CSE Learning Hub

Theory of Computation

(3160704)

DEGREE

COMPUTER Engineering | SEM: 6

Notes By: Harsh Porwal

Chapter :1

1. **Sets Theory =>** (Subset, Empty Set, Null String, Equal Set, Power Set)
2. **Operation On Set =>** (Union, Intersection, Difference, Complement)
3. Cartesian Product
4. **Functions =>** (Composition of Functions, Inverse of Functions, One-to-One Function {Injective}, Onto Function {Surjective}, Bijection)
5. **Logical Statement =>** (Negation, Conjunction {And}, Disjunction {OR}, Implication, If and only If)
6. **Relation =>** (Reflexive Relation, Irreflexive Relation, Symmetric Relation, Asymmetric Relation, Antisymmetric Relation, Transitive Relation, Equivalence Relation, Closure Relation)
7. **Examples:**
    a. Let A= {1,2,3,4,5,6} and R be a relation on A such that aRb iff (a) is a multiple of (b). Write R and check if relation is reflexive, symmetric, asymmetric, transitive?
    b. In each case a relation on the set {1,2,3} is given check whether it is reflexive, symmetric and transitive?
        i. R= {(1,3), (3,1), (2,1)}
        ii. R= {(1,1), (2,2), (3,3), (1,2)}
        iii. R= Φ
    c. Define equivalence relation. A relation on set {1,2,3} is given as R= {(a, b) | a-b is even no.}. Check weather R is equivalence or not.
8. **String =>** (Operation on String >> Concatenation, Transpose, Palindrome)
    a. Ex: Consider the string x= 110 and y=0110 then find. (i) xy (ii) yx (iii) x^2 (iv) Ey
9. **Inductive Proofs**
    a. Prove that: $1+2+3+\ldots+n = \frac{n(n\times1)}{2}$
    b. Prove that: $1^2+2^2+3^2+\ldots+n^2 = \frac{n(n+1)(2n+1)}{6}$
    c. Prove that: $1+3+5+\ldots+r = n^2$ for all $n>0$ Where, $r = (2n-1)$
    d. Prove that: $\sum_{i=1} \frac{1}{i(i+1)} = \frac{n}{(n+1)}$

Chapter :2

1. **Regular Expression and Language** (With Example).
2. **RE to FA conversion** (With Example).
3. **DFA (Deterministic Finite Automata)** (With Example).
4. **NFA (Non-Deterministic Finite Automata)** (With Example).
    a. Convert NFA to equivalent DFA.
5. **Minimization of Finite Automata.**
6. **Moore and Mealy Machine** (With Example).
7. **Find Regular Expression from DFA and NFA.**
8. **NFA to DFA Conversion.**
9. **Define Regular Language and Finite Automata.**
10. **Recursive definition of NFA.**
11. **Explain Pumping lemma and its application.**
12. **Explain Kleene's theorem Part – 1.**
13. **What is Finite State Machine?**

Chapter :3

1. **Definition of (CFG) Context free (Type:2) Grammar.**
    a. Construction of CFG.
2. **Regular Grammar.**
3. **Left most and Right most Examples.**
4. **Ambiguous and Unambiguous Grammar.**
5. **CNF** (Chomsky's Normal Form).
    a. Conversion of CFG to CNF.

Chapter :4

1. **What is PDA?**
2. **What is deterministic PDA?**
3. **Difference between Top-Down and Bottom-Up parsing.**
4. **PDA to CFG conversion steps.**

Chapter :5

1. Explain Turing Machine.
2. Model of Computation of Turing Machine.
3. Explain Universal Turing Machine.
4. Church's Turing thesis.
5. Recursive and Recursive Enumerable Language.
6. Context sensitive language.
7. Chomsky's Hierarchy.

Chapter :6

1. Partial, Total and Constant functions.
2. Primitive Recursive Function.

Chapter :7

1. Undecidable Problems About Turing Machine.
    a. Halting Problem.
    b. Post's Correspondence Problem.
2. Compare Push down automata and Turing Machine.
3. Enlist limitation of Turing Machine.
4. What is P and NP Class?
5. Difference between P and NP class.

Camlin Page
Date / /

Ch: 1 Review of Mathematical Theory

☆ Sets

→ Set is defined as collection of objects. These objects are called elements of the set. All the elements are enclosed within curly brackets and seperated by comma.

→ If a is and element of A then we can say, $a \in A$

→ If a is not an element of A then we can say, $a \notin A$.

(i) Listing method

→ Set of element which are less than 5.

$A = \{0, 1, 2, 3, 4\}$

(ii) Describing properties

→ Set of vowels.

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

(iii) Recursion method

→ $A = \{x / x \text{ is square of } n\}$ where $n \leq 10$

$A = \{0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100\}$

→ $|A|$ denotes length of set A. numbers of element in set A.

Ex. $A = \{1, 2, 3, 4\}$

$|A| = 4$

Camlin Page
Date / /

* **Subset :** The subset of A is called subset of B if every element of set A is present in set B. But reverse is not true. Denoted by $A \subseteq B$.

* **Empty set :** The set having no element in it is called empty set. Denoted by $A = \{\}$ and it can be written as $\emptyset$.

* **Null String :** The null element is denoted by ε or $\wedge$ character. Null element means no value character. But ε does mean $\emptyset$.

* **Equal set :** The two sets are said to be equal ($A = B$) if $A \subseteq B$ and $B \subseteq A$. Means every element of A in B and every elements of B in A.

* **Power Set :** It is set of all subsets, empty set and the original set itself. [Total element $= 2^n$, $2^3 = 8$]

Ex. $A = \{1, 2, 3\}$

Power set : $Q = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

Camlin Page
Date / /

☆ Operation on set

(i) **Union** : If two sets A and B are given, then union of A and B is equal to the set that contains all the elements present in A and B.

Ex. $A = \{1, 2, 3\}$, $B = \{3, 4, 5\}$ then,
$A \cup B = \{1, 2, 3, 4, 5\}$

(ii) **Intersection** : If two sets A and B are given, then intersection of A and B is equal to set which consist of element common to both A and B.

Ex. $A = \{1, 2, 3\}$ and $B = \{3, 4, 5\}$ then
$A \cap B = \{3\}$

(iii) **Difference** : If two sets A and B are given, then difference of A and B is equal to set which consist of elements that present in A but not in B.

Ex. $A = \{1, 2, 3\}$ and $B = \{3, 4, 5\}$ then
$A - B = \{1, 2\}$

(iv) **Complement** : If U is an universal set and A is any subset of U then the complement of A is set of all element of the set U apart from elements of set A.

Ex. $U = \{10, 20, 30, 40\}$
$A = \{10, 30\}$
then, $\overline{A} = U - A$
$= \{20, 40\}$

Camlin Page
Date / /

☆ Cartesian Product

→ Cartesian product of two set A and B is a set of all possible ordered pairs whose first component is member of A and second component is member of B.

Ex. $A = \{a, b\}$ and $B = \{0, 1, 2\}$

→ $A \times B = \{(a,0), (a,1), (a,2), (b,0), (b,1), (b,2)\}$

☆ Functions

→ Function can be defined as the relationship between two sets. That means using function we can use one element of one set to some other elements of another set.

→ Ex. $D = \{1, 2, 3\}$ and $f(x) = x3$

then, $R = \{f(1), f(2), f(3)\}$
$= \{1, 8, 27\}$

If we take cartesian product of D and R.
$F = \{(1,1), (2,8), (3,27)\}$

Domain (D) = {1, 2, 3}
Range (R) = {1, 8, 27}
Function (F) = {(1,1), (2,8), (3,27)}

Camlin Page
Date / /

* Composition of functions

→ It is a function in which result of one function is given as input to another function. It is denoted by gof or fog.

Ex. $f(x) = x+2$, $g(x) = 2x+1$

→ $fog = f(g(x))$
$= 2x+2+1$
$= 2x+3$

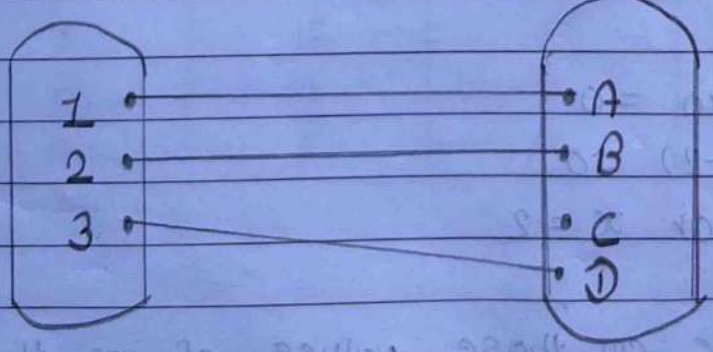
→ $gof = g(f(x))$
$= 2(x+2)+1$
$= 2x+5$

* Inverse of functions

→ Inverse of function is a function which is reversal of whatever the original function does as input. It is denoted as $f^{-1}(x)$.

* One-to-One (Injective)

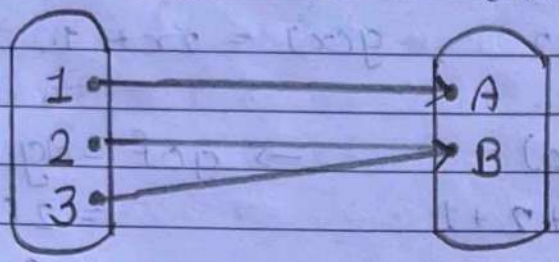
→ A function $f: A \to B$ is said to be one to one mapping when function that maps distinct element of its domain to distinct elements of its co-domain.



Camlin Page
Date / /

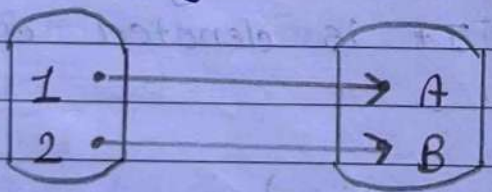
* Onto (Surjective)

→ A function $f: A \to B$ said to be onto if every element of B, there is atleast one or more element matching with A. then it is called onto function.



* Bijection

→ A function $f: A \to B$ said to be bijection if every elements of A can be mapped with exactly one element of B.



Ex. Determine domain: $y = \frac{x^2 + x - 5}{x^2 + 3x - 10}$

→ The only care has to be taken is that, function should not produce divide by zero error. hence,

$$x^2 + 3x - 10 = 0$$
$$(x+5)(x-2) = 0$$
$$x = -5 \quad \text{or} \quad x = 2$$

→ Domain will be all those values of x that are not equal to -5 or 2.

Camlin Page
Date / /

☆ Logical Statment

-> Logic is a study of reasoning.
In the study of formal language mathemeticcal logic is represented by propositions.

-> The propositions and sentence are declarative sentences, which can be either true or false but not both.
We can join propositions by 'and', 'but', 'if', 'or'. These are called connectives.

(i) Negation

-> Denoted by [¬, or ~ or having bar on letter.]

S	¬S
T	F
F	T

(ii) Conjuction (AND)

-> Denoted by [^]

A	B	A ^ B
T	T	T
T	F	F
F	T	F
F	F	F

(iii) Disjunction (OR)

-> Denoted by [$\vee$].

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

(iv) Implication

-> Denoted by [$A \rightarrow B$]

A	B	$A \rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

(v) if and only if

-> Denoted by [$A \leftrightarrow B$]

A	B	$A \leftrightarrow B$
T	T	T
T	F	F
F	T	F
F	F	T

Camlin Page
Date / /

Ex. Justify following logic formula.

$P \rightarrow (\neg P \vee \neg Q)$

→

P	Q	¬P	¬Q	¬P ∨ ¬Q	P → (¬P ∨ ¬Q)
T	T	F	F	F	F
T	F	F	T	T	T
F	T	T	F	T	T
F	F	T	T	T	T

☆ Relations

→ The relation R is a collection for the set S which represent pair of element.

→ (a, b) is in R. We can represent their relation as a R b. The first component of each pair is chosen from set called domain and second component chosen from Range.

* Reflexive Relation (iRi for all i in S)

A relation R on set A is said to be a reflexive if $(a,a) \in R$ for every $a \in A$.

Ex. $A = \{1, 2, 3\}$

$R = \{(1,1), (2,2), (3,3)\}$

* Irreflexive Relation

A relation R on set A is said to be a irreflexive if $(a,a) \notin R$ for every $a \in A$.

Ex. $A = \{1, 2, 3\}$

$R = \{(1,2), (1,3), (2,3), (2,1), (3,1)\}$

Camlin Page
Date / /

* Symmetric Relation (iRj implies jRi)

A relation R on set A is said to be Symmetric if $(a,b) \in R \leftrightarrow (b,a) \in R$.

Ex. $A = \{1, 2, 3\}$

$R = \{(1,1), (1,2), (2,1), (2,3), (3,2)\}$

* Asymmetric Relation

A relation R on set A is said to be Asymmetric if $(a,b) \in R$ but $(b,a) \notin R$.

Ex. $A = \{1, 2, 3\}$

$R = \{(1,2), (1,3), (2,3)\}$

* Antisymmetric Relation

A Relation R on set A is antisymmetric if $(a,b) \in R$ and $(b,a) \in R$ then $a = b$

Ex. $A = \{1, 2, 3\}$

$R = \{(1,1), (2,2), (3,3)\}$

* Transistive Relation (iRj and jRk imply iRk)

A relation R on set A is said to be transistive if $(a,b) \in R$ and $(b,c) \in R$ then $(a,c) \in R$.

Ex. $A = \{1, 2, 3\}$

$R = \{(1,2), (2,3), (1,3)\}$

* Equivalance Relation

A relation is said to be equivalence relation when it is reflexive, symmetric and transistive.

Camlin Page
Date / /

* Closure Relation

-> Sometimes when relation R is given, it may not be reflexive or transistive.

By adding some pair we can make relation as reflexive and transistive.

Ex. Let $A = \{1, 2, 3, 4, 5, 6\}$ and R be a relation on A such that aRb iff (a) is a multiple of (b). Write R. and check if relation (i) reflexive (ii) Symmetric (iii) Asymmetric (iv) transistive

-> $R = \{(a, b) : a = i * b\}$ where $1 \leq i \leq 6$

$R = \{(1,1), (2,2), (3,3), (4,4), (5,5), (6,6), (2,1), (4,2), (6,2), (6,3)\}$

(i) It is reflexive as $(a,a) \in R$

(ii) It is not symmetric as $(a,b) \in R$ but $(b,a) \notin R$.

