machine:** It has two tapes, one for read-only and contains the input, the other is read-write and is initially blank.

19. **Explain how unrestricted grammar can be used to generate the language**

$$L = \{a^n b^n c^n : n > 0\}$$

is there any difference between CFG and unrestricted grammar? Explain. [2074 Chaitra]

Please see example 1 of unrestricted grammar in theory part.

The major differences between CFG context free grammar and unrestricted grammar are as follows:

Unrestricted grammar	CFG
(a) Unrestricted grammar follows pattern $\alpha \rightarrow \beta$ where α can be any string of terminals and non-terminals that contains at least one non-terminal and β can be any string of terminals and non-terminals what so ever.	(a) Context free grammar (CFG) follows pattern $\alpha \rightarrow \beta$ where α can be only one non-terminal and β can be string of terminals and non-terminals what so ever.
(b) It is the superset of CFG.	(b) It is the subset of unrestricted grammar.
(c) It accepts recursively enumerable languages.	(c) It accepts context free language.
(d) It is also called type-0 grammar in Chomsky hierarchy.	(d) It is also called type-2 grammar in Chomsky hierarchy.
(e) It accepts language $L = \{a^n b^n c^n : n > 0\}$	(e) It doesn't accept language $L = \{a^n b^n c^n\}$
(f) The properties of unrestricted grammar is closed under intersection.	(f) The properties of context free grammar is not closed under intersection.

20. Explain unrestricted grammar with suitable examples. Is unrestricted grammar is superset of context free grammar? Justify your answer. **[2073 Chaitra]**

Please see theory part for definition and example.

The relationship between grammar by the Chomsky Hierarchy is given below:

Type O languages are those generated by unrestricted grammars, that is the recursively enumerable languages. Type - 1 consists of context sensitive language. Type – 2 consists of the context free languages and Type – 3 consists of regular languages. From above diagram, it can be said that family of type K is a paper subset of the family of type k – 1 or type 0 language is the superset of family of type 1, type 2, type 3 languages. That is unrestricted grammar can accept all the languages accepted by other grammars. And hence, unrestricted grammar is superset of context free grammar.

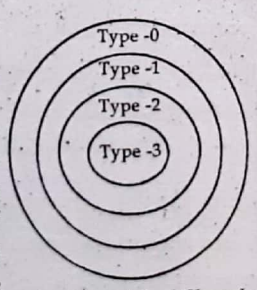


Fig. The original Chomsky Hierarchy

21. Define unrestricted grammar. Explain, how unrestricted grammar can be defined as superset of CFG and regular grammar? **[2073 Shrawan, Back]**

Unrestricted grammar can be defined as

$G = (V_n, V_t, P, S)$

where

V_n = a finite set of non-terminals

V_t = a finite set of terminals

S = starting non-terminal, $S \in V_n$

and P is set of productions of the following form

$\alpha \rightarrow \beta$

where α can be any string of terminals and non-terminals that contains at least one non-terminal and β can be any string of terminals and non-terminals what so ever. For example $aA \rightarrow aa$, here α is aA and β is aa.

For explanation, please see 2073 Chaitra .

22. Construct a grammar to accept the language $L\{a^n b^n c^n : n \geq 1\}$. **[2072 Kartik, Back]**

Please see example 1 of unrestricted grammar in theory part.

23. Let M_1, M_2 and M_3 be three Turing machines, can you combine these Turing machines to get new Turing machine M? If yes, elaborate your idea with required theory and illustration. Explain unrestricted grammar with suitable example. **[2071 Shrawan, Back]**

We can build complex Turing machines by combining simple one. Let us see first how we can combine two simple Turing machine and then we would get to know how many Turing machines can be combined to build complex Turing machine.

Suppose M_1 is a Turing machine that is to be made part of a large Turing machine 'M'. It is assumed that M_1 doesn't hang on input. M prepare some string as input to M_1, placing it near the right end of non-blank portion of the tape, passes control to M_1 with read/write head just beyond the end of that string and finally retrieves control from M_1 when M_1 has finished its computing. It is guaranteed that M_1 will never interfere with the operational computation of M.

Now, we adapt a graphical representation to show the combining of Turing machines. The connection from one machine to another machine (as given in fig (i)) will not be pursued until the first

machine (M_1) halts; the other machine (M_2) is then started from its initial state with the tape and head position as they were left by the first machine.

Fig. (i)

Here M_1 starts computing and halts and then control goes to M_2.

Let M_1, M_2 and M_3 be the three Turing machines which are combined to form Turing machine M. The graphical representation of combination of Turing machines is given below:

Now, this machine starts computing with machine M. Now when M halts, the symbol on the tape under the current bead position can be a, b or #. Then fig (ii) shows when M halts, the control goes to machine M_1, M_2 or M_3 according to the symbols under current head position which is a, # or b respectively. This is how we can combine three Turing machines M_1, M_2 and M_3 to form Turing machine M.

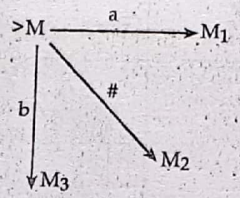


Fig. (ii)

For explanation of unrestricted grammar and examples please see theory part.

◆◆◆

Chapter - 5

Undecidability

Church Truing Thesis

'How can you define computation? or 'What is computable?' These are the questions that arose in the field of computer science.

Alonzo church, came up with a ideal LAMBDA CALCULUS and he said that whatever can be computed by LAMBDA CALOCULUS can be considered as computable.

And then, Allen Turing, who was the student of Alonzo church, came up with a idea and said that whatever can be computed by TURING MACHINE can be considered as computable or algorithm. And they finally agreed on the concept of Turing machine i.e.

"No computational procedure will be considered an algorithm unless it can be represented as a Turing machine." this statement is called Church. Thesis because Alonzo Church gave many sophisticated reasons for believing it. Church's original statement was slightly different because his thesis was presented slightly before the Turing invented his machines.

Church actually said that any machine that can do certain list of operations will be able to perform all conceivable algorithms. He tied together what logicians had called recursive functions and computable functions. Turing Machine (Tm's) can do all that church asked, so they are one possible model of the universal algorithm machine church described.

Unfortunately Church's thesis can't be a theorem in mathematics because ideas such as "can ever be defined by humans" and "algorithm that people can taught to perform" aren't part of any known mathematics. There are no axioms that deals with "people".

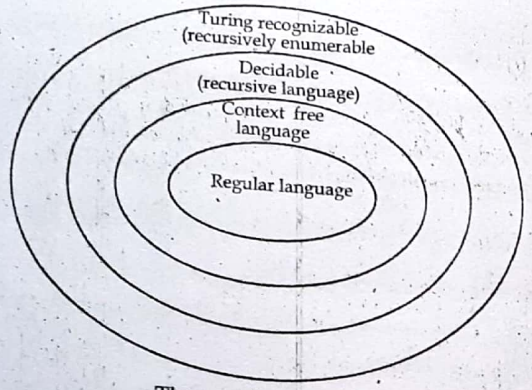


Fig. Chomsky hierarchy

Universal Turing Machine

Universal Turing machines 'U' takes two arguments, a description of a machine Tm, "Tm" and a description of an input string W, "W" we want U to have the following property: U halts on input "Tm", "W" if and only if Tm halts on input W.

