Theory of Computation

Published By:

Physics Wallah

ISBN: 978-93-94342-39-2

Mobile App: Physics Wallah (Available on Play Store)

Website: www.pw.live

Email: support@pw.live

Rights

Disclaimer

A team of PW experts and faculties with an understanding of the subject has worked hard for the books.

While the author and publisher have used their best efforts in preparing these books. The content has been checked for accuracy. As the book is intended for educational purposes, the author shall not be responsible for any errors contained in the book.

The publication is designed to provide accurate and authoritative information with regard to the subject matter covered.

(This Module shall only be Used for Educational Purpose.)

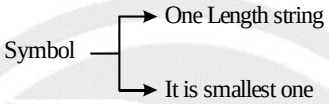
Theory of Computation

INDEX

1 BASICS OF THEORY OF COMPUTATION

1.1 Symbol

Symbol represents very unique in the world. Any small thing that never be broken into any other is called as symbol.



1.2 Alphabet (∑)

It is a set of finite number of symbols.

Example:

- English alphabet = {a, b, ... z}
- Binary alphabet = {0, 1}
- Decimal alphabet = {0, 1, 2, ... 9}
- We can create our own alphabet = {gate, cs, it, exam}
  where gate, cs, it, exam all are symbols

Alphabet (Σ) = {gate, cs, it, exam}
{ w , x , y , z }

1.3 String

- It is sequence of symbols defined over given alphabet.
  let ∑ = {a, b}
- Strings possible are ∈, a, b, aa, bb……
- Strings over English Alphabet: deva, gate, exam, etc.
- Strings over Binary Alphabet: 0010, 1011, 1101, 1111, etc.
- Strings over Decimal Alphabet: 3012, 2345, 5438, etc.

Note:

- Different length strings over given alphabet
  I Zero length string → Empty string / Null string ∈ or λ is used to denote empty string length of empty string. |∈| = 0.
  II One length string over ∑ = {a, b} are a, and b (2 strings).
  III Two length strings over ∑ = {a, b} are da, ab, ba, and bb (4 strings).

1.4 Operations on Strings

There are various operations on strings:

- Unary and Binary operations

1. **length of a string:** Length of a string is denoted as |w| and is defined as the number of positions for the symbol in the string.

   Example: w = aba
   |w| =|aba| = 3

2. **Reversal of a string:** It will reverse or changes the order of a given string w.

   Example: w = abb
   w^R = bba

1.4.1 Concatenation of Two Strings

Given two strings w_1 and w_2, we define the concatenation of w_1 and w_2 to be the string as w_1w_2.

Example:

w_1 = a, and w_2= ba

Then w_1w_2 = a.ba = aba.

1.4.2 Prefix of a String

A substring with the sequence of beginning symbols of a given string is called a "prefix".

Example:

(i) w = aaaa

Prefixes = {∈, a, aa, aaa, aaaa}

(ii) w = abcd

Prefixes = {∈, a, ab, abc, abcd}

Note:

If |w| = n then $|w^{Rev}|$ = n.

If length of w_1 is n_1 and length of w_2 is n_2 then length of w_1w_2 = $|w_1 w_2|$ = $n_1 + n_2$.

- Let w be n length string.

(i) Number of prefixes in w = n+1

(ii) Number of non-empty prefixes of w = n

(iii) Number of different length prefixes of w = n + 1

(iv) Number of different length prefixes of w excluding zero length = n

1.4.3 Suffix of a String

A substring with the sequence of ending symbols of a given string is called a "suffix".

Example:

(i) w = aaaa

Suffixes = {∈, a, aa, aaa, aaaa}

(ii) w = abcd

Suffixes = {∈, d, cd, bcd, abcd}

Note :

- Let w be n length string.
  (i) Number of suffixes of w = n + 1
  (ii) Number of non-empty suffixes of w = n
  (iii) Number of different length suffixes of w = n + 1
  (iv) Number of different length suffixes of w excluding zero length = n

1.4.4 Substring of a String

- Let w be n length string.