(iii) It is asymmetric because $(a,b) \in R$ but $(b,a) \notin R$.

(iv) It is transistive because $(a,b) \in R$ and $(b,c) \in R$ then $(a,c) \in R$.

Camlin Page
Date / /

Ex. In each case a relation on the set $\{1, 2, 3\}$ is given check three properties reflexive, symmetric and transitive.

(a) $R = \{(1,3), (3,1), (2,2)\}$

(b) $R = \{(1,1), (2,2), (3,3), (1,2)\}$

(c) $R = \emptyset$

→ (a) $R = \{(1,3), (3,1), (2,2)\}$

(i) It is not reflexive as $(1,1) \notin R$.

(ii) It is symmetric as $(1,3)$ and $(3,1) \in R$.

(iii) It is not transistive as $R \in (1,3)$ and $(3,1)$ but $\notin (1,1)$.

(b) $R = \{(1,1), (2,2), (3,3), (1,2)\}$

(i) It is reflexive as $(a,a) \in R$.

(ii) It is not symmetric as $(2,1) \notin R$

(iii) It is transistive as $R \in (1,2)$ and $(2,2)$ then $(1,2)$ also.

(c) $R = \emptyset$

(i) There is no element (x,y) in R and no (y,x) as well. Hence R is symmetric.

(ii) There is no element (x,y) and $(y,z) \in R$ then $(x,z) \in R$. Hence R is transistive.

(iii) $R \notin (x,x), (y,y)$. Hence R is not reflexive.

Camlin Page
Date / /

Ex. Define equivalence relation. A relation on set {1, 2, 3} is given as $R = \{(a,b) \mid a-b$ is even no.$\}$. Check wheathe R is equivalence or not.

→ $S = \{1, 2, 3\}$

$R = \{(a,b) \mid a-b$ is even no$\}$

$R = \{(1,1), (1,3), (2,2), (3,1), (3,3)\}$

→ It is Symmetric as (1,3) and (3,1) ∈ R and also (1,1) ∈ R.

→ It is transistive as (1,1) and (1,3) ∈ R and also (1,3) ∈ R.

☆ String

→ It is finite collection of symbol from alphabate.

→ An alphabate is finite collection of symbol.

Ex. $S = \{a, b, c, \ldots, z\}$

$W = \{0, 1\}$

Here, 0 and 1 alphabates denoted by W.

→ In string 'Mango'. The prefix can be 'Man' and suffix can be 'go'.

* Operations on String

(i) Concatenation.
(ii) Transpose
(iii) Palindrome

(i) Concatenation : In this operation two strings are combined together to form single string.

Ex. $x = \{white\}$, $y = \{house\}$

$xy = \{white\ house\}$

(ii) Transpose : The transpose of operation is also called reverse operation.

Ex. $(xa)^T = (ax)^T$

(iii) Palindrome : Palindrome of string is a property of a string in which string can be read same from left to right as well as from right to left.

Ex. "Madam"

Ex. Consider the string $x = 110$ and $y = 0110$ then find. (i) xy (ii) yx (iii) x^2 (iv) $\in y$

→ (i) $xy = (110) \cdot (0110) = 1100110$

(ii) $yx = (0110) \cdot (110) = 0110110$

(iii) $x^2 = x \cdot x = (110)(110) = 110110$

(iv) $\in y$ means the string belonging to set y which is 0110.

Camlin Page
Date / /

☆ Inductive Proofs

→ To prove S(n),

(i) Basis : Prove S(i) for an initial value i,
i = 1

(ii) Inductive : (i) Assume S(k) is true.
(ii) Prove that S(k+1) is also true.

Ex. Prove : $1+2+3+\ldots+n = n(n+1)/2$

→ (i) Basic of Induction :

Assume, n = 1

L.H.S = 1

R.H.S = $n(n+1)/2 = 1(1+1)/2 = 2/2 = 1$

(ii) Induction hypothesis :

Assume, n = k

$1+2+3+\ldots+k = k(k+1)/2$ ..... (1)

(iii) Inductive step

Assume, n = k+1

L.H.S = $1+2+3+\ldots+k+(k+1)$

$= k(k+1)/2 + (k+1)$

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

$= (k+1)(k+1+1)/2$

= R.H.S

Camlin Page
Date / /

Ex. Prove that $1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$

→ (i) Basic of Induction

Assume, $n = 1$

$L.H.S = (1)^2 = 1$

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

$\therefore L.H.S = R.H.S$

→ (ii) Induction hypothesis, Assume, $n = k$

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

→ (iii) Inductive Step: Assume, $n = k+1$

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

$L.H.S = 1^2 + 2^2 + 3^2 + \dots + k^2 + (k+1)^2$

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

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

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

$= (k+1)\left[\frac{2k^2 + 7k + 6}{6}\right]$

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

$= (k+1)\left[\frac{2k(k+2) + 3(k+2)}{6}\right]$

Camlin Page
Date / /

$$= (k+1)\left[\frac{(2k+3)(k+2)}{6}\right]$$

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

Ex. Prove that $1+3+5+\ldots+r = n^2$ for all $n>0$ where, $r = 2n-1$

→ (i) Basic Induction

Assume, $n = 1$

L.H.S = $1+3+5+\ldots+(2n-1)$

$= (2-1) = 1$

R.H.S = $(n)^2 = 1$

$\therefore$ L.H.S = R.H.S

→ (ii) Induction hypothesis, Assume $n = k$,

$P(k) = 1+3+5+\ldots+(2k-1) = k^2$

→ (iii) Inductive step, Assume $n = k+1$

L.H.S = $1+3+5+\ldots+(2k-1)+(2k+1)$

$= k^2 + (2k+1)$

$= k^2 + 2k + 1$

$= (k+1)^2$

= R.H.S

Camlin Page
Date / /

Ex. Prove that $\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{(n+1)}$

→ (i) Basic Induction, Assume $n = 1$

$$L.H.S = \frac{1}{1(2)} = \frac{1}{2}$$

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

$\therefore$ L.H.S = R.H.S

→ (ii) Induction hypothesis, Assume $n = k$

$$\sum_{i=1}^{k} \frac{1}{i(i+1)} = \frac{k}{(k+1)}$$

→ (iii) Inductive step, Assume $n = k+1$

$$\sum_{i=1}^{k+1} \frac{1}{i(i+1)} = \left(\sum_{i=1}^{k} \frac{1}{i(i+1)} \right) + \frac{1}{(k+1)(k+1+1)}$$

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

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

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

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

$$= R.H.S$$

Camlin Page
Date / /

Ch:2 Regular Languages

☆ Regular Expressions

→ The language accepted by finite automata can be easily described by simple expressions called regular expression.

(i) $\emptyset$ is a regular expression which denotes empty set.
(ii) ε is a regular expression and denotes the set $\{\varepsilon\}$ and it is null string.
(iii) For each 'a' in Σ 'a' is a regular expression and denotes the set $\{a\}$.
(iv) If r and s are regular expressions denoting the languages L_1 and L_2 respectively then,
r+s is equivalent to $L_1 \cup L_2$. Union
rs is equivalent to $L_1 \cdot L_2$. Concatenation.
r^* is equivalent to L_1^*. closure.

→ r^* is known as kleen closure or closure which indicates occurrence for r for ∞ number of times.

→ ~~$R^* = \{$~~

→ $R = a^* = \{\varepsilon, a, aa, aaa, \ldots\}$
$R = a^+ = \{a, aa, aaa, \ldots\}$

☆ Regular Language: These are the languages that are represented by RE. These are the languages that can be modeled using FA.

Camlin | Page
Date / /

Ex. Write regular expression for following languages of all string in $\{0,1\}^*$.

→ Containing any number of 0's and 1's

$$r.e = (0+1)^*$$

→ Containing any number of 0's and 1's, except null string.

$$r.e = (a+1)^+$$

→ Strings which are start with 0 and end with 1.

$$r.e = 0\,(0+1)^*\,1$$

→ String that any number of 0's followed by any number of 1's.

$$r.e. = 0^*\,1^*$$

→ String that 3rd character from right end is always 0.

$$r.e. = (0+1)^*\,0\,(a+1)\,(0+1)$$

→ String which accept at least two 1's.

$$r.e. = (0+1)^*\,1\,(0+1)^*\,1\,(0+1)^*$$

→ String which accept exactly two b's.

$$r.e = a^*\,b\,a^*\,b\,a^*$$

Camlin Page
Date / /

→ String that do not end with 01.

$$r.e = \varepsilon + 0 + 1 + [(0+1)^* \cdot (00+11+10)]$$

→ String that begin and end with 00 and 11.

→ String that begin or end with 00 and 11.

$$r.e = (00+11)(0+1)^*(00+11)$$

$$r.e = [(00+11)(0+1)^*] + [(0+1)^*(00+11)]$$

→ String with odd number of 1's.

$$r.e = (0^*10^*10^*)^*10^*$$

→ String with odd number of 0's.

$$r.e = (1^*01^*01^*)^*01^*$$

→ String that starts with 1 but not end with 10.

$$r.e = 1(0+1)^*(01+00+11+1)$$

→ String that do not contain the substring 110.

$$r.e = 0^*(100)^*1^*$$

→ String that contain both 101 and 010 as substring

$$r.e = (0+1)^*[(101(0+1)^*010) + (010(0+1)^*101) + (1010) + (0101)](0+1)^*$$

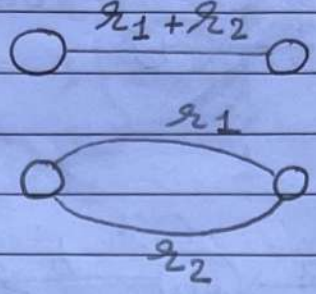
→ String in which both number of 0's and 1's are odd.

$$r.e = 0(00+010+100+101)^* + 1(11+101+011+110)^* + 01 + 10$$

Camlin Page
Date / /

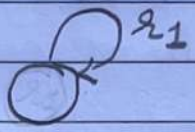
☆ RE to FA

* Union :
$(r_1 + r_2)$

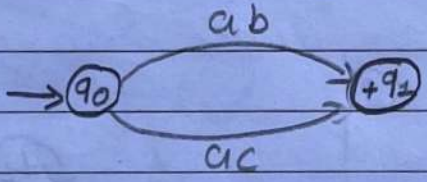


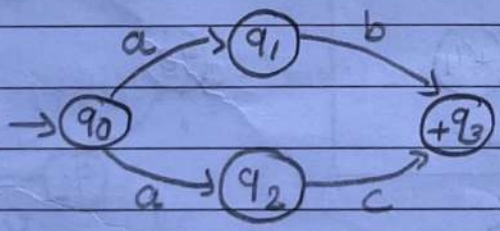
* Concetanation :
$(r_1 . r_2)$

* Closure :
r_1^*

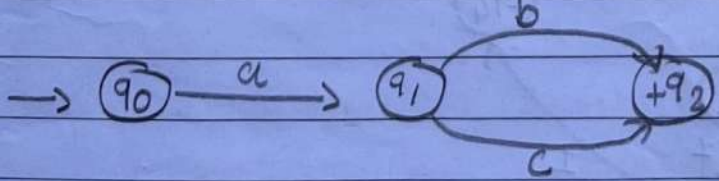


Ex. $ab + ac$

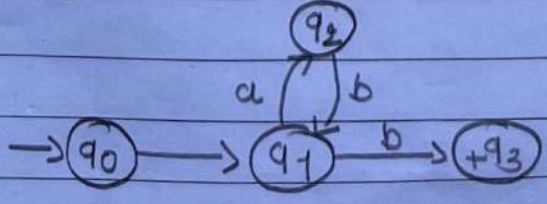




Ex. $a + bc$

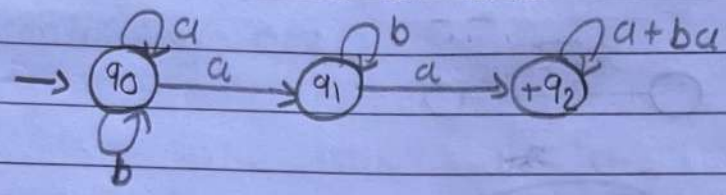


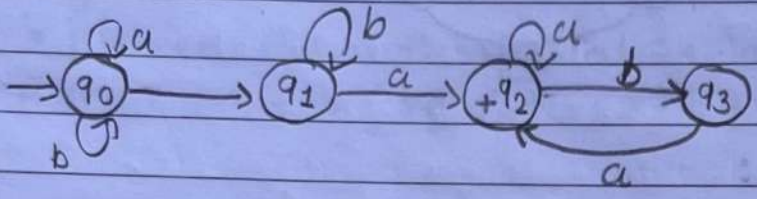
Ex. $a(ab)^* b$



Camlin Page
Date / /

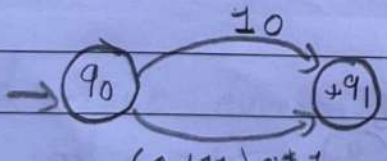
Ex (a+b)* a b* a (a+ba)*

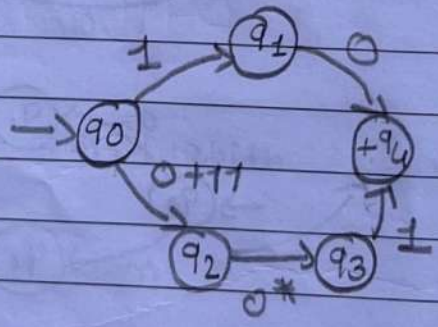


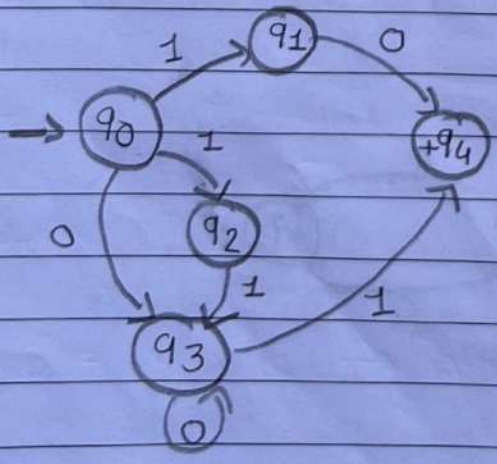


Ex. (a+b)* (abb + a*b)

Ex. 10 + (0+11)0*1

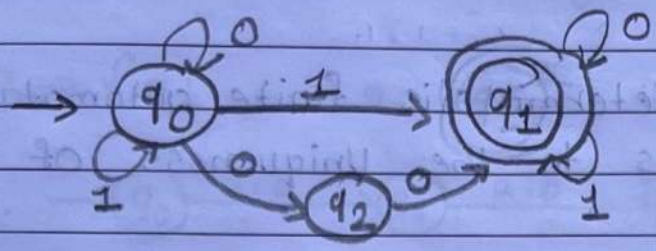




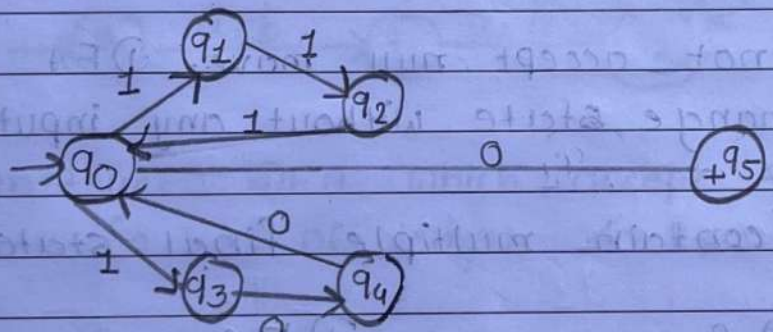


Camlin Page
Date / /

Ex. $(0+1)^* (1+00) (0+1)^*$



Ex. $(111+100)^* 0$



Camlin Page
Date / /

☆ DFA (Deterministic Finite automata)

→ DFA refers to determinstic finite automata. Deterministic refers to the Uniqueness of the computation.
The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time.