It is the functional notation of universal Turing machine.

In general, it is more like a computer in which we write description of a Turing machine (Tm) and inputs (w) of that Turing machine to make it behave like Tm or,

A universal Turing machine is a Turing machine that can simulate an arbitrary Turing machine on arbitrary input. The universal machine essentially achieve this by reading both the description of the machine to be simulated as well as the input thereof from its own tape.

To justify this we have to define a language whose strings are all legal representations of Turing machines, we adopt the following convention. A string representing a Turing machine state is of the form {q} {0, 1}*, that is, the letter q followed by a binary string. Similarly, a tape symbol is always represented as a string in {a} {0, 1}*.

Let Tm = (Q, Σ, δ, s) be a universal Turing machine and k and *l* be the smallest integer such that $2^k \geq |Q|$ and $2^l \geq |\Sigma + 2$. Then, each state in Q will be represented by 'q' followed by a binary string of length K, each symbol in Σ will be likewise represented as letter 'a' followed by a binary string of *l* bits. The head directions that left (L) and right (R) are included in input tape symbols to justify "+2" term in definition of *l*.

Example, construct a Turing machine m = (Q, Σ, δ, s) where

Q = {s, q, h}

Σ = {#, b, a} and δ is given in following table

States	Symbol	δ
s	a	(q, #)
s	#	(h, #)
s	b	(s, R)
q	a	(s, R)
q	#	(s, R)
q	b	(q, R)

Since there are three states, three symbols in Σ, we have k = 2 and *l* = 3. These are the smallest integers such that $2^k \geq 3$ and $2^l \geq 3 + 2$. The states and symbols are represented as follows:

States	Representation
s	q00
q	q01
n	q11
#	a000
b	a001
L	a010
R	a011
a	a100

Thus the representation of baa # a is "baa # a" = a001a100a100 a000a100 .

The representing "Tm" of the Turing machine Tm is the following strings.

"Tm" = (q00, a100, q01, q000)

(q00, a000, q11, a000), (q00, a001, q00, a011),

(q01, a100, q00, a011), (q01, a000, q00, a011)

(q01, a001, q01, a011)

These are the transition function stated, in δ table of form (s, a, q, #)

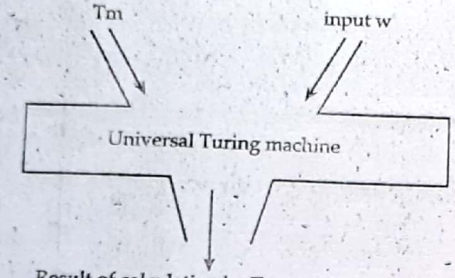


Fig. Pictorial representation of universal Turing machine

Halting Problem

Let assume that we have given the description of Turing machine Tm and an input w, when started in initial configuration q_0w, perform a computation that eventually halts? Using an abbreviated way of talking about the problem, we ask whether Tm applied to w, or simply (Tm, w) halts or doesn't halt. The domain of this problem is to be taken as the set of all Turing machines and all W, that is we are looking for a single Turing machine, that, given the description of an arbitrary Tm and w, will predict whether or not the computation of Tm applied to w will halt.

We can't find the answer by simulating the action of Tm on w, say by performing it on universal Turing machine, because there is no limit on the length of the computation. If Tm enters an infinite loop, then no matter how long we wait, we can never be sure that Tm is in fact in a loop. It may be sample case of very long computation. What we need is a generalized algorithm that can determine the correct answer for any Tm and w by performing some theoretical analysis on the machine's description and the input. But it is clear that no such algorithm exists.

In short halting problem is

To determine for an arbitrary given Turing machine Tm and input w, whether Tm will eventually half on input w.

OR

Can we design a generalized algorithm that tell us about halting or not halting condition of Turing machine on particular input string without running test of input string in Turing machine?

And the answers to above question are:

(a) In general we can't always know.

(b) The best we can do is run the program and see whether it halts.

(c) For many programs we can see that it will always halt or sometimes loop.

Example of halting problem: The blank tape halting problem.

Given a TM M, decide whether or not TM halts on a blank tape.

Steps: to prove halting problem

Step I: We reduce the halting problem to the blank tape halting problem.

Step II: Assume that a Turing machine TM A solves the blank tape problem.

Step III: Construct a Turing machine TM that solves the halting problem (illustrate).

1. Given (M, W), construct Mw that, given a blank tape.
   (a) Writes W onto the tape.
   (b) puts itself in the configuration q_0W.
   (c) behaves like M thereafter.
2. If ($A(M_w)$ = = "yes")
   return "yes"; // M halts on θw else.
   return "no"; // M doesn't halt W.

Step IV: But, the halting problem is undecidable.

Step V: Conclusion our assumption that A exists (Turing machine that halts on "input string ∈ empty string) is false.

Undecidable Problems

The problem for which no algorithm or no Turing machine exists for their solution are considered to be undecidable problems. Undecidable problems mostly contain recursive enumerable language. Here, we would mainly focus on undecidable problems for Turing machine and grammars.

Undecidable problems for Turing machine

(a) Given a Turning machine M and string W, does M halt on W?

(b) Given a Turing machine M, does M halt on blank / empty tape?

(c) Given a Turing machine M, does there exist a string W such that M halts on W i.e. $L(M) = \phi$?

(d) Given a Turing machine M, does M halt on all strings i.e. is $L(M) = \Sigma^*$?

(e) Given Turing machines M_1 and M_2, do they accept the same language $L(M_1) = L(M_2)$?

(f) There is a specific Tm Mo for which the problem is undecidable: given W, does Mo half on W?

(g) Given a Turing machine, M, does M accepts regular language, context free language or recursive languages or not?

Problems about grammars

(a) Given, grammar G, determine whether G is ambiguous or not?

(b) Given a grammar G, does empty string $\in$ belong to G i.e. $e \in L(G)$?

(c) Given a grammar G, is $L(G) = \phi$?

(d) Given a grammar G, does all string contained by grammar or not i.e. $L(G) = \Sigma^*$?

(e) Given grammars G_1 and G_2, is $L(G_1) = L(G_2)$?

(f) Given a grammar G and string W, does string W can be produced by grammar G or not i.e. is $W \in L(G)$?

Recursively Enumerable and Recursive language

When a Turing machine executes an input, there are four possible outcomes of execuition. Then Tm.

(a) Halts and accept the input.

(b) Halts and reject the input.

(c) Never halts (fall into loop)

(d) Crash

Recursive language

A language L is said to be recursive if there exists a Turing machine 'Tm' that accepts every strings belonging to language L(i.e. $W \in L$) and rejects all the strings that doesn't belong to language L (i.e. $W \notin L$). The Turing machine will halt every time and give an answer (accepted or rejected) for each and every string input. Therefore, these languages are always decidable.

Recursively enumerable language

A language 'L' is said to be recursively enumerable language if there exists a Turing machine Tm that only accepts the input string 'W' which belongs to language L (i.e. $w \in L$). But it may or may not halt for all input strings which are not in L.