(i) Number of substrings of w
- minimum = n +1 (over 1 symbol)
- maximum = 1+ n + (n – 1) + ...+1 (all characters distinct) =1 + ∑n = ∑n +1

(ii) Number of non-empty substrings of w
- minimum = n
- maximum = ∑n

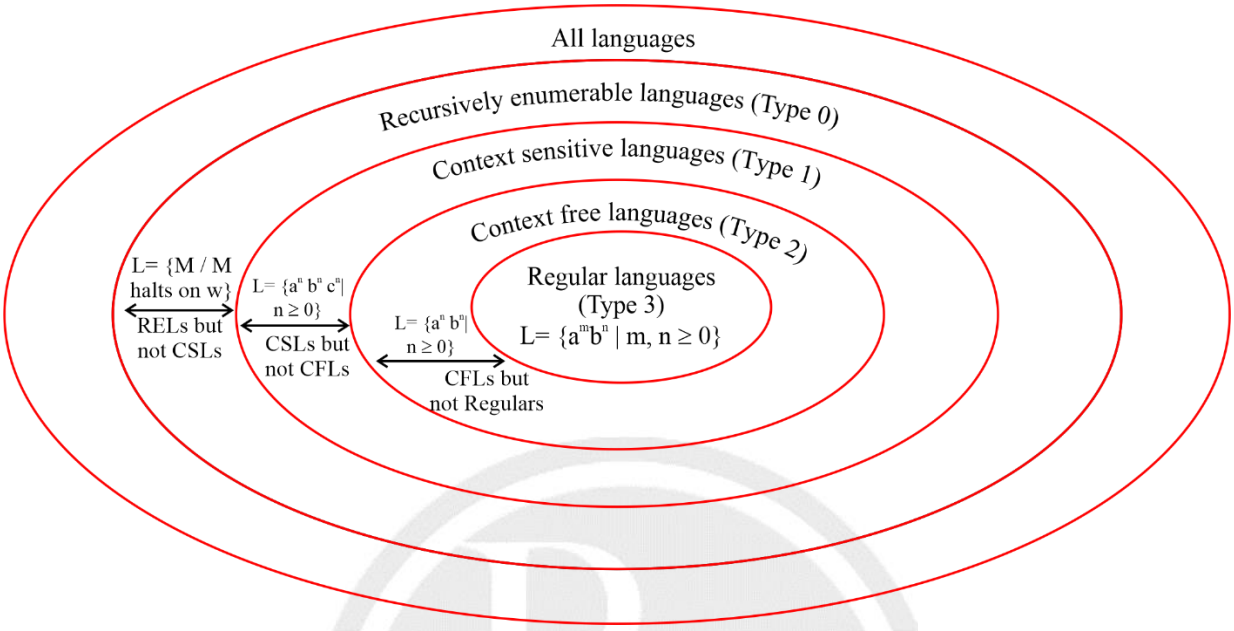
- Number of different length substrings of any given n length string =n+1
- Number of different length substrings of any given n length string excluding zero length=n.

1.5 Relation between Symbol, Alphabet, String and Language

Character	Symbol
⇓	⇓
Character set	Alphabet
⇓	⇓
Token	String
⇓	⇓
C Language	Language

1.5.1 Chomsky Hierarchy

- Chomsky hierarchy includes all the problems in the world classified into classes.



- Type 3 is the smallest class, and Type 0 is the biggest class.

	Type 3	Type 2	Type 1	Type 0
Language:	Regular	Context free	Context sensitive	Recursively Enumerable
Automata:	Finite Automata	Push down Automata	Linear Bounded Automata	Turing Machine
Grammar:	Regular	Context free	Context Sensitive	Unrestricted

1.6 Language

- Language is set of strings defined over alphabet (Σ).
- Let $\Sigma = \{a,b\}$. Then $\Sigma = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup$ …….
  = set of all strings
  = universal language
  = $\{\in, a, b, aa, ab, ba, ...\}$
- $\Sigma^2 = \Sigma^1 \bullet \Sigma^1 = \{a,b\}.\{a,b\} = \{aa, ab, ba, bb\}$
- Language: It is a subset of Σ^*

$\therefore$ $\boxed{L \subseteq \Sigma^*}$

- A language is a collection of strings that must be a subset of Σ^* where Σ^* is a universal language.