→ In DFA, there is only one path for specific input from current to next state.

→ DFA does not accept null move. DFA can not change state without any input.

→ DFA can contain multiple final state.

* Formal Definition of DFA

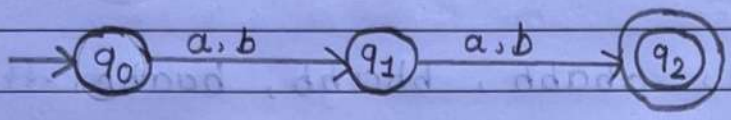
→ DFA is a collection of 5 - tuples,
Q : finite set of state.
Σ : finite set of input symbol.
q_0 : Initial state
F : Final state
δ : Transition function

$$\delta = Q \times \Sigma \rightarrow Q$$

Camlin Page
Date / /

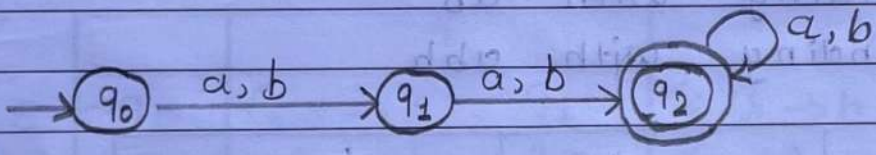
Ex. Construct DFA which accept set of all string over {a, b} of length = 2

→ L = {ab, ba, aa, bb}



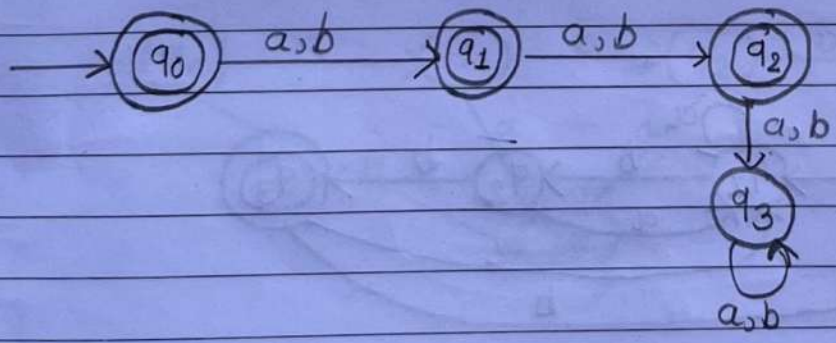
Ex. Construct DFA which accept set of all string over {a,b} of length >= 2

→ L = {aa, ab, ba, bb, aaa, aab, bba, ...}



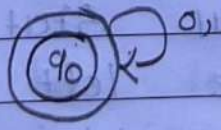
Ex. Construct DFA which accept set of all string over {a,b} of length <= 2.

→ L = {ε, a, b, aa, ab, ba, bb}



Ex. Construct a DFA which accept set of all strings over {0, 1}

→ L = {ε, 0, 1, 00, 11, 10, 01, 1010, 0111, ...}



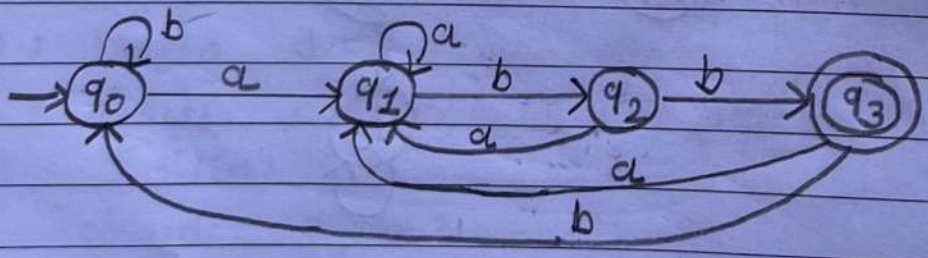
Ex. Construct DFA which accept set of all strings over {a, b} that ends with a string "abb".

-> L = { abb, aabb, ababb, bbabb, baabb, ... }

* Method

q_0 -> No symbol accepting state
q_1 -> Ending with a
q_2 -> Ending with ab
q_3 -> Ending with abb

		a	b
ε	$\to q_0$	$(\varepsilon.a \to a)\ q_1$	$(\varepsilon.b \to b)\ q_0$
a	q_1	$(aa)\ q_1$	$(ab)\ q_2$
ab	q_2	$(abb)\ q_1$	$(abb)\ q_3$
abb	$*q_3$	$(abba)\ q_1$	$(abbb)\ q_0$



-> In this method in transition table check string ending state.

-> Here, if there is no string match in table then write initial state q_0.

-> In this we can minimize left side string and consider right most side string.

Camlin Page
Date / /

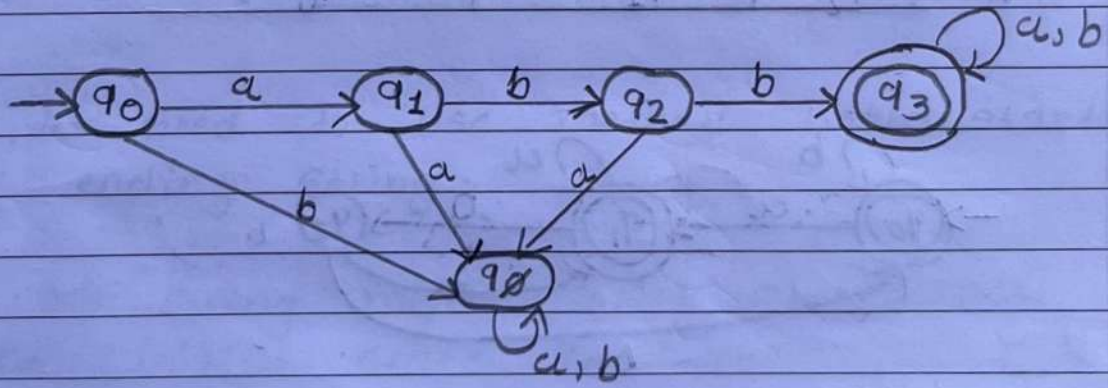
Ex. Construct a DFA which accept set of all strings over {a, b} that start with "abb"

→ L = { abba, abbaa, abbab, abbabb, ... }

* Method.

q_0 → No symbol accepting state
q_1 → starts with a.
q_2 → starts with ab.
q_3 → starts with abb.

		a		b	
ε	→q_0	(ε.a → a)	q_1	(ε.b → b)	q_ϕ
a	q_1	(a a)	q_ϕ	(a b)	q_2
ab	q_2	(ab a)	q_ϕ	(abb)	q_3
abb	* q_3	(abb a)	q_3	(abb b)	q_3
	q_ϕ		q_ϕ		q_ϕ



→ In this method transition table we check string at starting state.

→ If there is no string match with starting string in table then write q_ϕ.

→ Here we have to consider full state string, we can not minimize right side string.

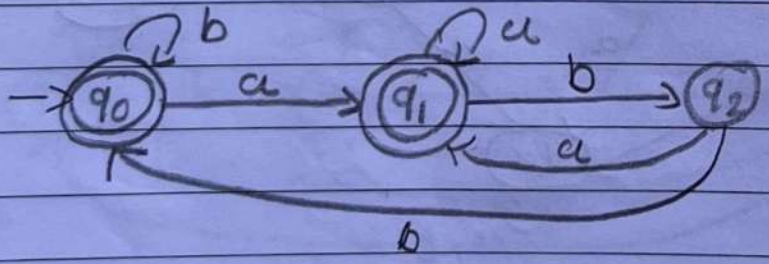
Camlin Page
Date / /

Ex. Construct a DFA which accept set of all strings over {a, b} that does not end with ab.

→ In, this construct DFA which accep set of all strings that end with ab. After construct for convert not end with ab just interchange final state with normal state.
(non-final)

q_0 → No symbol accept state
q_1 → Ends with a
q_2 → Ends with ab

	a	b
ε	q_0	q_1	q_0
a	q_1	q_1	q_2
ab	q_2	q_1	q_0



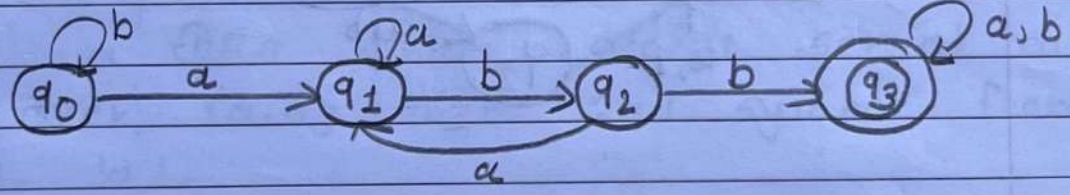
Camlin Page
Date / /

Ex. Construct a DFA which accept set of all strings over {a, b} that contains sub string "abb".

→ L = {abb, abba, babb, babba, ... }

q_0 → No symbol accepting state
q_1 → Substring as 'a'
q_2 → Substring as 'ab'
q_3 → Substring as 'abb'

		a		b	
ε	→ q_0	(ε-a→a)	q_1	(ε-b→b)	q_0
a	q_1	aa	q_1	ab	q_2
ab	q_2	aba	q_1	abb	q_3
abb	* q_3	abba	q_3	abbb	q_3



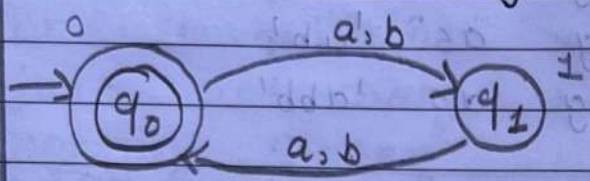
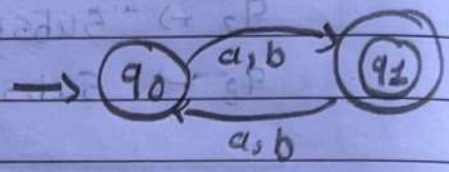
→ Here, we need to use method both starting and ending string.

→ If No string match with state then write initial state q_0.

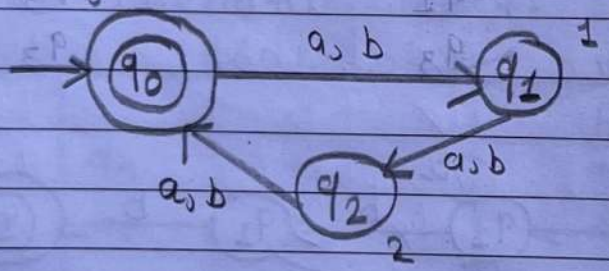
Camlin Page
Date / /

Ex. Construct a DFA which accept set of all string over {a, b} where length of string is