Decidable languages: A language L is decidable it is a recursive language. All decidable languages are recursive language and vice-versa.

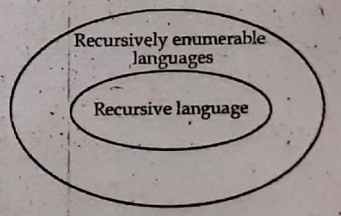
Partially decidable language: A language 'L' is partially decidable if 'L' is a recursively enumerable language.

Undecidable language: A language L is undecidable language if it is not decidable or partially decidable language if a language is not even partially decidable, then there exists no Turing machine for that language.

Theorem: Prove that "If language L is recursive then it is recursively enumerable."

Proof: A language is recursively enumerable if there exists a Turing machine that accepts every string of the language i.e. $w \in L$. so, this type of Turing machine may not halt every time for some input string w.

A language is recursive if there is a Turing machine Tm that accepts every string of the language $w \in L$ and doesn't accept strings that aren't in the language i.e. $\notin L$. And the Turing machine accepts such languages always either 'halts and accept) or (halt and reject) the string.



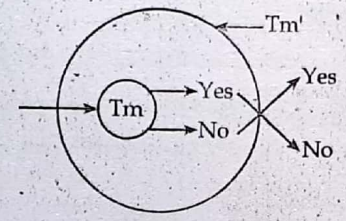
So, every recursive language is also recursively enumerable. But the vice-versa is not true. Hence, proved.

Properties of Recursive and Enumerable Languages

Property 1: The complement of a recursive language is recursive.

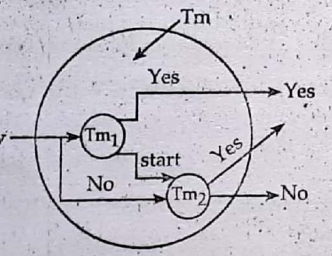
Proof: Let L be a recursive language and Tm be Turing machine that halts on all inputs and accepts L. Let us construct a Turing machine Tm' from Tm so that if Tm enters a final state on input w, then Tm' halts without accepting. If Tm halts without accepting Tm' enters a final state.

Since one of these two events occurs, Tm' is an algorithm. So clearly T (Tm') is the complement of L and thus the complement of L is recursive language. Following figure shows construction of Tm' from Tm.



Property 2: The union of two recursive languages is recursive

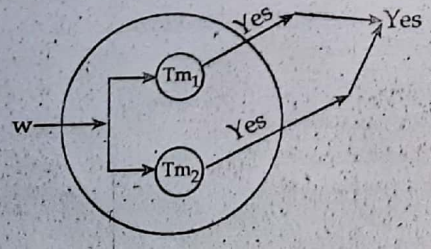
Proof: Let L_1 and L_2 be the two recursive languages accepted by truing machines Tm_1 and Tm_2. We construct a Turing machine Tm, which first simulate Tm_1. If Tm_1 accepts, then Tm accepts. If Tm_1 rejects then Tm simulates Tm_2 and accepts if and only if Tm_2 accepts. Since both Tm_1 and Tm_2 are algorithm so Tm is guaranteed to halt. Clearly Tm accepts $L_1 \cup L_2$.



So, $L_1 \cup L_2$ is also recursive since there exist a Turing machine Tm for it.

Property 3: The union of two recursively enumerable languages is recursively enumerable.

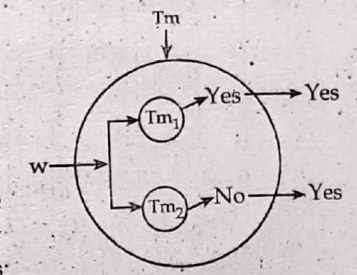
Proof: We have studied earlier that enumerating Turing machine have one input tape and one output tape.



Let L_1 and L_2 be recursively enumerable languages and their enumerative Turing machines are Tm_1 and Tm_2 respectively. Let us design a Turing machine Tm which can simulate Tm_1 and Tm_2 simultaneously on separate tape. If either accepts, then Tm accepts as follows.

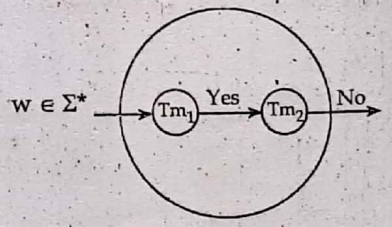
Property 4: If a language L and its complement $\bar{L}$ are both recursively enumerable, then L(and hence $\bar{L}$) is recursive.

Proof: Let Tm_1 and Tm_2 accept L and $\bar{L}$ respectively. Let us construct a truing machine Tm which simulate Tm_1 and Tm_2 simultaneously Tm accepts w if Tm_1 accepts and rejects w if Tm_2 will accept. Thus, Tm will always say either 'Yes' or 'No', but never say both. Note that there is not a priority limit on how long it may take before Tm_1 or Tm_2 accepts, but it is certain that one or the other will do so. Since Tm is algorithm that accepts L, it follows that L is recursive.



Property 5: If L is a recursive language then $\Sigma^* - L$ is recursive.

Proof: The required Turing machine Tm complement can be represented by following diagram.



The machine Tm complement functions as follows: When a string $w \in \Sigma^*$ is given an input to Tm complement, its control passes the string to Tm_1 as input to Tm_1. As Tm_1 decides the language L, therefore, for $w \in L$ after a finite number of moves, Tm_1 outputs "Yes" which is given as input to Tm_2, which in turn returns "No."

Similarly, for $w \notin L$, Tm_2 returns "Yes'. Hence, there exists a Turing machine Tm complement, for $\Sigma^* - L$. So, it is Turing decidable, that is recursive.

Exam Solution

1. Describe in detail about on universal Turing machines with examples. [2075 Ashwin, Back]

Universal Turing machine is a Turing machine that can simulate an arbitrary Turing machine on arbitrary input. The universal machine essentially achieve this by reading both description of machine to be simulated as well as the input thereof from its tape.

In other words, universal Turing machine 'U' takes two arguments, a description of a machine Tm, "Tm" and a description of a machine Tm, "Tm" and a description of an input string w, "W". We want 'U' to have following property U halts on input 'Tm', 'w' if and only if Tm halts on input w.

U("m" "m") = "m(w)"

It is functional notation of Turing machine Binary representation of Turing machine.

To feed the description of a Turing machine m and input string 'w'. We adopt the following convention.

Let Tm = (Q, Σ, δ, s) be a Turing machine and k and l be smallest integer such that $2^k \geq |Q|$ and $2^l > |\Sigma| + 2$. Then each state in Q will be represented by q followed by a binary string of length k; each symbol in Σ will be likewise represented as letter 'a' followed by a binary strings of l bits. The head directions that left (L) and right (R) are included in input tape symbols to justify "+2" term in definition of L.

Example, consider the Turing machine Tm = (Q, Σ, δ, s) where

Q = {s, q, h}

Σ = {#, b, a} and δ is given in following table.