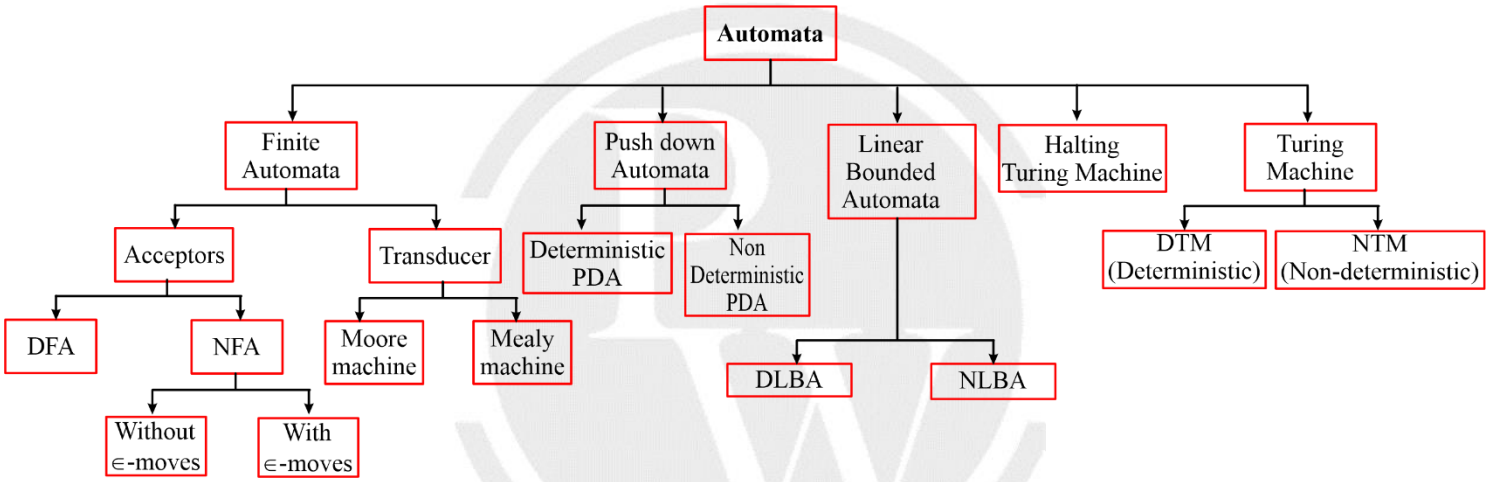
Example:

L = ab* = {a, ab, abb,.....}

1.7 Types of Languages

- Finite Language
- Infinite Language
- Regular language
- DCFL (Deterministic CFL)
- CFL
- CSL
- Recursive Language
- Recursive Enumerable Language (REL)

1.8 Types of automata



- Ralations between these machines:

$$FA < DPDA < PDA < LBA < HTM < TM$$

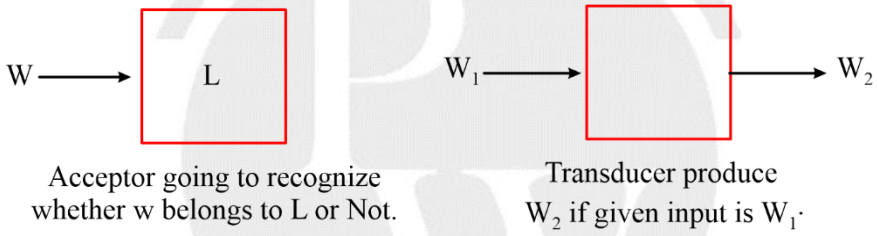
1. **Less power:** It can Represent less number of languages.
2. **More power:** It can Represent more number of languages Compare to all these machines.

❐❐❐

2 FINITE AUTOMATA

2.1 Introduction

- Finite Automata describes or represents a regular language.
- Finite Automata is of two types:
  (i) Acceptor: Accepts or rejects given string.
  (ii) Transducer: Produces output string for given input string.



Acceptor going to recognize whether w belongs to L or Not.

Transducer produce W_2 if given input is W_1.