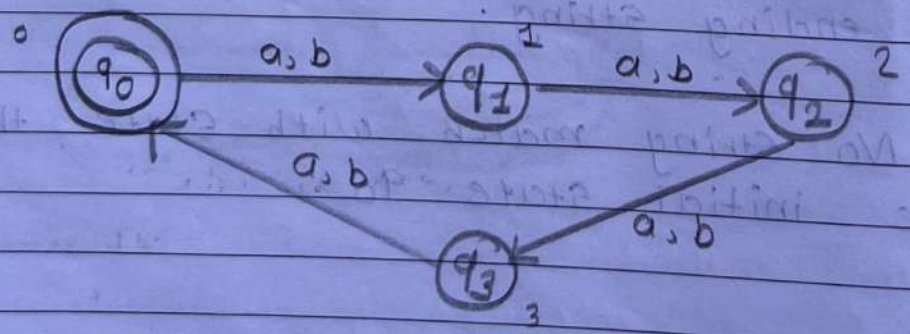
* divisible by 2,

even length	odd length
	

* divisible by 3,



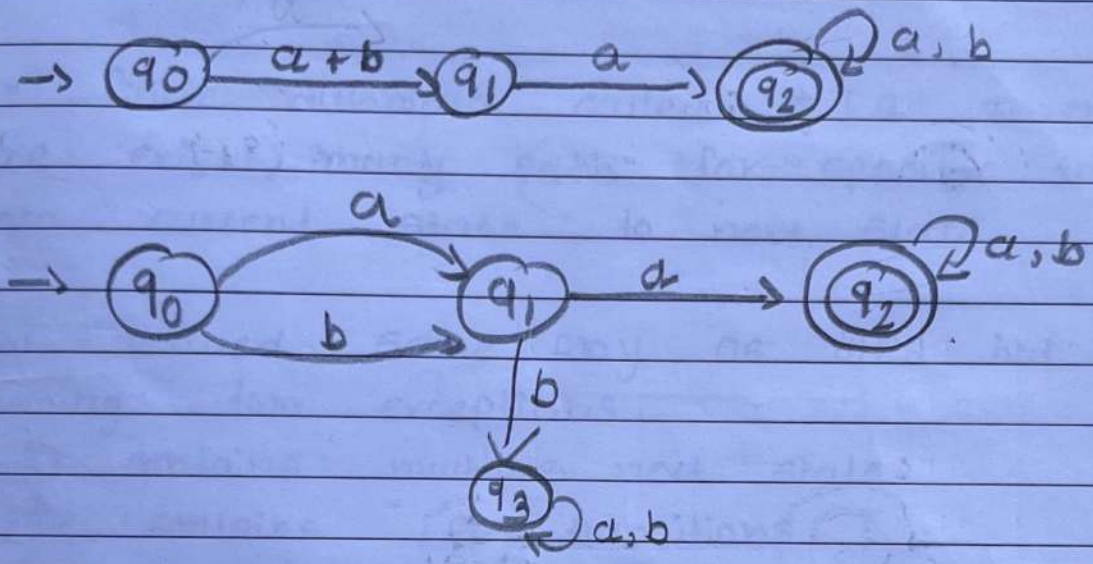
* divisible by 4,



Camlin Page
Date / /

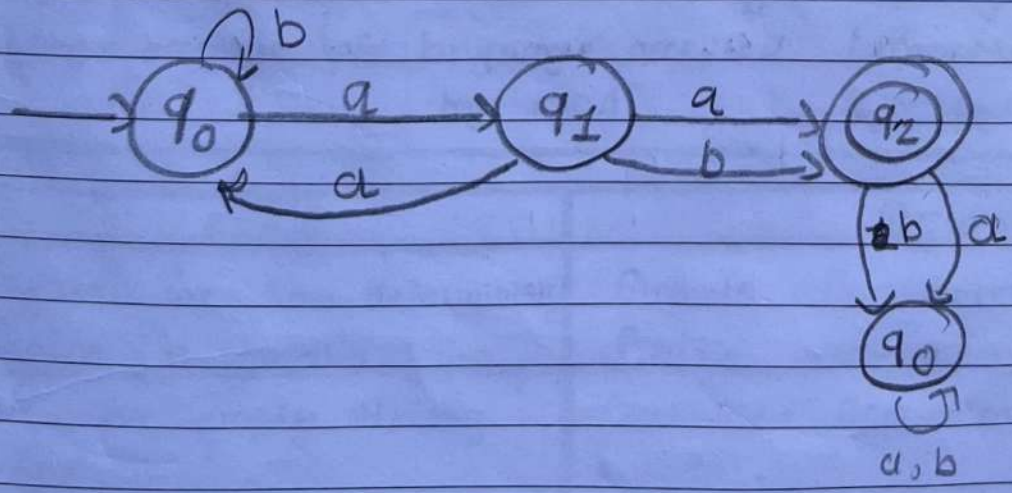
Ex. Construct a DFA which accept set of all string over {a,b} that 2nd symbol from LHS is 'a'.

→ $(a+b)\,a\,(a+b)^*$



Ex. Construct a DFA which accept set of all string over {a,b} that 2nd symbol from RHS is 'a'.

→ $(a+b)^*\,a\,(a+b)$



Camlin Page
Date / /

☆ NFA (Non Deterministic finite Automata)

→ NFA stands for non-deterministic finite automata. It is easy to construct an NFA when compared to DFA for RE.

→ The finite automata called NFA ~~Where~~ when there exist many path for specific input from current state to next state.

→ NFA defined same way as DFA but with following two exceptions,
- It contains multiple next state.
- It contains ε transitions.

→ It contains collection of 5-tuples same as DFA.

*

FA	NFA	NFA - ^
→ Determinstic in nature.	Non-Deterministic	Non-Deterministic
→ Difficult to design.	Easy to design.	Easy to design.
→ Easy to recognize language accepted by FA.	Difficult to recognize language accepted by NFA.	Using ε transitions concetenated languag accepted by this NFA

*

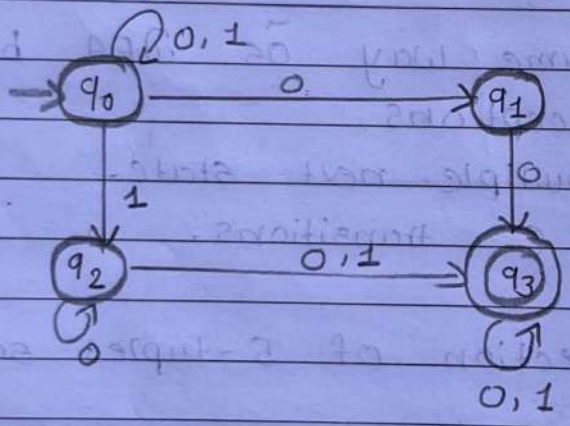
NFA	DFA
→ Stands for Non-deterministic finite automata.	Stands for deterministic finite automata.
→ Can Use empty string transition.	Can not use empty string transition
→ Every input there can be more than one transition. Construction is simple.	Every input there can be only one transition. Construction is difficult.

Camlin Page
Date / /

* Design NFA diagram for following transition table.

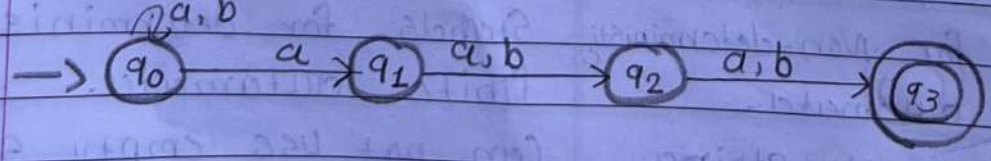
→ *

	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_3\}$	—
q_2	$\{q_2, q_3\}$	$\{q_3\}$
* q_3	$\{q_3\}$	$\{q_3\}$



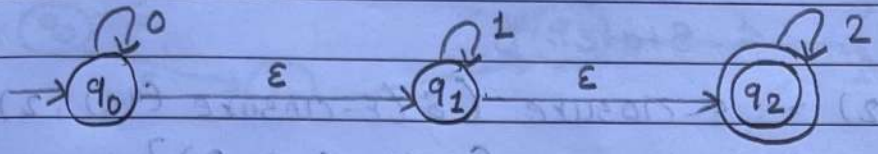
* Construct NFA for language L which accepts all the strings in which the third symbol from right hand is always a. over $L = \{a, b\}$

→ R.E = $(a+b)^* a (a+b)^* (a+b)^*$



Camlin Page
Date / /

* Convert given NFA into its equivalent DFA.



→ ε-closure $(q_0) = \{q_0, q_1, q_2\} = A$
ε-closure $(q_1) = \{q_1, q_2\} = B$
ε-closure $(q_2) = \{q_2\} = C$

→

		0	1	2
	q_0 A	q_0	q_1	q_2
	q_1 B	$\emptyset$	q_1	q_2
	q_2 C	$\emptyset$	$\emptyset$	q_2

→ $\delta(q_0, 0)^{A,0} = \varepsilon$-closure $(\delta(\varepsilon\text{-closure}(q_0), 0))$
$= \varepsilon$-closure $(\delta\{q_0, q_1, q_2\}, 0))$
$= \varepsilon$-closure $(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$
$= \{q_0\}$
$= \{q_0, q_1, q_2\}$ → State : A

→ $\delta(q_0, 1)^{A,1} = \varepsilon$-closure $(\delta(\varepsilon\text{-closure}(q_1), 1))$
$= \varepsilon$-closure $(\delta(q_0, q_1, q_2), 1)$
$= \{q_1\}$
$= $ State : B

→* $\delta(q_0, 2)^{A,2} = \varepsilon$-closure $(\delta(\varepsilon\text{-closure}(q_2), 2))$
$= \varepsilon$-closure $(\delta(q_0, q_1, q_2), 2)$
$= \{q_2\}$ }*
$= $ State : C

→ $\delta(q_1, 0)^{B,0} = \varepsilon$-closure $(\delta(\varepsilon\text{-closure}(q_1), 0))$
$= \varepsilon$-closure $(\delta(q_1, q_2), 0)$
$= \emptyset$

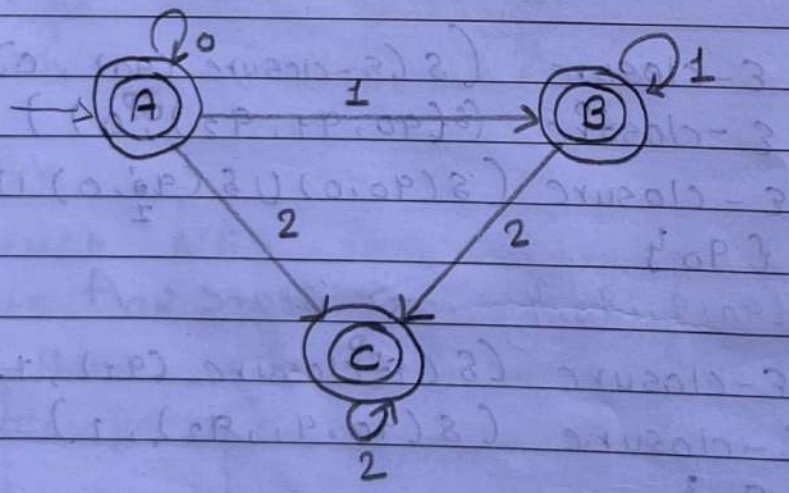
Camlin Page
Date / /

→ $\delta(q_1, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_1), 1))$
$= \varepsilon\text{-closure}(\delta(q_1, q_2), 1)$
$= \{q_1\}$
$=$ State : B

→ $\delta(q_1, 2) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_1), 2))$
$= \varepsilon\text{-closure}(\delta(q_1, q_2), 2)$
$= \{q_2\}$
$=$ State : C

→ $\delta(q_2, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_2), 0))$
$= \varepsilon\text{-closure}(\delta(q_2), 0)$
$= \emptyset$

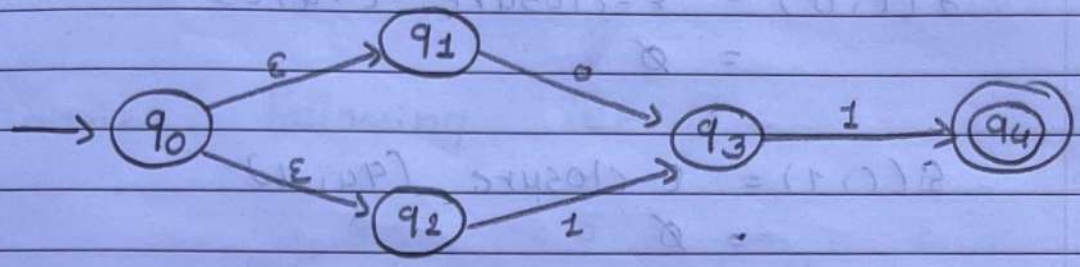
→ $\delta(q_2, 1) = \emptyset$
$\delta(q_2, 2) = q_2$



→ For finding final state check question final state in set of all state.

Camlin Page
Date / /

* Convert NFA to its equivalant DFA



→ ε-closure {q0} = {q0, q1, q2}
{q1} = {q1}
{q2} = {q2}
{q3} = {q3}
{q4} = {q4}

→ ε-closure {q0} = {q0, q1, q2} = A

δ(A, 0) = ε-closure (δ(q0, q1, q2), 0)
= ε-closure (q3)
= q3 = state : B

δ(A, 1) = ε-closure (δ(q0, q1, q2), 1)
= ε-closure (q3)
= q3 = state : B

δ(B, 0) = ε-closure (δ(q3, 0))
= ∅

δ(B, 1) = ε-closure (δ(q3, 1))
= ε-closure (q4)
= q4 = state : C

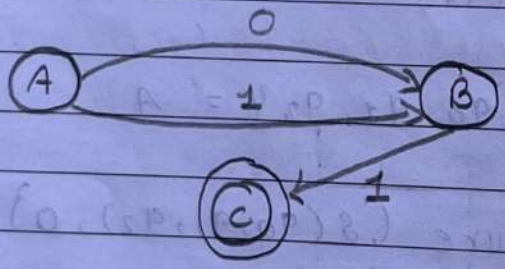
*

Camlin Page
Date / /

$\delta(C, 0) = \varepsilon\text{-closure}(q_4, 0)$
$= \emptyset$

$\delta(C, 1) = \varepsilon\text{-closure}(q_4, 1)$
$= \emptyset$

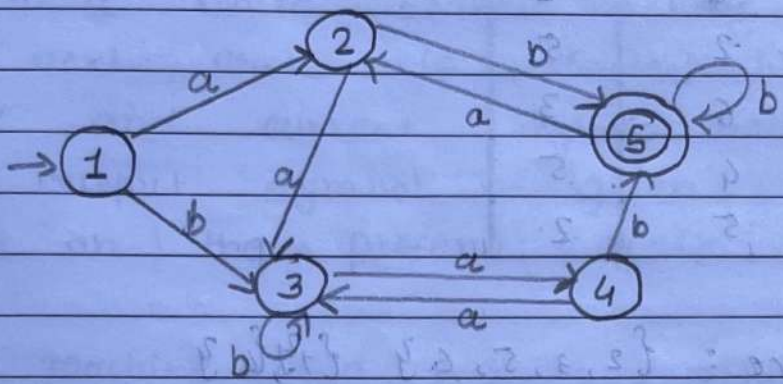
	0	1
A	B	B
B	∅	C
C	∅	∅



Camlin Page
Date / /

☆ Minimization of Finite Automata

Ex. Minimize following DFA.