States	Symbol	δ
s	a	(q, #)
s	#	(h, #)
s	b	(s, R)
q	a	(s, R)
q	#	(s, R)
q	b	(q, R)

The transition table is written arbitary. Since, there are three states, three symbols in Σ, we have k = 2 and l = 3. These are the smallest integers such that $2^k \geq 3$ and $2^l > 3 + 2$. The states and symbols are represented as follows:

States	Representation
s	q00
q	q01
h	q11
#	a000
b	a001
L	a010
R	a011
a	a100

Thus, representation of input string 'w' = baa # a (say) for universal Turing machine is

"baa # a" = a001a100a100a000a100

And transition (δ) of Truing machine m is the following strings.

"Tm" = (q00, a100, q01, a000), (q00, a000, q11, a000), (q00, a001, q00, a011), (q01, a100, q00, a011), (q01, a000, q00, a011), (q01, a000, q00, a011), (q01, a001, q01, a011)

These are the transition function stated in table of from (s, a, q, #).

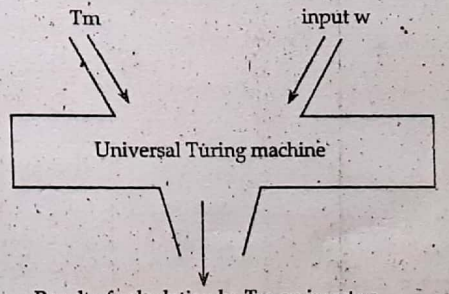


Fig. Pictorial representation of universal Turing machine

2. Explain the church Turing thesis show that the "halting problem" is undecidable. [2075 Ashwin, Back]

Church Turing thesis states that "No computational procedure will be considered as an algorithm unless it can be represented as a Turing machine." In simple words, "A Turing machine can compute anything that can be computed by a general purpose digital computer. This statement is called church Turing thesis because Alonzo church (1936), gave many sophisticated reasons for believing it Church's original statement was slighting different from this thesis because his thesis was presented slightly before the Turing invented his machines.

Church actually said that any machine that can do all list of operations will be able to perform all conceivable algorithms. He tied together what logicians had called recursive functions and computable functions Tm's can do all that church asked. So they are one possible model of the universal algorithm machine church described.

To prove: $HALT_{Tm} = |(Tm, w)|$. The Turing machine Tm halts on input w is undecidable (halting problem)

Proof: We assume that $HALT_{Tm}$ is decidable, and get a contradiction. Let Tm_1 be the Turing machine such that.

$T(Tm_1) = HALT_{Tm}$ and Tm_1 halt eventually on all (Tm, w). We construct a Turing machine Tm_2 is follows:

(a) For Tm_2, (Tm, w) is an input

(b) The Tm_1, acts on (Tm, w)

(c) If Tm_1 rejects (Tm, w) then Tm_2 rejects (Tm, w)

(d) If Tm_1 accepts (Tm, w), simulate Turing machine Tm on the input string w until Tm halts.

(e) If Tm has accepted w, Tm_2 accepts (Tm, w) otherwise Tm_2 rejects (Tm, w).

When Tm_1 accepts (Tm, w) (in step qd), the Turing machine Tm halts on w. In this case either an accepting q or a state q' such that δ (q', a) is undefined till some symbol a in w is reached. In the first case (the first alternative of step e) Tm_2 accepts (Tm, w). In the second case (the second alternative o step e) Tm_2 reject (Tm, w).

It follows from the definition of Tm_2 that Tm_2 halt eventually.

Also, $T(Tm_2)$ = |(Tm, w)| the Turing machine accepts w.

$= A_{Tm}$

This is a contradiction since A_{Tm} is undecidable.

Note: A_{Tm} is Turing machine which halts and accept w.

3. What is universal Turing machine? How universal Turing machine works? Explain. [2074 Ashwing, Back]

Please see 2075 Ashwin.

4. Explain halting problem is it solvable problem? Discuss. [2074 Ashwin, Back]

For a given description of Turing machine Tm and input 'w' can we have an algorithm which can determine whether the Turing machine once started its computation for 'w' will eventually halt on input 'w'. And answer is 'no'. It is yet to design such a machine or algorithm which would determine so. And this problem is called Halting problem. In short to determine an arbitrary given Turing machine 'Tm' and input 'w', whether Tm will eventually halt on w.

Though, one way to answer the above question is universal Turing machine. But universal Turing machine might go into infinite loop or take long time to halt which may be in fact loop or sample case of very long computation. So what we can concluded i.e. 'The halting problem is unanswerable or not solvable fill now. And proof for this is described below:

Proof:

$HALT_{Tm} = |(Tm, w)|$. The Turing machine Tm halts on input w is undecidable.

We assume that $HALT_{Tm}$ is decidable and get a contradiction. Let Tm_1 be the Turing machine such that

$T(Tm_1) = HALT_{Tm}$ and Tm_1 half eventually on all (Tm, w). We construct a Turing machine Tm_2 as follows:

(a) For Tm_2, (Tm, w) is an input.

(b) The Tm_1, acts on (Tm, w).

(c) If Tm_1 rejects (Tm_1, w) then Tm_2 rejects (Tm, w).

(d) If Tm_1 accepts (Tm, w), simulate the Turing machine Tm on the input string w until Tm halts.

(e) If Tm has accepted w, Tm_2 accepts (Tm, w) otherwise Tm_2 rejects (Tm, w).

When Tm_1 accepts (Tm, w) (in step d), the Turing machine Tm halts on w. In this case either an accepting q or a state q' such that δ (q', a) is undefined till some symbol a in w is reached. In the first case the (the first alternative of step e) Tm_2 accepts (Tm, w). In the second case (the second alternative of step e) Tm_2 rejects (Tm, w). It follows from the definition of Tm_2 that Tm_2 halts eventually.

Also $T(Tm_2)$ = |(Tm, w)| the Turing machine accepts w.

$= A_{Tm}$

This is a contradiction since A_{Tm} is undecidable.

Note: A_{Tm} is Turing machine which halts and accepts w.

5. Explain encoding technique of universal Turing machine show that the complement of recursive language is recursive. **[2074 Chaitra]**

We adept the following convention of encoding technique of universal Turing machine. Let Tm = (Q, Σ, δ, S) be a Turing machine and k and *l* be smallest integers such that $|2^k| \geq |Q|$ and $2^l \geq |\Sigma| + 2$. Then each state in Q will be represented by 'q' followed by a binary string of length k; each symbol in Σ will be likewise represented as letter 'a' followed by a binary string of *l* bits. The head directions that left and right (R) are included in input tape symbols to justify "+2" term in definition of *l*. For example.

Consider the Turing machine Tm = (Q, Σ, δ, s) where

Q = {s, q, h}

Σ = {#, b, a} and

δ is

States	Symbol	δ
s	a	(q, #)
q	a	(s, R)
s	#	(h, #)

Here, k = 2 and *l* = 3 so that $2^k \geq 3$ (Q) and $2^l \geq 3(\Sigma) + 2$. Then,

the above states symbol can be represented as

States/symbol	Representation
s	q00
q	q01
h	q11
#	a000
b	a001
L	a010
R	a011
a	a100

To prove: Complement of a recursive language is recursive.

For proof see property 1 of recursive and enumerable languages in theory part.