Example:

1's complement of binary number: Input = 10100, Output = 01011

2.2 Finite Acceptor (Finite Automata)

- Finite Automata:

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

- F → Set of final states
- q_0 → Initial state
- δ → Transition function
- Σ → Set of input symbols
- Q → Set of finite number of states

Transition function (δ)

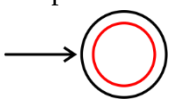
(1) DFA : ⟶ $\delta: Q \times \Sigma, \rightarrow Q$

(2) NFA: ⟶ Without $\in-$ moves $\delta: Q \times \Sigma, \rightarrow 2^Q$

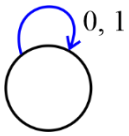
⟶ With $\in-$ moves $\delta: Q \times \Sigma \cup \{\in\} \rightarrow 2^Q$

2.3 Construction of DFA

1. If epsilon belongs to L, then initial state must be final in DFA.

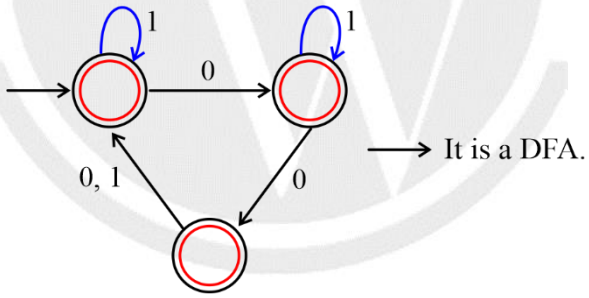


2. **Dead state:** It is non-final state but it never contain a path to final.

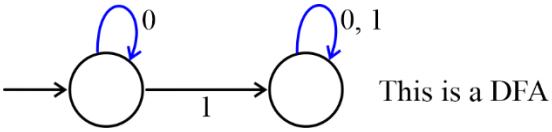


- Once we reach the dead state, there is no way to reach to the final state.
- If every state is final then every string accepted in the DFA.

3. If all states are finals in DFA then L(DFA) = $\sum^*$



4. If every state is non final in a DFA then L(DFA) = ϕ or L(DFA) = {},



$\boxed{\overline{L} = \sum^* - L}$

$\boxed{L \cup \overline{L} = \sum^*}$

$\boxed{L \cap \overline{L} = \phi}$

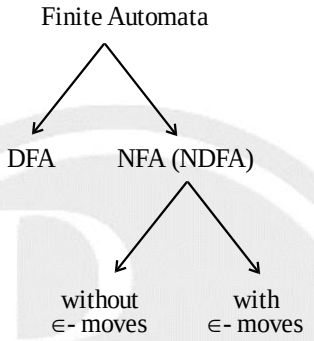
- If $|w| = n$ or $|w| \leq n$, then number of states in minimum DFA = (n + 2) states
- If $|w| \geq n$ then number of states in minimum DFA = n+ 1 states.
- Start condition, exactly, atmost length question requires Dead state but end condition, contain substring, atleast length questions do not require dead state.

- Over one symbol $\sum = \{a\}$. Length of string is equal to number of a's in string. $|w| = n_a(w)$.
- Let $L=\{w|w \in \{a,b\}^*, n_a(w)$ is divisible by m, $n_b(w)$ is divisible by n$\}$. Then Number of states in DFA= m×n.
- Let L={w | w belongs to {a. b}*, n^{th} symbol of w from begin is 'a'}. Then Number of states in DFA = (n + 2).
- Number of equivalence classes for any regular set L = Number of states in minimal DFA that accepts L.

2.4 Non-Deterministic Finite Automata

2.4.1 Introduction

- Finite Automata can be designed in two ways:



- All these finite state machines are equivalent. DFA ≅ . NFA
- We can convert one finite state machine to any other finite state machine.
- For every gular language, we can design infinite equivaent DFAs or infinite equivalent NFAs but minimum DFA is uique for given regular language.
- For every regular language, one or more minimum NFAs may exist.

Note : For every regular language:

(i) Unique minimum DFA exists.

(ii) One or more minimum NFAs exists.

2.5 Comparison of NFA and DFA

	$\|w\| = n$	$\|w\| \le n$	$\|w\| \ge n$
Number of states in NFA	n + 1	n + 1	n + 1
Number of states in DFA	n + 2	n + 2	n + 1

- If every string has n^{th} symbol from begin is 'a' over binary alphabet {a, b} then (n + 1) states in minimum NFA but (n+2) states in minimum DFA.

- If every string has n^{th} symbol from end is 'a' over binary alphabet {a, b} then 2^n states in minimum DFA and (n + 1) states in minimum NFA.