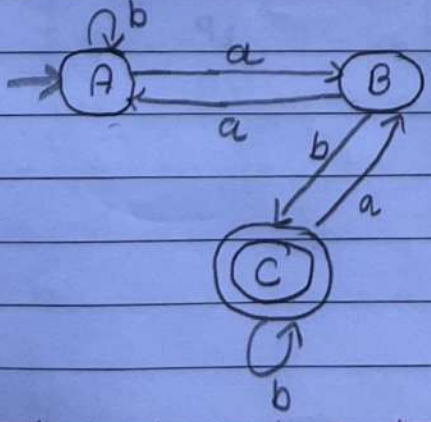


→ Transition table for given DFA.

	a	b
→1	2	3
2	3	5
3	4	3
4	3	5
* 5	2	5

→ 0-Equivalence : {1, 2, 3, 4} {5}
1-Equivalence : {1, 3}, {2, 4}, {5}
2-Equivalence : {1, 3} {2, 4} {5}
(A) (B) (C)

→



	a	b
A	B	A
B	A	C
C	B	C

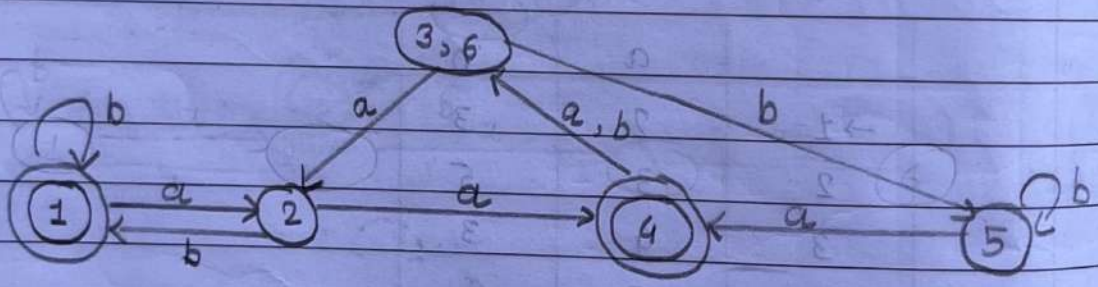
Ex. Minimize the following DFA.

	a	b
* 1	2	1
2	4	1
3	2	5
* 4	6	3
5	4	5
6	5	2

→ 0-Equivalence : {2, 3, 5, 6} {1, 4}

~~1-Equivalence : {2, 5} {3} {6} {1} {4}~~

1-Equivalence : {2}, {3, 6} {5} {1} {4}



	a	b
1	2	1
2	4	1
3	2	5
4	3	3
5	4	5

Camlin Page
Date / /

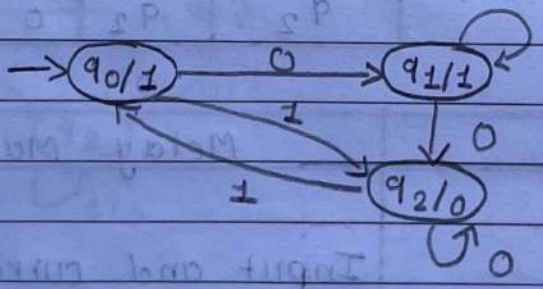
☆ Moore and Mealy Machines

* Moore Machine

→ It is a finite state machine in which the next state is decided by current state and current input symbol. The output symbol at given time depends only on the present state of the machine.

Moore machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$,
Q is finite set of state.
Σ is finite set of input symbol.
Δ is an output alphabet.
δ is transition function such that $Q \times \Sigma \to Q$.
λ is output function $Q \to \Delta$
q_0 is a initial state of machine.

→ Ex.



Current state	Next state		output
	0	1	
q_0	q_1	q_2	1
q_1	q_2	q_1	1
q_2	q_2	q_0	0

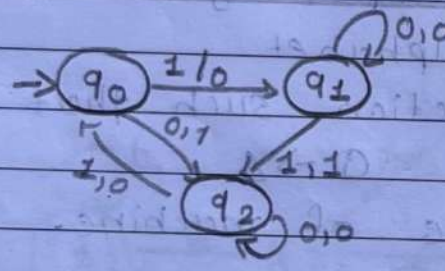
Camlin Page
Date / /

* Melay Machine

→ Melay machine is a machine in which output symbol depends upon present input symbol and present state of the Machine.

→ Melay Machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, Five tupples same as moore machine only different is,
λ is a machine function such that $Q \times \Sigma \to \Delta$

→ Ex.



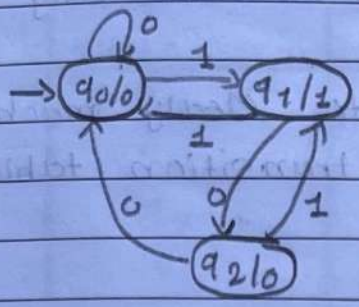
Present State	0		1	
	State	o/p	State	o/p
q_0	q_2	1	q_1	0
q_1	q_1	0	q_2	1
q_2	q_2	0	q_0	0

* Moore Machine	Melay Machine
→ Only current state determine output.	Input and current state determine output.
→ Places its output on the transition.	Places its output on the transition.
→ More state required.	Less state required.
→ They react slower to input.	They react faster to inputs.
→ Output is placed on state.	Output is placed on transition.
→ Easy to design.	Difficult to design.

Camlin Page
Date / /

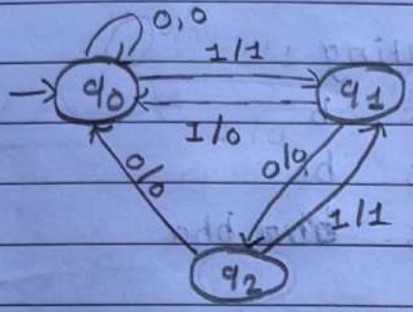
* Moore to Melay

→ Moore



Current State	Next State		o/p
	0	1	
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_0	q_1	0

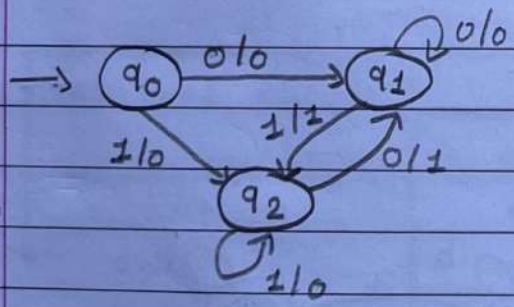
Melay



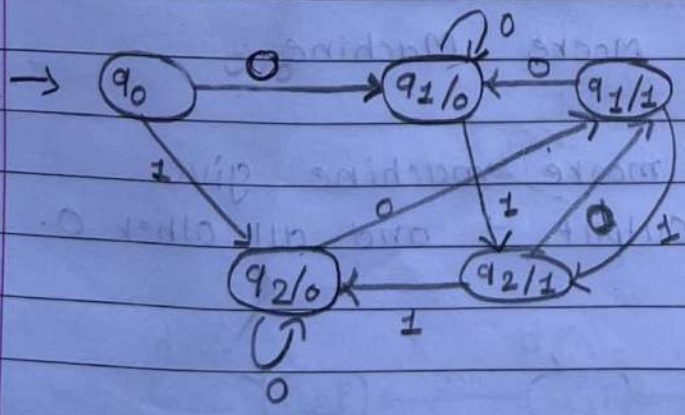
Current State	0		1	
	State	o/p	State	o/p
q_0	q_0	0	q_1	1
q_1	q_2	0	q_0	0
q_2	q_0	0	q_1	1

* Melay to Moore

Mealy



Moore



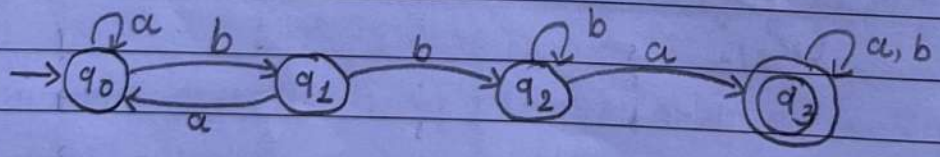
Camlin | Page
Date / /

Ex. Construct the moore machine that counts no. of occurrences of substring 'bba' over $\Sigma = \{a, b\}$.
Now convert this Moore to Mealy machine and show diagram and transition table.

→ Construct DFA:

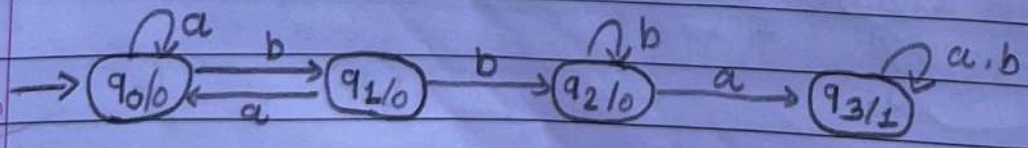
q_0 → No symbol accepting
q_1 → Sub string as b
q_2 → Sub string as bb
q_3 → Sub string as bba

		a	b
ε	q_0	q_0	q_1
b	q_1	q_0	q_2
bb	q_2	q_3	q_2
bba	q_3	q_3	q_3



→ Convert DFA to Moore Machine

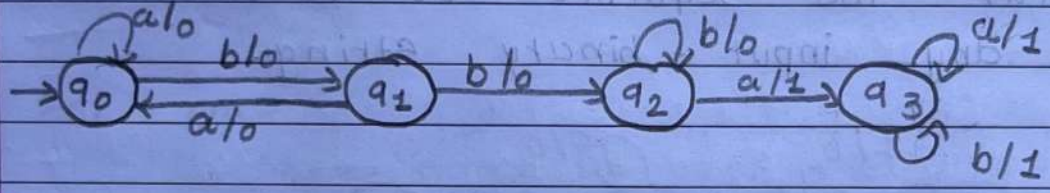
→ For converting in moore machine give final state as output 1 and all other 0.



Camlin Page
Date / /

Current state	Next state		o/p
	a	b	
q_0	q_0	q_1	0
q_1	q_0	q_2	0
q_2	q_3	q_2	0
q_3	q_3	q_3	1

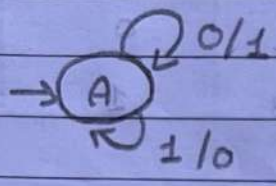
* Convert to Mealy Machine



Current state	Next state			
	(a) state	o/p	(b) state	o/p
q_0	q_0	0	q_1	0
q_1	q_0	0	q_2	0
q_2	q_3	1	q_2	0
q_3	q_3	1	q_3	1

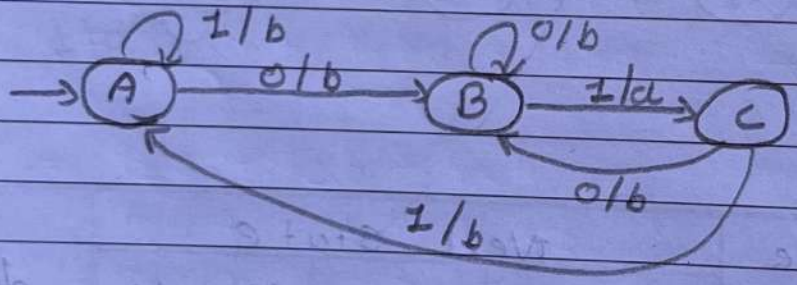
Camlin Page
Date / /

Ex. Construct Mealy Machine that produces the 1's complement of any binary input string.



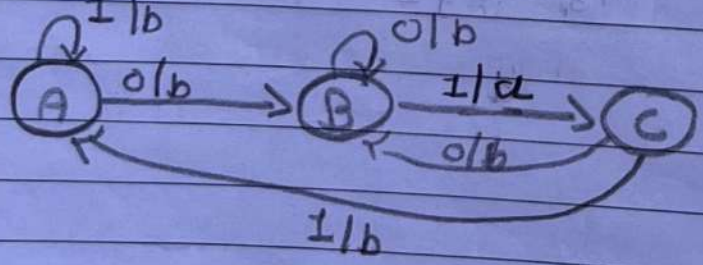
Ex. Construct a Mealy Machine that prints 'a' whenever the sequence '01' is encountered in any input binary string.

→



Ex. Construct a Mealy Machine that prints 'a' whenever the sequence '01' is encountered in any input binary string.

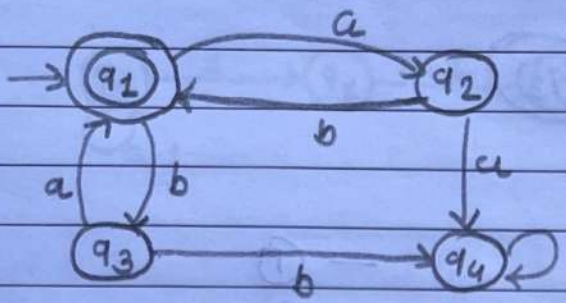
→



Camlin Page
Date / /

Find Regular Expression from DFA

Ex.



→ $q_1 = \varepsilon + q_2 b + q_3 a$ — (1)

$q_2 = q_1 a$ — (2)

$q_3 = q_1 b$ — (3)

$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$ — (4)

→ For equation (1),

$q_1 = \varepsilon + q_2 b + q_3 a$

$= \varepsilon + q_1 ab + q_1 ba$

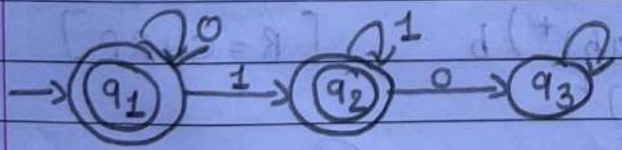
$q_1 = \varepsilon + q_1 (ab + ba)$ $[R = Q + RP]$

$R \quad Q \quad R \quad P$

$= \varepsilon (ab + ba)^*$ $[R = QP^*]$

$R = (ab + ba)^*$ $[\varepsilon R = R]$

Ex.



$q_1 = \varepsilon + q_1 0$ — (1)

$q_2 = q_1 1 + q_2 1$ — (2)

$q_3 = q_2 0 + q_3 0 + q_3 1$ — (3)

→ $q_1 = \varepsilon + q_1 0$ $[R = Q + RP]$

$q_1 = \varepsilon 0^* = 0^*$ $[R = QP^*]$

→ $q_2 = q_1 1 + q_2 1$

$= 0^* 1 + q_2 1$ $[R = Q + RP]$

$= 0^* 1 (1)^*$ $[R = QP^*]$

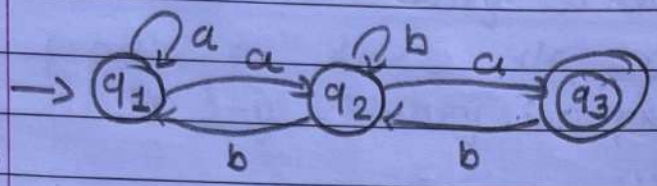
→ R.E = $q_1 + q_2 = 0^* + 0^* 1 (1)^* = 0^* (\varepsilon + 1 1^*)$ $[\varepsilon + RR^* = R^*]$

$= 0^* 1^*$

Camlin Page
Date / /

☆ Find Regular Expression from NFA

Ex.