6. What do you mean by Church Turing thesis? State when a problem is said to be decidable and give an example of an undecidable problem. **[2074 Chaitra]**

Church Turing thesis states that, "no computational procedure will be considered as an algorithm unless it can be represented as a Turing machine."

We know that recursive languages are those languages which are accepted by at least one Turing machines and these sets of recursive languages are sub-class of regular sets, called the recursive sets. So, A problem is said to be decidable if its language is recursive.

The post correspondence problem is an undecidable decision problem that was introduced by Emil post in 1946. And it is described as follows:

Let Σ be a finite alphabet and let A and B be two lists of non empty strings over Σ,

with |A| = |B|, i.e.

$A = (w_1, w_2, w_3, \ldots w_k)$

and $B = (x_1, x_2, x_3, \ldots x_k)$

Does there exists a sequence of integers $l_1, l_2, \ldots, l_m$ such that $m \geq 1$ and

$w_{i1} w_{i2} w_{i3} \ldots w_{im} = x_{i1} x_{i2} x x_{i3} \ldots x_{im}$?

For example, suppose A = (a, abaaa, ab) and B = (aaa, ab, b). then the required sequence of integers is 2, 1, 1, 3 giving

abaaa a a ab = abaa aaa b

This example has a solution. But, it will turn out that post's correspondence problem is insolvable or undecidable in general.

7. What is recursive language? Mention its properties. Prove that, "A language is recursive if and only if both it and its complement are recursively enumerable." [2073 Chaitra]

A Formal language is recursive if there exists a Turing machine that, when given a finite sequence of symbols as input, accepts it if it belongs to the language and rejects it otherwise. These languages are also called decidable. Its properties are:

(a) The complement of a recursive language is recursive.

(b) The union of two recursive languages is recursive.

(c) The union of two recursively enumerable languages is recursively enumerable.

(d) If a language L and its complement $\bar{L}$ are both recursively enumerable, then (L and hence $\bar{L}$) is recursive.

(e) If L is a recursive language than $\Sigma^* - L$ is recursive.

For proof, please see property 4 of properties of recursive and recursive enumerable language in theory part.

8. Explain recursive and recursively enumerable languages with suitable example of each language. [2073 Shrawan, Back]

A language L is said to be recursive if there exists a Turing machine 'Tm' that accepts every strings belonging to language L (i.e. $w \in L$) and rejects all the string that doesn't belong to language L (i.e. $w \notin L$). The Turing machine will halt every time and given an answer (accepted or rejected) for each and every input string. Therefore, these language are also called decidable. An example of recursive language is the language that accepts all the strings over alphabet {a, b} that contains equal number of a's and b's.

A language L is said to be recursively enumerable language if there exists a Turing machine 'Tm' that only accepts the input strings 'w' which belongs to language L (i.e. $w \in L$). But it may or may not halt for all input strings which are not in language L. Therefore, these languages are also called undecidable language and are considered as the superset of recursive language. The examples of recursively enumerable languages are the languages for which no Turing machine exists for its solution. Like Halting problem, post correspondence problem and so on. For example, given two lists of non-empty strings over Σ, A and B. i.e.

$A = (w_1, w_2, w_3, \ldots w_k)$

$B = (x_1, x_2, x_3, \ldots x_k)$

Does there exists a sequence of integers $l_1, l_2, \ldots l_m$ such that $m \geq 1$ and $w_{i1} w_{i2} w_{i3} \ldots w_{im} = x_{i1} x_{i2} x_{i3} \ldots x_{im}$?

Thus, above language which describe post correspondence problem is a example of recursively enumerable language.

9. State church Turing thesis. What is a recursive language? [2072 Chaitra]

Church Turing thesis states that "No computational procedure will be considered an algorithm unless it can be represented as a Turing machine in other words, "what can be computed by a general purpose digital computer or human can also, i.e. computed by Turing machine. Unfortunately this thesis can't be a theorem in mathematics because ideas such as "can ever be defined by human" are not part of any known mathematics. There are no axioms that deal with people.

For definition of recursive language, please see 2073 Shrawan.

10. Show that if a language L and its complement both are recursively enumerable, then L and its complement is recursive. Explain the halting problem. [2072 Chaitra]

For proof, please see property (d) of properties of recursive and recursively enumerable languages in theory part.

For explanation of halting problem please see theory part.

11. What is an "Algorithm" according to Church Turing thesis? Why is it called thesis and not a theorem? Prove that if a language 'L' and its complement $\bar{L}$ are recursively enumerable, then L is recursive [2071 Chaitra]

Whatever can be computed or can be represented by Turing machine can be considered as an "Algorithm" according to church Turing

thesis. This statement is called church Turing thesis because Alonzo church (1936), gave many sophisticated reasons for believing it. Church's original statement was slightly different because his thesis was presented slightly before the Turing invented his machines.

Unfortunately, church Truing thesis can't be a theorem in mathematics because ideas such as "can ever be defined by humans" and" algorithm that people can taught to perform" aren't part of any known mathematics. There are no axioms that deal with people.

For proof, please see property (d) of properties of recursive and recursively enumerable languages in theory part.

12. "Turing machines is believed to be the ultimate calculating mechanism, "elaborate with the help of church Turing thesis. How halting problems suffer the computational procedures? Explain with suitable example. [2070 Chaitra]

How can you define computation? or 'what is computation?', these are the questions that arose during the infancy of the computability theory. Several model of the computations were suggested to answer this question. And it started with Godel, who was trying to understand to which proof systems his incompleteness theorem applies, came up the formalism of general recursive functions and then church came up with the λ calculus as an attempt at paradox free foundations for mathematics. And finally it ended with Alan Turing (who was the student of church), he was motivated by himself by a problem of Hilbert, who asked for a "purely mechanical process" for determining the truth value of a given mathematical statement.

At the time, Turing's attempt as defining computability seemed as the most satisfactory. It eventually turned out that all models of computation described above are equivalent, they all described the same notion of computability. For historical reasons, the Turing machine model came out as the most canonical way of defining computability mechanisms. The Turing machine model is also very rudimentary and so easy to work with, compared to many others including the ones listed above.

And then, it is believed that there are no functions that can be defined by humans, whose calculation can be described by any well-defined mathematical algorithm that people can be taught to

perform, that can't be computed by Turing machine. Therefore, this gave rise to a Church Turing thesis which states that "no computational procedure will be considered an algorithm unless it can be represented by a Turing machine. In other words, "what can be computed by humans, digital computer or algorithm can also be computed by Turing machine and hence, Turing machine is believed to be the ultimate calculating mechanisms.

Halting problems suffer the computational procedures in the way that we assume this problem is decidable or solvable, but eventually end up with the fact that it is not solvable (or undecidable). For example,

The blank-tape halting problem

Given a Tm, decide whether or not Tm halts on a blank tape.

Steps to prove halting problem

Step I: We reduce the halting problem to the blank tape halting problem.

Step II: Assume that a Truing machine Tm, A solves the blank tape problem.

Step III: Construct a Turing that solves the halting problem.

1. Given (M, w) construct M w that, given a blank tape.
   (a) write w onto the tape.
   (b) puts itself in the configuration q_0W.
   (c) behaves like M thereafter.
2. If (A (Mw) = "yes"
   return "yes"; M halts on w
   else
   return "no"; M doesn't halt on w.