2.5.1 COMPARISON OF DFA AND NFA (NFA vs DFA)

	NFA	DFA
(1) Transition Function (δ)	$Q \times \Sigma \rightarrow 2^Q$	$Q \times \Sigma \rightarrow Q$
(2) Number of paths for string	For valid string: 1 path For invalid string: >=0 paths	For valid string: 1 path For invalid string: 1 path

2.5.2 Number of states in DFA and NFA for Regular Languages

Language	NFA states	DFA states
(1) {w | w ∈ {a, b}*, |w| = n}	n + 1	n + 2
(2) {w | w ∈ {a, b}*, |w| ≤ n}	n + 1	n + 2
(3) {w | w ∈ {a, b}*, |w| ≥ n}	n + 1	n + 2
(4) {w |w ∈ {a, b}*, w starts with a}	2	3
(5) {w| w∈ {a, b}*, n^{th} symbol from begin is a}	n + 1	n + 2
(6) {w| w ∈ {a, b}*, n^{th} symbol from end is 'a'}	n + 1	2^n

2.5.3 Finding number of states in DFA using NFA

NFA (n states)

⇓ Subset Construction

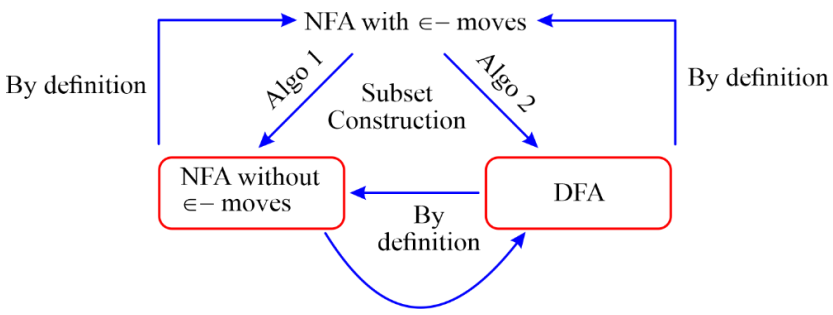
DFA (2^n sates exists)

⇓ Partition algorithm

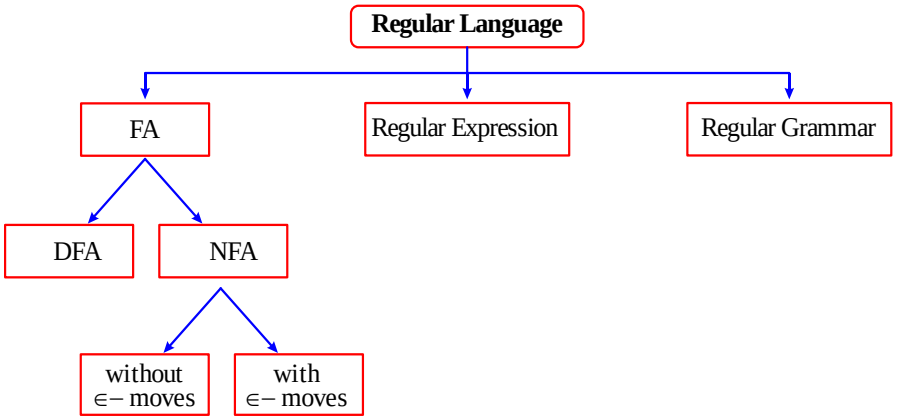
Minimum DFA (≤ 2^n states) atmost 2^n states

- Every DFA is NFA, but NFA need not be DFA.