→ $q_3 = q_2 a$ — (1)

$q_2 = q_1 a + q_2 b + q_3 b$ — (2)

$q_1 = \varepsilon + q_1 a + q_2 b$ — (3)

(1) → For equation (1),

$q_3 = q_2 a$

$= (q_1 a + q_2 b + q_3 b) a$

$= q_1 aa + q_2 ba + q_3 ba$ — (4)

(2) → $q_2 = q_1 a + q_2 b + q_3 b$

$= q_1 a + q_2 b + q_2 ab$

$= q_1 a + q_2 (b + ab)$ — (5)

$= (q_1 a)(b + ab)^*$ — (6)

(3) → $q_1 = \varepsilon + q_1 a + q_2 b$

$= \varepsilon + q_1 a + ([q_1 a)(b + ab)^*) b$

$= \varepsilon + q_1 (a + a(b + ab)^*) b$ [R = Q + RP]

$= \varepsilon ((a + a(b + ab)^*) b)^*$

$= (a + a(b + ab)^* b)^*$ — (6)

→ For Final state (q_3),

$q_3 = q_2 a$

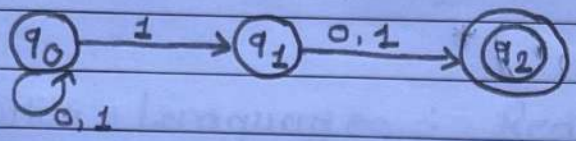
$= (q_1 a (b + ab)^*) a$

R.E = $q_3 = ((a + a(b + ab)^* b)^* a(b + ab)^* a$

Camlin Page
Date / /

☆ NFA to DFA Conversion

Ex.

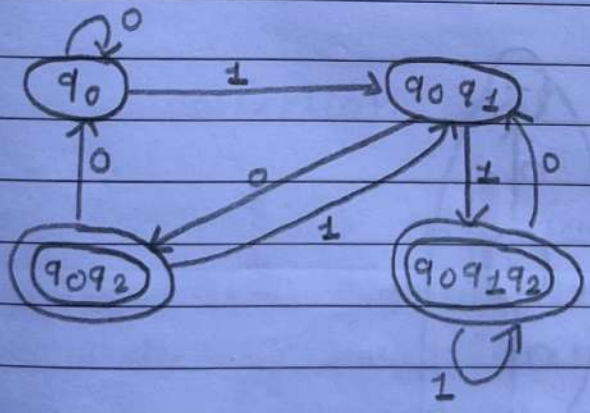


→ NFA :

	0	1
→ q_0	q_0	q_0, q_1
q_1	q_2	q_2
* q_2	–	–

→ DFA :

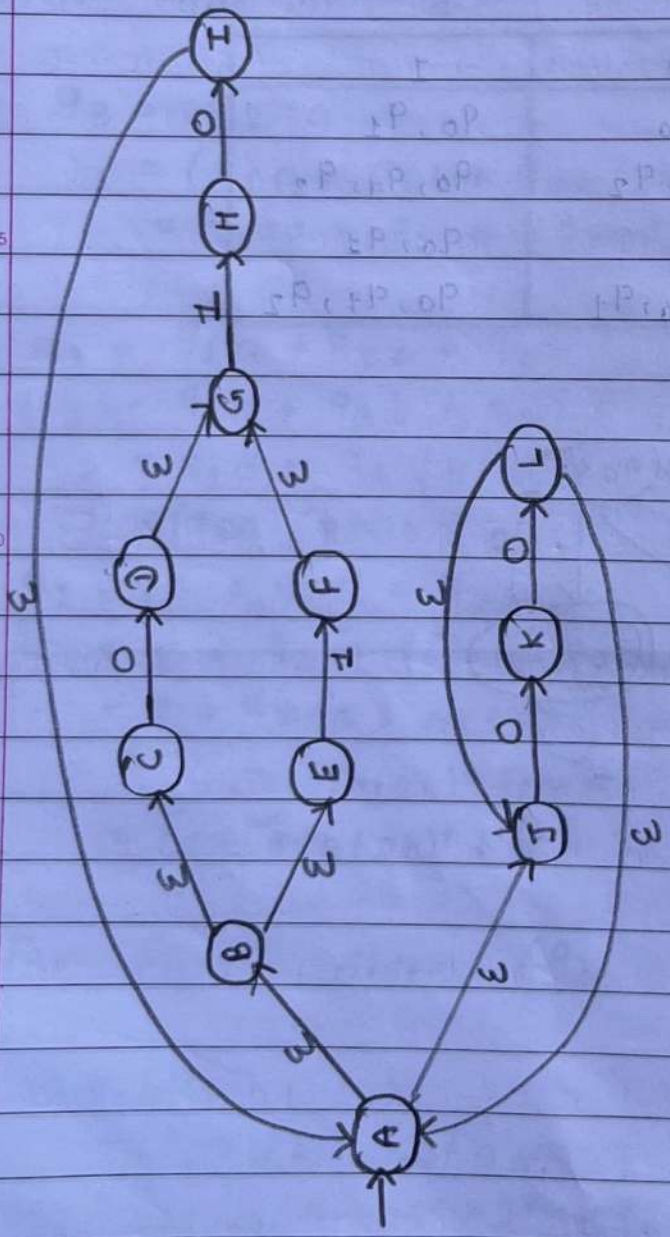
	0	1
→ q_0	q_0	q_0, q_1
q_0, q_1	q_0, q_2	q_0, q_1, q_2
* q_0, q_2	q_0	q_0, q_1
* q_0, q_1, q_2	q_0, q_1	q_0, q_1, q_2



Camlin | Page
Date / /

RE to ε-NFA

Ex. $(0+1)^* 10 + (00)^*)^*$



Camlin Page
Date / /

☆ Define Regular language and Finite Automata.

→ Regular Language : Regular language are the languages that are represented by regular expression. The regular languages are the languages that can be modeled using finite automata.

→ Finite Automata : The finite state system represents a mathematical model of a system with certain input. The model finally gives certain output. The input that given to machine processed by various states, these states are called as intermediate state.

- It is collection of 5-tuples $(Q, \Sigma, \delta, q_0, F)$

Q is finite set of state.
Σ is input alphabate, indicate input set.
q_0 is an initial state.
F is a final state.
δ is a transition function.

☆ Recursive definition of NFA.

→ Let, $M = \{Q, \Sigma, q_0, A, \delta\}$ be an NFA. The fnction $\delta^* : Q \times \Sigma^* \to 2^Q$ is defined as follow.

(1) For any $q \in Q$, $\delta^*(q, \wedge) = \{q\}$

(2) For any $q \in Q$, $y \in \Sigma^*$, and $a \in \Sigma$,

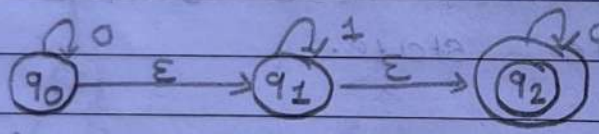
$$\delta^*(q, ya) = \bigcup_{r \in \delta^*(q,y)} \delta(r, a)$$

Camlin Page
Date / /

☆ Define ε-NFA and State procedure of converting ε-NFA to NFA and FA.

→ Epsilon NFA : It is similar to NFA but have just a minor difference which is epsilion (ε) move. The transitions without consuming an input symbol are called ε-transition.

- In the state diagram, they are labeled with ε.
- ε-transition provide convenient way of modeling of the system.
- Due to empty move, the first string of language may be empty or epsilion.



→ ε-NFA to NFA.

- Step 1 : Find out all the ε-transitions from each state form Q. That will be called e-closure.
- Step 2 : Then, δ1 transition can be obtained. The δ1 transition means ε-closure on δ moves.
- Step 3 : Repeat step 2 for each input symbol and for each state.
- Step 4 : By using resultant state, the transition table for equivalent NFA without ε can be buit.

→ $\delta'(q_1, a) = \varepsilon\text{-closure}\ (\delta\ (\varepsilon\text{-closure}\ (q_1), a))$

Camlin Page
Date / /

☆ Explain Procedure of minimize DFA.

- Step 1 : Remove all the state that are unreachable from the initial state via any set of the transition of DFA.
- Step 2 : Draw transition table for all pair of state.
- Step 3 : Split the transition table into two tables T_1 and T_2. T_1 contains all non-final state. T_2 contains final state.
- Step 4 : Find similar rows from T_1. Means, find two state which have the same value of a and b and remove one of them.
- Step 5 : Repeat step 3 until we find no similar rows available in transition table T_1.
- Step 6 : Repeat step 3 and 4 for T_2.
- Step 7 : Now combined the reduced T_1 and T_2 tables. That combined transition table is transition table of minimized DFA.

Camlin | Page
Date / /

☆ Explain Pumping Lemma and its application.

→ This is a basic and important theorem used for checking weather given string is accepted by regular expression or not. In short, this lemma tells us weather given language is regular or not.

* Application

→ Pumping lemma is to be applied to show that certain languages are not regular.
→ It should never be used to show a language is regular.

- If L is regular, It satisfies pumping lemma.
- If L does not satisfy pumping lemma, it is non-regular.

* Method

→ At first we have to assume that L is regular.
→ So pumping lemma should hold for L.
→ Use pumping lemma to obtain contradiction:
- Select w such that $|w| \geq c$
- Select y such that $|y| \geq 1$
- Select x such that $|xy| \leq c$
- Assign remaining string to z.
- Select k such that the resulting string is not in L.

Camlin Page
Date / /

☆ Explain Kleene's theorem Part - I

→ It is used to show the equivalence between regular language and expressions, and finite automata.

→ Kleene theorem state that :
For any regular expression of a language there exists a finite automaton.

→ In simple word, regular expression can be used to represent finite automaton and vice versa. Regular expression are composed Union, Concetenation and Kleene's star.

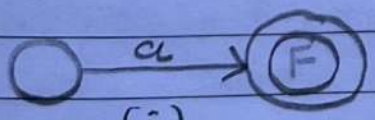
→ Kleen's theorem defines rules to perform these operation on regular expression to convert them into FA.

* Union

→ $r_1 + r_2$ is a regular expression too, whose corresponding language is $L(r_1) \cup L(r_2)$.

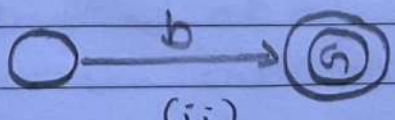
$R = S + T$

(S)



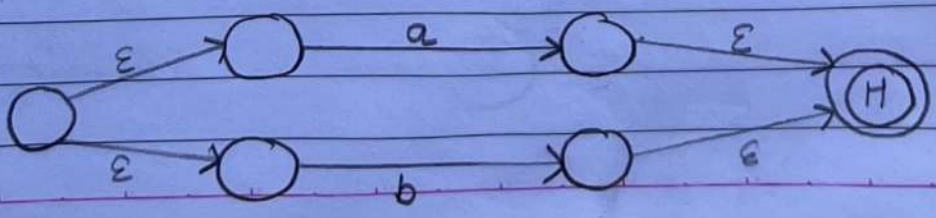
(i)

(T)



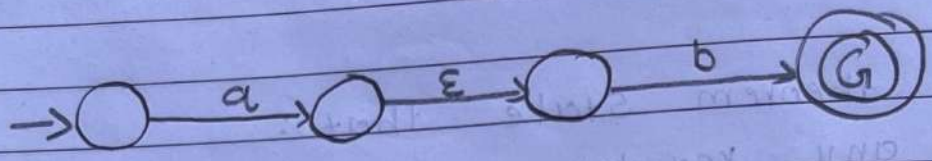
(ii)

(R)



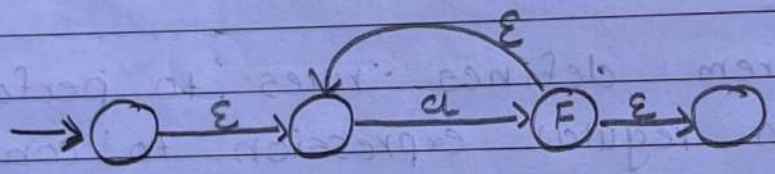
* Concetenation

→ $r_1 . r_2$ is a regular expression too, whose corresponding language is $L(r_1) . L(r_2)$.



* Closure

→ r_1^* is a regular expression too, whose corresponding language is $L(r_1)^*$

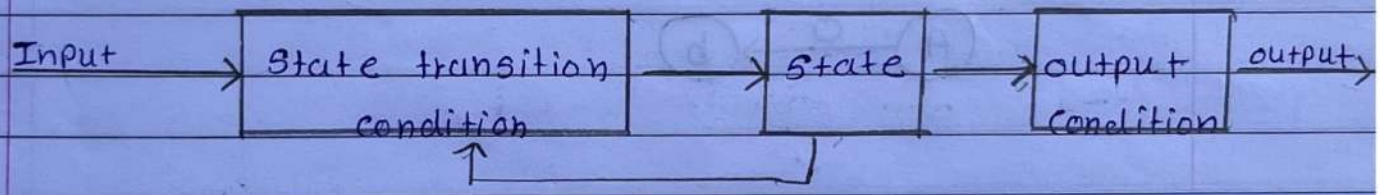


Camlin Page
Date / /

☆ What is finite State Machine.

→ A finite state machine has a set of state and two function called next state and output function.

- The set of state corresponds to all the possible combination of internal storage. If there are n bits of storage, there are 2n possible state.
- The next state function is a combinational logic function that given input and current state, determines next state of system.



→ There are two types of finite state machine:
(i) Moore Machine (ii) Mealy Machine.

* Component

→ State : These are drawn with circle. (A)

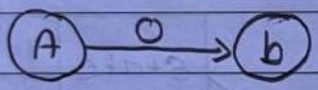
→ Initial state : It is starting point of our system. It is usually drawn with arrow pointed to state.

Finite state → (A)

Camlin Page
Date / /

→ Final State : It is a subset of known state that indicates weather the input we processed is valid or not. Accepting state usually drawn as a double circle.