Step IV: But, the halting problem is undecidable.

Step V: Our assumption that A exists (Turing machine that halts on input string ∈ (empty string) is false.

13 What is universal Turing machine? Explain with example, how universal Turing machine works?

Please see 2075 Ashwin, Back]

14. What do you mean by church Turing thesis? Explain Turing recognizable languages and Turing decidable language with suitable examples. [2071 Shrawan, Back]

For church Turing thesis, please see theory part.

A language L is recognizable languages (also called recursively enumerable) if there exists a Turing machine that accepts every string of the language and doesn't accepts strings (i.e. may go into infinite loop or will not halt) that are not in language. For example. The languages of the halting problem, post correspondence problem, etc.

A language L is decidable language (or recursive) if there exists a Turing machine that accepts every strings of language and rejects every string over the same alphabet that is not in the language. For example, the language L that accepts all strings containing equal number of a's and b's over alphabet $\Sigma = \{a, b\}^*$.

15. Explain the church Turing thesis. Show that the halting problem is undecidable. [2072 Kartik, Back]

Please see [2075, Ashwin, Back].

◆◆◆

Chapter - 6

Computational Complexity

Introduction

The branch of theory of computation that deals with the inherent properties of computation in the field of computer science is called computational complexity. There are basically two types of complexities and they are:

1. Time complexity

It is a measure of how long a computation takes to execute. As far as Turing machine is concerned, this could be measured as the number of moves which are required to perform a computation. In the case of a digital computer, this could be measured as the number of machine cycles which are required for the computation. Time complexities are classified under following terms:

(a) **Polynomial time complexity/algorithm:** A polynomial time complexity is an algorithm whose execution time is either given by a polynomial on the size of the input, or can be bounded by such a polynomial. Problems which can be solved by a polynomial-time algorithm are called 'tractable' problems. For example, in order to find the largest element in any array requires a single pass through the array, so the algorithm which does this is of O(n), or it is a 'linear time' algorithm. Sorting algorithms take O(n log n) or $O(n^2)$ time. Bubble short takes O(n) time in the average and worst case. Heap sort, merge sort, insert sort, etc. all take O(n log n) time where n is the size of the array.

Find time complexity of following machine.

$F_1(x) = 5n^3 + 3n + 1$

The time complexity of this machine would be term containing the highest order neglecting the coefficient i.e. $O(n^3)$ time where n is the site of input.

(b) **Exponential time complexity:** A exponential time complexity is an algorithm whose executed time is expressed in terms of exponential function. For example the time complexity of INTEGER BIN PACKING is $O(2^n)$.

Space complexity

It is a measure of how much storage is required for a computation. In the case of a Turing Machine, the obvious measure is the number of squares used, for a digital computer, the number of bytes used.

Polynomially bounded Turing machine.

A turning machine TM = (Q, Σ δ, S) is said to be polynomially bounded turning machine if there is a polynomial P(n) such that machine always halts after almost P(n) steps, where n is the length of the input.

In short, a turning Turing machine that has polynomial time complexity for its deterministic computation is said to be polynomially bounded Turing machine. In this type of machine, it always halts and either accepts or rejects the input for given language.

Class P

Class P includes all the problems or languages that have algorithms to be solved in polynomial time complexity by deterministic Turing machine.

Or, a language L is said to be in class P if there exists a polynomial bounded turning machine (deterministic) such that TM is of time complexity P(n) for some polynomials P and TM accepts L. this language L is also called polynomial decidable.

Or, P is the set of problems that can be solved in deterministic polynomial time. That means for a problem with inputs of size N, there must be some way to solve problem in F(N) steps for some polynomial F.

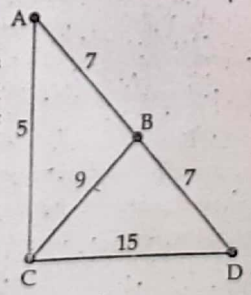
Example of class - P

1. **Minimum weight spanning tree (MWST)**

   Given a weighted graph G, final the minimum weight spanning tree.

   In other words, convert the given graph into a tree, that includes all the nodes of the original graph and minimizes the summation of weights of the edges in the resulting tree.

   This problem can be solved using Kruskal's Algorithm whose time complexity is $O(n^2)$ i.e. polynomial.



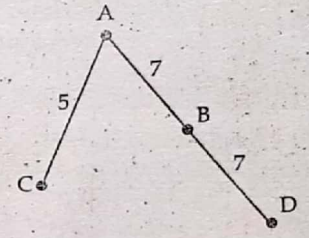
Steps:

(i) Sort all the edges in non-decreasing order of their weight.

(ii) Pick the smallest edge check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge else discard it.

(iii) Repeat step 2 until there are (v – 1) edges in spanning three.

(iv) Using above steps, the answer to this problem is



The MWST problem belongs to the P class of problems, since there is an algorithm that solves it in polynomial time.

2. **Integer partition problem**

   Given a set of integers, determine whether a given set can be partitioned into two subsets such that sum of elements in both subsets is same for example, A = {1, 5, 11, 5}

Steps:

(i) Calculate sum of the array. If sum is odd, there can't be two subsets with equal sum, so return false.

(ii) If sum of trray elements is even, calculate sum/2 and a subset of array with sum equal to sum/2.

Using dynamic programming, the time complexity of this problem comes to be O(sum × n). i.e. polynomial. Let. B and C be two subsets of A.

B = {1, 5, 5}

C {11} is the solution

Since, tie complexity is polynomial, if falls under class P.

There are other examples of class P like Eulerian and Hamilton Graph problem,optimization problem, equivalenceof finite automataand so on.

Class NP

'A language L is in class NP if there is polynomially bounded non-deterministic Turing machine TM such that TM is of time complexity P(n) for some polynomial P (n) and TM accepts L.'

In other words class NP includes all the problems that have algorithms to solve in polynomial time complexity by non-deterministic Turing machines.

NP is the set of problems you can solve in non-deterministic polynomial time. That means for a problem with inputs of size N, there must be some way to solve the problem in F(N) steps for some polynomial F just as before. In NP, however the program is allowed to make lucky guesses, though it must prove the solution is correct. The standard format for a program in NP is:

(i) Guess the answer.

(ii) Verify that answer is correct in polynomial time.

For example, factoring is in NP. Suppose you have a number A that you want to break into two factors. The NP program is

- Guess factors P and Q.
- Multiply P times Q and verify that the result is A.

This takes only two non-deterministic steps so the problem is in NP. Note that all problems in P are also in NP.

Other example of NP

1. **Travelling salesman problem (TSP)**

   Given a graph in which the nodes cities are connected by directed cities (routes), where the weight of an edge is the distance between two cities, edge is the distance between two cities, the problems is the find a path that visits each city once, returns to the starting city, and minimizes the distance travelled. The only known solution that guarantees the shortest path, requires a solution time that grows exponentially with the problem size (i.e. number of cities). This is an example of an NP complete problem for which no known efficient (i.e. polynomial time) algorithm exists.

2. **Hamiltonian cycle problem (HCP)**

   A Hamiltonian circuit (HC) is a cyclic ordering of a set of nodes such that there is an edge connecting every pair of nodes in the graph in order. The cycle condition ensures that the circuit is closed and the requirement that all the nodes are included (with no repeats) ensures that the circuit doesn't cross over itself, and passes through every node. The problem is to find if a HC exists for a given graph, thus determining if a given graph has a cycle visiting each vertex exactly once. This is considered to be the Hamiltonian cycle problem.

3. **Linear programming.**

   We have on hand X amount of butter, Y amount of flour, Z eggs, etc. we have cookie recipies that use varying amount of these ingredients. Different kinds of cookies bring different prices. What mix of cookes should we make in order to maximize profits?

   Other examples of NP problems includes the graph colouring problem, independent set problem, the Bin packing problem, the satisfiability problem (SAT), the 3 satisfiability problem (3 SAT), the maximum clique problem and others.

Does P = NP?

No one has ever proven that they are equal and no one has ever proven they are not either. So, this question still remains unanswered. No one has ever found a deterministic polynomial time algorithm for any of these problems (i.e. NP problems) so that we can show P = NP. There is a large class of NP complete problems that we can show are equally difficult. For example if you find a deterministic solution for the traveling salesman problem (finding the shortest path that visits a set of cities and returns to the start), we can also find one for the SAT problem (find an assignment of values to Boolean variables in a logical statement to make it true). Despite all efforts, however no one has ever shown that an NP class problem is in P.

On the other hand, no one has ever succeeded in proving that no deterministic polynomial time algorithm exists, either for class NP problem. Therefore, we can't also say that P ≠ NP. However, many researcher believe that N < > NP but no one knows for-sure.

NP problems are further classified as

NP Hard: A problem is NP-hard if an algorithm for solving it can be translated into one for solving any other NP-problems (non deterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem" although it might, in fact, be harder.

The NP-hard class is a superset containing the NP-complete class (discussed later on this chapter). This class is potentially harder to solve than NP-complete problems because although if any NP-complete problem is intractable then all NP-hard problems are intractable, the reverse is not true.

NP Complete: Any NP problem can be considered as NP-complete problem if all the other NP problem are reducible to it in polynomial time complexity.

OR

If a problem is NP and all other NP problems are polynomial time reducible to it, the problem is NP-complete.

The most important property of NP complete is the so called polynomial time reducibility. Any NP complete problem can be transformed into any other NP complete problem in polynomial time. Thus, if it could be proved that any NP complete problem is formally intractable, all such problems would have proved intractable and vice-versa.

Thus, finding an efficient algorithm for any NP complete problem implies that an efficient algorithm can be found for all such problems. It is not known whether any polynomial-time algorithms will ever be found for NP complete problems and determining whether these problems are tractable or intractable remains one of the important questions in theoretical computer science. Examples of NP problems include the Hamiltonian cycle, travelling salesman problems, vertex cover problem, Boolean satisfiability and so on.

Boolean satisfiability

Assume we have n Boolean variables viz. A, B, C, ... and an expression in the propositional calculus i.e. we can use and ($\wedge$), or ($\vee$) and not ($\sim$) to form the expression. Is there an assignment of truth values to the variables (for example, A – true, B = true, C = false), that will make the expression true.

Here is a non-deterministic algorithm to solve the problem for each Boolean variable, assign if proper truth value. This is linear algorithm. We can find a deterministic algorithm for this problem. Effectively, the idea is to set up a systematic procedure to try every possible assignment of truth values to variables the algorithm terminates when a satisfactory solution is found, or when all 2^n possible assignments have been tried. Again, the deterministic solution requires exponential time.

Example, check whether the Boolean formula

$$\phi = (\bar{x}) \wedge y) \vee (x \wedge \bar{z})$$

is satisfiable or not.

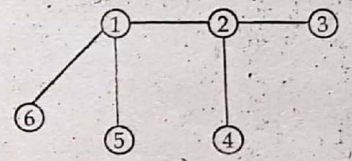
Solution: When x = 0, y = 1 and z = 0, we have

Note: We picked values randomly i.e. no-deterministically.

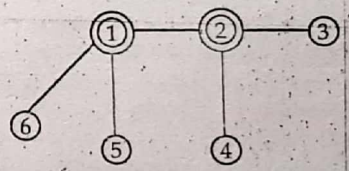
$$\phi = (1 \wedge 1) \vee (0 \wedge 1)$$
$$= 1 \vee 0$$
$$= 1$$

So, it is satisfiable.

Vertex cover problem: Given a undirected graph, the task is to find minimum number of vertices that can cover or lined with all the remaining vertices in the graph.



A brute-force algorithm to find a vertex cover in a graph is to list all subsets of vertices V and check each one to see if it forms a vertex cover. Like here, if we mark vertices 1 and 2, it covers all the remaining vertices i.e. {3, 4, 5, 6}.



It is linear algorithm.

Difference between NP hard and NP-complete

NP hard	NP complete
(a) If an even harder is reducable to all the problems in NP set (at least as hard as any NP problem then that problem is called NP hard.	(a) If an NP hard is inside the set of NP problems then that is NP complete.
(b) NP hard problems may not belong to NP class.	(b) NP complete problem always belongs to both NP and NP hard.
(c) It is more harder to solve than NP complete.	(c) It is less harder to solve than NP hard.
(d) It is the superset of NP complete problems.	(d) It is the subset of NP hard problems.
(e) For example; the halting problem the subset sum problem, etc.	(e) For example; travelling Salesman problem, vertex cover problem, Hamiltonian cycle problem and so on.

Exam Solution

1. Explain NP hard and NP complete problems with reference to polynomial time redaction. [2075 Ashwin, Back]

A problem is NP hard if an algorithm for solving it can be translated into non-deterministic polynomial time reducible algorithm (or NP-complete problem) in general, a problem which is harder than any NP complete problem though it is reducible to polynomial time complexity of NP complete problem is called NP Hard. NP hard therefore means "at least as hard as any NP problem," although it might in fact, be harder. A more precise specification is: A problem H is NP hard when every problem L in NP can be reduced in polynomial time to H. Assume, a solution for H takes 1 unit time, we can use H' solution to solve L in NP in polynomial algorithm to solve any NP hard problem would give polynomial algorithms for all the problems in NP, which is unlikely as many of them are considered difficult. Example of NP hard halting problem, the subset sum problem, etc.

If a problem is NP and all the other NP problems are polynomial time reducible to it, then the problem is NP complete. The most important property of NP complete problems is the so called polynomial time reducibility.

Any NP complete problem can be transformed into any other NP complete problem in polynomial time. Thus, if it could be proved that any NP-complete problem is formally intractable, all such problems would have proved intractable, and vice-versa. While no profs of intractability has been found, no polynomial algorithms have ever been found that solve any of these problems and because of the breadth of the class of problems it is widely believed that no such algorithms exist.

Thus, finding an efficient algorithm for any NP complete problem implies that an efficient algorithm can be found for all such problems, since any problems belonging to this class can be recast into any other member of the class. It is not known whether any polynomial time algorithms will ever be found for NP complete problems. Examples of NP complete problems: Hamiltonian cycle, travelling salesman, Boolean satisfiability, vertex-cover problem, etc.