→ Transition : The machine moves from one state to another and is indicated as transition. These are drawn as two states connected with a line.



Camlin Page
Date / /

Ch: 3 Context Free Grammar

✡ Definition of context free Grammer (Type: 2)

→ CFG stands for context free grammer. It is formal grammer which is used to generate all possible patterns of strings in a given formal language.

→ The CFG can be formally defined as a set denoted by $G = (V, T, P, S)$ where V and T are set of non terminal and terminals respectively. P is set of production rules, where each production rule is form of,

Non terminal to Non terminal or
Non terminal to terminal.

$G = (V, T, P, S)$

- G is the grammar, which consists of a set of production rule.
- T is the final set of a terminal symbol. It is denoted by lower case.
- P is a set of production rules.
- S is the start symbol which is used to derive the string.

Ex. Construct CFG for following language.

* $L = a^n b^{2n}$, where $n \geq 1$

→ $S \rightarrow aSbb \mid abb$

* $L = \{a^x b^y \mid x \neq y\}$

→ $S \rightarrow aSb \mid R_1 \mid R_2$
$R_1 \rightarrow aR_1 \mid a$
$R_2 \rightarrow bR_2 \mid b$

* $L = wcw^T$, where $w \in \{a, b\}^*$

→ $S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow c$

* R.E = $(110 + 11)^* (10)^*$

→ r.e = $\underbrace{(110 + 11)^*}_{A} \underbrace{(10)^*}_{B}$

$S \rightarrow AB$
$A \rightarrow 110A \mid 11A \mid 110 \mid 11 \mid \varepsilon$
$B \rightarrow 10B \mid \varepsilon$

Camlin Page
Date / /

* $L = \{ 0^i 1^j 0^k \mid j > i+k \}$

→ $L = \underbrace{0^i}_{A} \underbrace{1^i}_{} \underbrace{1^k}_{B} \underbrace{1^p}_{C} 2^k$

$S \rightarrow ABC$
$A \rightarrow 0A1 \mid \varepsilon$
$B \rightarrow 1B \mid \varepsilon 1$
$C \rightarrow 1C2 \mid \varepsilon$

* $L = \{ 0^i 1^j 2^k \mid i = j \}$

→ $S \rightarrow AB$
$A \rightarrow 0A1 \mid \varepsilon$
$B \rightarrow 2B \mid \varepsilon$

* $L = \{ a^i b^j c^k \mid i = j \text{ or } j = k \}$

→ Here two condition (i) $i = j$ (ii) $i = k$,

$S \rightarrow AB \mid XY$
$A \rightarrow aAb \mid \varepsilon$
$B \rightarrow cb \mid \varepsilon$
$X \rightarrow aX \mid \varepsilon$
$Y \rightarrow bYc \mid \varepsilon$

* $L = a^* b^*$

→ $S \rightarrow AB$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow bB \mid \varepsilon$

Camlin Page
Date / /

* $L = \{a^{n+2} b^n \mid n \geq 0\}$

→ $S \rightarrow aSb \mid aa$

* $L = (0+1)1^*(1+(01)^*)$

→ r.e = $\underbrace{(0+1)}_{A}\ \underbrace{1^*}_{B}\ \underbrace{(\underbrace{1}_{D} + \underbrace{(01)^*}_{E})}_{C}$

$S \rightarrow ABC$
$A \rightarrow 0 \mid 1$
$B \rightarrow 1B \mid \varepsilon$
$C \rightarrow D \mid E$
$D \rightarrow 1$
$E \rightarrow 01E \mid \varepsilon$

* (i) $\{w \mid w$ contains at lest three 1's$\}$
(ii) $\{w \mid w$ starts and end with same symbol$\}$

→ (i) r.e = $(0+1)^* 1 (0+1)^* 1 (0+1)^* 1 (0+1)^*$
$S \rightarrow A1A1A1A$
$A \rightarrow 0A \mid 1A \mid \varepsilon$

(ii) r.e = $[0(0+1)^*0] + [1(0+1)^*1]$

$S \rightarrow 0A0 \mid 1A1$
$A \rightarrow 0A \mid 1A \mid \varepsilon$

Camlin Page
Date / /

☆ Regular Grammer

→ Regular language can be generated by regular grammer.

Regular grammer is defined as,
$G = (V, T, P, S)$ where,

→ V is set of symbol called non terminals which are used to define rule.
→ T is set of symbol called terminals.
→ P is set of production rule.
→ S is start symbol.

→ When grammer can be represented by some finite automata then it is regular grammer.

→ There are two types of Regular Grammer.

(i) Right linear : If the non terminal symbol appears as a rightmost symbol in each production of regular grammer then it is right linear grammer.
Ex. $A \to aB$, $A \to a$, $A \to \varepsilon$

(ii) Left linear : If the non terminal symbol appears as left most symbol in each production of regular grammer then it is called left linear regular grammer.

Ex. Let, G be the grammer

$S \to aB \mid bA$

$A \to a \mid aS \mid bAA$

$B \to b \mid bS \mid aBB$

For string aaabbabbba find left most and right most derivation.

→

Left most	Right most
S	S
$a\underline{B}$	$a\underline{B}$
$aa\underline{B}B$	$aaB\underline{B}$
$aaa\underline{B}BB$	$aaBb\underline{S}$
$aaab\underline{B}B$	$aaBbb\underline{A}$
$aaabb\underline{S}B$	$aaBbba$
$aaabba\underline{B}B$	$aaaBBbba$
$aaabbaa\underline{B}B$	$aaaBbbba$
$aaabbaab\underline{B}$	$aaabSbbba$
$aaabbaabb\underline{S}$	$aaabbAbbba$
$aaabbaabbb\underline{A}$	$aaabbabbba$
$aaabbabbba$	

Ex. Consider Grammer : $S \to aAs \mid a$

$A \to sbA \mid ss \mid ba$

Derive left and right most for string aabbaa

→

Left most	Right most
S	S
aAS	aAS
aSbAS	aAa
aabbaS	aSbAa
aabbaa	aSbaa
	aabbaa

Camlin Page
Date / /

Ambiguous and Unambiguous Grammar

→ A grammer is said to be ambiguous if there exists more than one left most derivation or more than one right most derivation or more than one parse tree for given input.

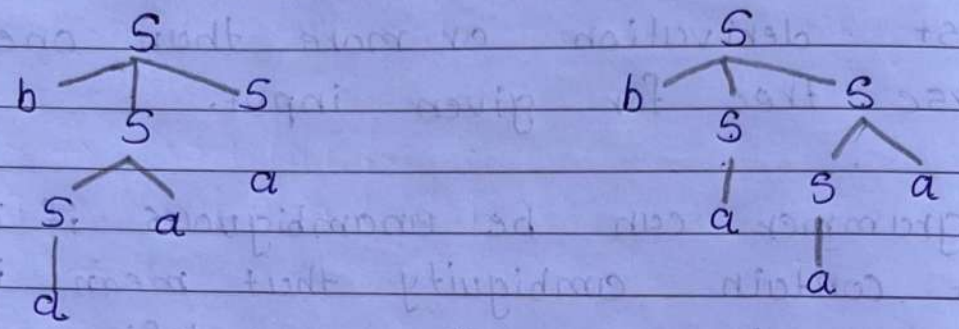
→ A grammer can be unambiguous if it does not contain ambiguity that mean it does not contain more than one left and right most derivation and parse tree for input.

*	Ambiguous Grammer	Unambiguous Grammer
→	It generates more than one parse tree.	It generates exactly one parse tree.
→	The left and right most derivation represent different parse tree.	The left and right most derivation represent Same parse tree.
→	It contains smaller number of non-terminal.	It contains a greater number of non-terminal.
→	length of parse tree is less.	length of parse tree is more.

Camlin Page
Date / /

Ex. Show that CFG : $S \to a \mid Sa \mid bSS \mid SSb \mid SbS$ is ambiguous.

→ Consider string baaa



→ There is two different parse tree is generated then it is ambiguous.

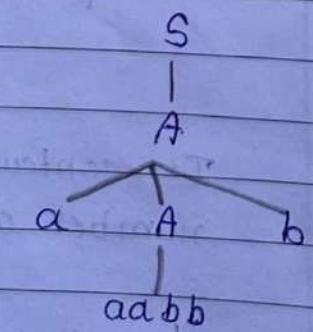
Ex. Check weather ambiguous or not.

$S \to A \mid B$

$A \to aAb \mid aabb$

$B \to abB \mid \varepsilon$

→ Consider string aaabbb



Camlin Page
Date / /

☆ Chomsky's Normal Form (CNF)

→ CNF stands for Chomsky's Normal form.
A CFG is in CNF if all production rules satisfy one of the following conditions:

- Start symbol generating ε.
  $A \rightarrow \varepsilon$
- A non-terminal generating two non-terminals.
  $S \rightarrow AB$
- A non-terminal generating a terminal.
  $S \rightarrow a$

Ex. Convert following CFG to CNF
$S \rightarrow aAbB$, $A \rightarrow Ab|b$, $B \rightarrow Ba|a$

→

Production rule	CNF
$S \rightarrow aAbB$	$S \rightarrow XY$
	$X \rightarrow aA$
	$Y \rightarrow bB$
	$X \rightarrow PA$
	$P \rightarrow a$
	$Y \rightarrow QB$
	$a \rightarrow b$
$A \rightarrow Ab$	$A \rightarrow AQ$
$A \rightarrow b$	already in CNF
$B \rightarrow Ba$	$B \rightarrow BP$
$B \rightarrow a$	already in CNF

Camlin Page
Date / /

Ex. Convert following CFG to CNF.

$S \to ASA \mid aB$
$A \to B \mid S$, $A \to b \mid \varepsilon$

→ We should eliminate ε and unit production,

$S \to ASA \mid aB \mid a$
$A \to b \mid ASA \mid aB \mid a$
$B \to b$

$S \to ASA$	$S \to AP$ $P \to SA$
$S \to aB$	$S \to QB$ $Q \to a$
$S \to a$	
$A \to b$	
$A \to ASA$	$A \to AP$
$A \to aB$	$A \to QB$
$A \to a$	
$B \to b$	

Camlin | Page
Date / /

Ex. Convert following CFG into CNF.

S → aY | Ybb | Y
X → ε | a
Y → aYY | bb | XXa

→ We should eliminate ε and unit production

S → aY | Ybb | aYY | bb | XXa

S → aY | Ybb | aYY | bb | a | Xa | XXa
X → a
Y → aYY | bb | Xa | a

→ S → aY	S → XY
	X → a
S → Ybb	S → YZ
	Z → BB
	B → b
S → aYY	S → XT
	T → YY
S → bb	S → BB
S → XXa	S → PX
	P → XX
X → a	
Y → aYY	Y → XQ
	Q → YY
Y → bb	Y → BB
Y → Xa	Y → XX
Y → a	

CFG to CNF

* $S \to S(S) \mid \varepsilon$

→ Eliminate ε,

$S \to (S) \mid S(S) \mid S() \mid ()$

$S \to (S)$	$S \to AB$
	$A \to ($
	$B \to ST$
	$T \to)$
$S \to S(S)$	$S \to SQ$
	$Q \to AB$
$S \to S()$	$S \to SR$
	$R \to ()$
$S \to ()$	$S \to ()$

* $S \to aX \mid Yb$

$X \to S \mid \varepsilon$

$Y \to bY \mid b$

→ Eliminate ε,

$S \to aX \mid a$

$S \to Yb$

$X \to aX \mid a$

$Y \to bY$

$Y \to b$

$S \to aX$	$S \to AX$
	$A \to a$
$S \to a$	
$S \to Yb$	$S \to YB$
	$B \to b$
$X \to aX$	$X \to AX$
	$A \to a$
$X \to a$	
$Y \to bY$	$Y \to ZY$
	$Z \to b$
$Y \to b$	

Camlin Page
Date / /

Ch : 4

PDA, CFL and NCFL

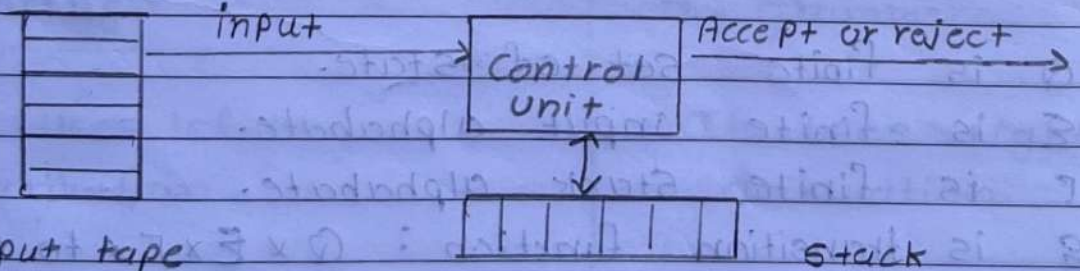
☆ What is PDA ?

-> PDA stands for pushdown automata.

-> PDA is a way to implement a context free grammer (CFG) in a similar way to design the DFA for regular grammer.

-> A DFA can remember a finite ammount of information but a PDA can remember an infinite ammount of information.

-> PDA has three component, Input tape, control unit, stack with infinite size.



-> PDA can be defined as 7-tuples, $(Q, \Sigma, S, \delta, q_0, I, F)$ where $S = \Gamma$ and $I = Z$

- Q is a finite number of state.
- Σ is the input alphabate.
- $S(\Gamma)$ is stack symbol.
- δ is transition function : $Q \times \{\Sigma \cup \{\epsilon\}\} \times S \times Q$
- q_0 is the initial state.
- $I(Z)$ is initial state top symbol.
- F is set of accepting state.

Camlin Page
Date / /

☆ What is deterministic PDA ?

→ A DPDA is type of PDA that can be in only one state at a time for a given input symbol and top of stack symbol. A DPDA is a formal model of computation that recognizes context free languages.

→ A DPDA differes from NPDA in that DPDA only make one possible move for a given input symbol and top of stack symbol, wheares the NDPA can make multiple possible moves. This property makes DPDA to easier to andlyze and implement than NPDA.

→ A DPDA is defined as 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, z, F)$

- Q is finite set of state.
- Σ is finite input alphabate.
- Γ is finite stack alphabate.
- δ is transition function : $Q \times \Sigma \times \Gamma$ to $Q \times \Gamma^*$
- q_0 is start state.
- z is the start stack symbol.
- F is the set of accept stack.