2. Explain P and NP class of problems [2074 Chaitra]

Class P includes all the problems or languages that have algorithms to be solved in polynomial time complexity by deterministic Turing machine or a language L is said to be in class P if there exists a polynomially bounded turning machine TM (deterministic) such that TM is of time complexity P(n) for same polynomial P and TM accepts L. the language L of this class P is also called polynomial decidable. And the problems of this class P are also tractable problems

it means that for every problem in class P. It is guaranteed that there exists a polynomial time algorithm for its solution in F(N) steps for some polynomial F where N is the size of input. Class P is known to contain many natural problems, including the decision versions of linear programming, calculating the greatest common divisor and finding a maximum matching. Languages in P class are also closed under reversal, intersection, union, concatenation, kleene closure, inverse homomorphism and complementation other examples of class P are minimum weight spanning tree, the partition problem.

A language L is in NP class if there is polynomially bounded non-deterministic Turing machine TM is of time complexity P(n) for some polynomial P and TM accepts L. In other words, class NP includes all the problems that have algorithms to solve in polynomial time complexity by non-deterministic Turing machines. NP problems are closed under union, intersection, concatenation, kleene star and reversal. It is not known whether NP is closed under complement (this question is so called NP versus co-NP question). An important notion in this context is the set of NP complete decision problems. Although the definition of P and NP seems similar, but there is a vast difference between them. When problem L is in P, the number of moves to test whether any string of length n is less than or equal to P(n) when problem L is in NP the number of moves for testing is less than or equal to n only for string accepted by TM. Thus in case of NP, the bound P(n) for the number of moves in useful only when we are able to find a string W in L. Examples of NP problems. Hamiltonian cycle problem, travelling salesman problem (TSP), vertex cover problem and so on.

3. What are two factors affecting the computational complexity of a problem? Explain class NP with suitable examples. [2074 Ashwin, Back]

Two factors that affect the computational complexity of a problem are.

(i) **Time complexity:** Time required by a program or algorithm for its execution. It is generally expressed in O(time0-function). For example, time complexity of bubble sort is O(n) for best case and $O(n^2)$ for worst case.

(ii) **Space complexity:** Space required by a program or algorithm for its execution. For Turing machine, it is number of squares used in computation whereas number of bytes used in case of digital computer.

A language L is in NP class if there is polynomially bounded non-deterministic Turing machine is of time complexity P(n) for some polynomial P and TM accepts L. Class NP problems are closed under union, intersection, concatenation, kleene star and reversal. It is not known whether class NP is closed under complement. An

important notion in this context is the set of NP complete problems. Class NP problems are also called intractable problems or undeadable problems. The difference between class P and class NP is that when problem L is in P, the number of moves to test whether any string of length n is less than or equal to polynomial time P(n). But when L is in NP, the number of moves for testing is less than or equal to n only for string accepted by TM. Examples of NP problems are travelling salesman problem (TSP), vertex cover problem, Hamiltonian cycle problem, the Bin-packing problem, the graph colouring problem etc.

Travelling salesman problem

Given a graph in which the nodes (cities) are connected by direct edges (routes), where the weight of an edge is the distance between two cities, the problem is to find a path that visits each city once, returns to starting city and minimizes the distance travelled. The only known solution that guarantees the shortest path, requires a solution time that grows exponentially with the problem size (i.e. number of cities). This is an example of an NP complete problem, for which no known efficient (i.e. polynomial time) algorithm exists.

4. Define computational complexity and polynomial time deduction. Also, explain NP hard and NP-complete problems [2073 Chaitra]

The branch of theory of computation that deals with the inherent properties of computation in the field of computer science is called compulational complexity. There are two types of complexities and they are time complexity and space complexity.

Polynomial time execution time is either given by a polynomial on the size of input or can be bounded by such a polynomial. Polynomial time reduction is always expressed in O(polynomial time function) like $O(n)$, $O(n^2)$, $O(n^3)$ and so on.

For explanation of NP hard and NP complete problems, please see 2075 Ashwin, Back.

5. Explain class P and NP problems with examples what is NP-complete problem? [2073 Shrawan, Back]

For explanation of class P and class NP problems, please see 2074 Chaitra.

Any NP problem is Np complete problem if all other NP problems are polynomial time reducible to it. For example, travelling salesman problem (TSP), the Bin-packing problem etc.

6. Write short notes on:

(a) Computational complexity. [2072 Chaitra]

(b) NP hard and NP complete problems.

Please see theory part and 2075 Ashwin, Back.

7. Define class P and class NP problems with example. How do they relate to NP complete problems?

For class P and class NP problems, please see 2074 Chaitra.

We could relate class P to NP complete problems if we could show P = NP. But, the problem is no one has yet been able to prove their 'equality' or not equality. So, P = NP remains one of the most important questions in theoretical computer science as no one has even found a deterministic polynomial time algorithm for NP problems and no one has ever succeeded in proving that no deterministic polynomial time algorithm for NP problems, either. So, relation between P and NP complete problem is still a mystery.

Whereas, NP and NP complete problems could be related in such a way that any NP problem will be NP complete problem if all the other NP problems are polynomial time reducible to it. Thus, finding an efficient algorithm for any NP complete problem implies that an efficient algorithm can be found for all such NP problems (a deterministic one). When an NP complete problem must be solved, one approach is to use a polynomial algorithm to approximate the solution, the answer thus obtained will not necessarily be optimal but will be reasonably close.

8. Explain what is the class P. Describe the travelling salesman problem. [2072 Kartik, Back]

For explaination of class P, please see 2072 Chaitra.

For description of travelling salesman problem please see 2074 Ashwin Back.

9. With reference to polynomial time reducibility, explain NP hard and NP complete problems. [2070 Chaitra]

Please see 2075 Ashwin Back.

10. Why computational complexity analysis is required? Define class NP and explain how travelling salesman problem (TSP) is class NP problem. [2071 Shrawan, Back]

Computational complexity analysis is the amount of time, storage and / or other sources necessary to execute problems / algorithms. It provides theoretical estimates for the resources needed by an algorithm which solves a given computational problem which in turn provides an insight into reasonable directions of search for efficient algorithms.

For explanation of NP class and description of travelling salesman problem please see 2074 Ashwin, Back.

TSP is class NP problem because the only solution that guarantees the shortest path for TSP requires a solution that grows exponentially with the problem size (i.e. number of cities and we know, NP problems have exponential time reduction algorithm or problem.

NOTE

Bibliography

- "*An Introduction to Automata Theory and Formal Languages*", Adesh K Pandey.
- "*Theory of Automata of Automata, Formal Languages and Computation*", S.P. Eugene Xavier.
- "*Introduction to Formal Languages, Automata Theory and Computation*", Kamala Krithivasan, Rama R.
- "*Elements of the Theory of Computation*", Harry R. Lewis, Christos H. Papadimitriou.
- Wikipedia
- http://mathinsingth.org
- www.cs.wcuea.org
- http://stackoverflow.com
- http://math.stackexchange.com
- http://www.youtube.com