Camlin | Page
Date / /

☆ Difference between Top down and Bottom up parsing.

Top down Parsing	Bottom up Parsing
→ It is top down if it discovers a parse tree top to bottom.	It is bottom up if it discovers a parse tree bottom to top.
→ Given grammer G is tries to derive a string through a sequence of derrivation starting with start symbol.	In this the source string is reduced to the start symbol of the grammer. It is also called shift reduce parsing.
→ Top down parsing attemts to find the left most derrivation for given string.	Bottom up parsing attempts to reduce the input string to first symbol of grammer.
→ It uses leftmost derrivation.	It uses rightmost derrivation.

Camlin Page
Date / /

☆ PDA to CFG Conversion

→ Input : A CFG, $G = (V, T, P, S)$
Output : Equivalent PDA, $P = (Q, \Sigma, S, \delta, q_0, z, F)$

Step 1 : Convert the production of the CFG into GNF.

Step 2 : PDA will have only one state {q}

Step 3 : Start symbol of CFG will be start symbol in the PDA.

Step 4 : All non-terminals of the CFG will be the stack symbol of PDA and all the terminals of the CFG will be the input symbol of the PDA.

Step 5 : For each production in the form $A \rightarrow aX$ where a is terminal and A, X are combination of terminal and non-terminal make a transition $\delta(q, a, A)$.

Camlin Page
Date / /

Ch : 5

Turing Machines

☆ Explain Turing Machine.

→ Turing machine is a computation model, like FA, PDA which works on unrestricted grammer. The turing machine is the most powerful computation model compare to FA and PDA.

→ It consists of infinite tape divided into cell, A read-write head that can read and write symbol on tape, and a control unit that determines the next action based on current state and symbol read by the head.

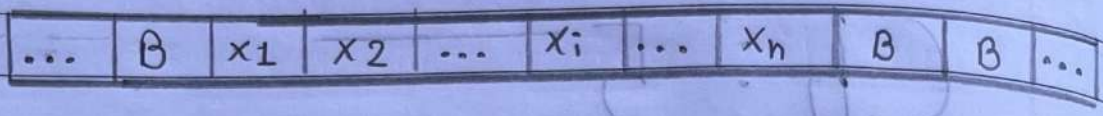
→ Turing Machine M can be defined as :

$M = \{Q, X, \Sigma, \delta, q_0, B \text{ or } \Delta, F\}$

- Q is finite, non-empty set of state.
- X represent set of tape alphabate.
- Σ represent the non-empty set of input alphabate.
- δ is transition function :

$\delta : Q \times X \rightarrow Q \times X \times \{\text{Left-shift, Right-shift}\}$

- q_0 is the initial state of machine.
- B or Δ represent blank symbol.
- F is set of final state or halt state.



-> A Turing machine can have three types of action upon an input.

move one square to the Left and go to state q_j.
move one square to the Right and go to state q_j.
do not move (N) and go to state q_j.

* Advantages

-> TM can remember previous input.
-> TM can output the symbols for corresponding symbol read.
-> TM can act as computer by computing arithmetic function, recognizing well formed paranthesis, reverse or concatanete strings and so on.

Camlin Page
Date / /

☆ Model of Computation of TM. ☆

→ The input tape having infinite number of cells, each cell containing one input symbol and thus the input string can be placed on tape. The empty tape is filled by blank characters.

→ The finite control and the tape head which is responsible for reading the current input symbol. The tape head can move left or right.

→ A finite set of state through which machine has to undergo.

→ Finite set of symbol called external symbol which are used in building the logic of turing machine.

Camlin Page
Date / /

☆ Explain Universal Turing Machine

-> A general purpose computer can be programmed to solve different types of problem.

-> Universal turing machine is a TM that work as follows :

- It is assumed to receive an input string of the form e(T)e(z), where T is an arbitary TM, z is a string over input alphabate of T, and e is an encoding function whose values are strings in {0,1}*. It following two properties :

(i) TU accepts the string e(T)e(z) if and if only T accept z.
(ii) If T accepts z and produce output y, then TU produce output e(y).

-> We can assume UTM as 3-tape TM:

(i) Input is written on first tape.
(ii) Moves of the TM is encoded is written on the second tape.
(iii) The current state of TM is written on third tape.

Camlin Page
Date / /

☆ Church's Turing thesis.

-> It is also known as church's hypothesis.

-> Hypothesis mean proposing certain facts. It can be stated as,

"The assumption that the intuitive notion of computable function can be identified with partial recursive functions."

-> The church turning thesis says that every solvable decision problem can be transfomed equivalent turing machine problem.

-> This hypothesis can not be proved. The computability of recursive function is based on following assumptions.

(i) Each elementary function is computable.

(ii) Let f be the computable function and g be the another function which can be obtained by applying an elementary operation to f, then g becomes a computable function.

(iii) Any function becomes computable if it is obtained by rule (i) and (ii).

Camlin Page
Date / /

☆ Recursive and Recursive enumerable language.

-> Recursive Language :

A Recursive language is a formal language that can be recognized by a turing machine that always halt, either by accepting and rejecting a given input string. In other words a recursive language is one that can be decide by turing machine, meaning that the machine can determine with certainly weather given input string is member of the language or not.

-> Recursive Enumerable Languages :

It is a formal language that can be recognized by turing machine that may either halt and accept a given input string, or loop idefinitely without halting. In other words, This language is one of that can be recognized by a turing machine that can accept any input string that is member of the language but may not necessarily halt if the input string is not a member of the language.

* **Decidable Language** : If a language is recursive then it is called decidable languages.

* **Undecidable Language** : If a language is non-recursive then it is called undecidable language.

☆ Context Sensitive Language

→ These are the languages which are defined by context sensitive grammer. In this grammer more than one terminal or non terminal symbol may appear on the left hand side of the production rule.
Context Sesitive gramer follows following rules:

(i) The number of symbol on the left hand side must not exceed number of symbol on the right hand side.

(ii) The rule of the form $A \rightarrow \varepsilon$ is not allowed unless A is a start symbol. It does not occure on the right hand side of any rule.

Camlin Page
Date / /

☆ Chomsky Hierarchy

→ The Chomsky's heirarchy represent the class of languages that are accepted by different machine.

Language class	Language	Grammer	Machine	Ex.
Type-3	Regular	Regular	NFA or DFA	a^*b^*
Type-2	Context free	Context free	PDA	$a^n b^n$
Type-1	Decidable	Context Sensitive	Linear bounded automata	$a^n b^n c^n$
Type-0	Computable	Unrestricted	Turing Machine	$n!$

* Type-3 - Regular Languages

→ Regular languages are those language which can be describe using regular expression. These language can be modelled by NFA or DFA.

* Type-2 - Context Free Languages.

→ These are the languages which can be represented by context free grammar.

Camlin Page
Date / /

→ The production Rule is of the form,

$$A \rightarrow \alpha$$

Where, A is any single non terminal and α is any combination of terminals and non-terminals.

→ NFA and DFA can not recognize strings of these language because these automata are not having stack to memorize. PDA can be used to represent these language.

* Type-1 - Context Sensitive Language

→ These are ~~Used to~~ represented by context sensitive grammar. It follows following rule:

(i) It may have more than one symbol on the left hand side of production rule.

(ii) Number of symbols on the left hand side must not exceed the number of symbol on right hand side.

(iii) The Rule of the form $A \rightarrow \varepsilon$ is not allowed unless A is start symbol. It does not occure on the right hand side of any rule.

* Type-0 - Unrestricted Language

→ There is no restriction on the grammar rules of these type of languages. These languages can be effectively modeled by Turing Machines.

Camlin Page
Date / /

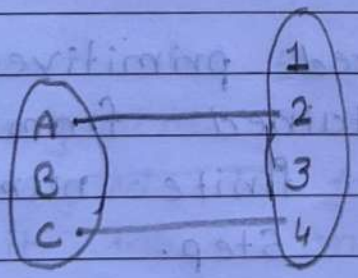
Ch: 6 Computable Functions

☆ Partial, Total and constant Functions

→ The problem of finding out weather given problem is solvable or not is based on evalution of function on the set of natural number by given alphabates.

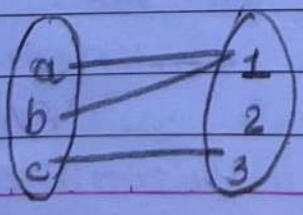
* Partial Function

→ A partial function f from A to B is rule which assigns every element of A to the most one element of B.



* Total Function

→ A total function f from A to B is a rule which assigns to every element of A a Unique element of B.



Camlin Page
Date / /

* Constant Function

→ A constant function $f(x)$ over n and every k (natural number) using recursion can be defined as

$$f(0) = k$$
$$f(n+1) = g(n, f(n))$$

Ex.

(i) $z(x) = 0$ zero function
(ii) $f(x) = x$ Identity function

☆ Primitive Recursive Function

→ Function is considered primitive recursive if it can be obtained from initial functions and through finite number of composition recursive step.

→ Recursive function is class of function those are turing computable.

→ The thoery of recursive function is just converse to Church's hypothesis.

Camlin Page
Date / /

→ Recursive function theory begins with some very elementary functions that are intutively effective, then it provides few method for building more complicated function from simpler function.

→ The initial functions are the elementary function whose values are independent of their smaller arguments.

- The zero function : $Z(x) = 0$
- The successor function : $S(x)$ = successo of x, $(x+1)$
- The identity function : $id(x) = x$

→ The zero function returns zero regardless of its argument.

→ The successor function returns successor of its argument.

→ The zero and successor function takes only one argument each. But the identity function is designed to take any number of argument. When it takes one argument it return it argument as its value. When it takes two argument it returns one of them as its value,

$$id(x, y) = x$$
$$id(x, y) = y$$

→ We will buid more complex function from the initial set by using three methods that are,

(i) Composition
(ii) Primitive Recursion
(iii) Minimization

Camlin Page
Date / /

Ch: 7 Undecidability

☆ Undecidable Problems about TM

* Halting Problem

→ Usually, program consists of loops that are limited or unlimited length.

→ The total work done by the program completely depends on the input given to the program.

→ The program may consists of servel different number of loops that may be in linear or nested manner.

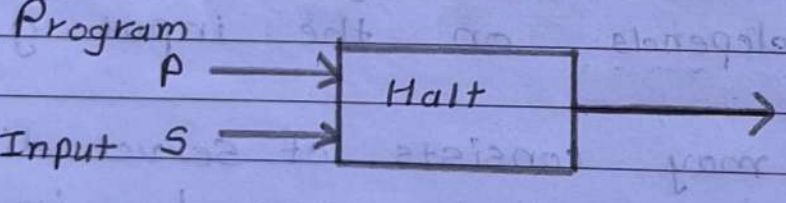
→ The halting problem is the problem of deciding or concluding based on given arbitary computer program and its input, wedther that program will stop executing or run - in an infinite loop for the given input.

→ Halting problem tells that it is not easy to write computer program that executes in the limited time that is capable of deciding weather program halts for an input.

→ In addition to that the halting problem never says that it is not practicable to determine weather a given random program is going to halt.

→ Generally, it is important because it shows that there are some problems that are undecidable, meaning that there is no algorithm that can solve them for all possible inputs.

→ Diagrammatic Representation :



Camlin Page
Date / /

* Post's Correspondence Problem.

→ In this section we will discuss the undecidability of strings and not of Turing Machines. The undecidability of strings is determined with the help of post correspondence problem (PCP).

→ The post correspondence problem consists of two list of strings that are of equal length over the input Σ.
The two lists are $A = w_1, w_2, w_3, \ldots, w_n$
$B = x_1, x_2, x_3, \ldots, x_n$ then there exists an non empty set of integer $i_1, i_2, i_3, \ldots, i_n$ such that, $w_1, w_2, w_3, \ldots w_n = x_1, x_2, x_3, \ldots x_n$.

→ To solve post correspondence problem we try all the combinations of $i_1, i_2, i_3, \ldots, i_n$ to find the $w_i = x_i$ then we say that PCP has a solution.

Camlin Page
Date / /

☆ Compare Push down automata and Turing Machine.

→ Both PDA and TM accepts accepts regular as well as context free languages. Both the language can remember already read input.

→ The TM can be programmed hence it accepts larger class of language than PDA.

→ We can write an output on input tape in TM. But in the case of simple PDA it is not possible to produce output.

☆ Enlist Limitation of Turing Machine

→ Following problem can not solved by TM.

(i) Determining if program will ever halt on given input.
(ii) Determining weather two programs will produce the same output.
(iii) Determining weather program will accept the given input
(iv) Determining behaviour of other TM.

Camlin Page
Date / /

What are P and NP class in TOC

P - class

-> The p consist of those problems that are solvable in polynomial time.

-> It can be solved in time $O(n^k)$, in the worst-case, where k is constant.

-> These types of problems are called tractable and other are called intractable or super polynomial.

-> Generally, an algorithm is a polynomial time algorithm, if there exists a polynomial p(n) such that the algorithm can solve any instance of size n in a time $O(p(n))$

-> The advantage in the class of polynomial time algorithm is that all reasonable deterministic single processor model of computation can be simulated on each other with at most a polynomial slow-d.

Camlin Page
Date / /

* NP - class

-> It stands for non-deterministic polynomial time.

-> The class NP consists of problems that are verifiable in polynomial time.

-> NP is a class of decision problem for which it is easy to check the correctness of a given answer with the aid of little extra information.

-> Hence we are not asking for a way to find solution, but only verify that a solution is correct.

-> Any problem for which answer is either yes or No is called decision problem. The algorithm for decision problem is called decision algorithm.

-> Any problem that contains of identification of optimal cost is called optimization problem. The algorithm for optimization problem is called optimization algorithm.

Camlin Page
Date / /

☆ Differences between P and NP class.

	P class	NP class
→	Problems that can be solved in polynomical time.	Problems that can be solved in non-deterministic polynomical time.
→	These are simple to solve.	These are complex to solve.
→	P problems are subset of NP problems.	NP problems are superset of P problems.
→	All the problems are deterministic in nature.	All the problems are non-determinstic in nature.
→	Ex. Selection sort, Linear Search	Ex. knapsack problem