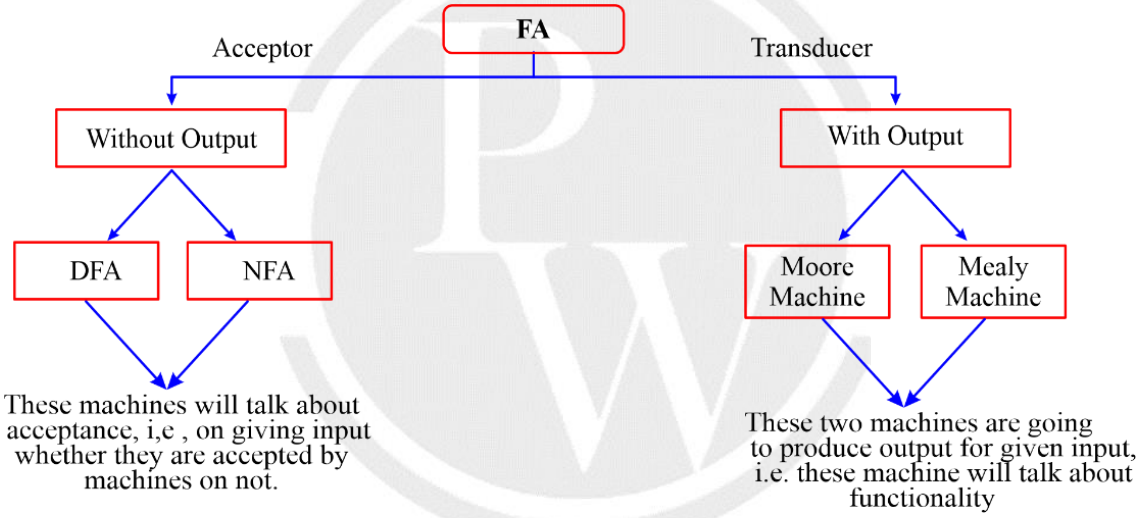
2.5.4 Relation between NFA with epsilon, NFA without epsilon, and DFA



2.6 Regular language Representation



2.7 FA Classification

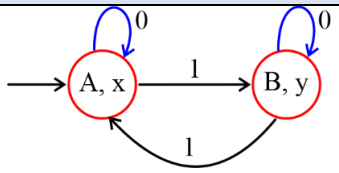


2.8 Moore machine and mealy machine

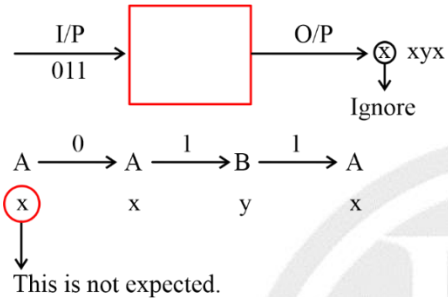
Moore machine	Mealy machine
Transition Function $\delta : Q \times \Sigma \rightarrow Q$ Output Function $\lambda : Q \rightarrow \Delta$ Outpput is asociated with every state.	$\delta : Q \times \Sigma \rightarrow Q$ $\lambda : Q \times \Sigma \rightarrow \Delta$ Output is associated with transition.

2.8.1 Example for Moore machine and Mealy machine

Moore

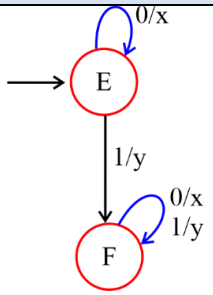


Q = {A, B}
∑ = {0, 1}
Δ = {x, y}
Outpput is asociated with state.

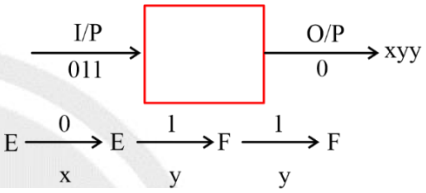


- Extra one output is produced other than desired output.
- So, we can ignore this extra output as this is the machine property.

Mealy



Q = {A, B}
∑ = {0, 1}
Δ = {x, y}
Output is associated with transition.

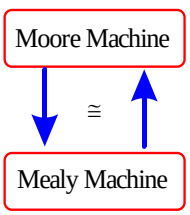


- No extra output is produced.
- For 3 length input, we are getting the 3 length output.

2.9 Difference between Moore machine and Mealy machine

		Moore machine	Mealy machine
1.	δ	$Q \times \Sigma \to Q$	$Q \times \Sigma \to Q$
2.	λ	$Q \to \Delta$	$Q \times \Sigma \to \Delta$
3.	Length of O/p	If n length I/P (assume 1 length O/P symbol is taken at each state) then O/P length is (n+1).	If n length input (assume 1 length O/P symbol is taken at each transition) is given then O/P length is n.
4.	By default	DFA no final state.	DFA no final state.

2.10 Constuction of FA with output



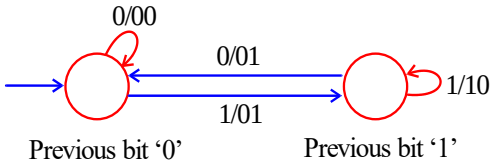
Note :

For every problem, if moore machine exists the we can also construct equivalent mealy machine.

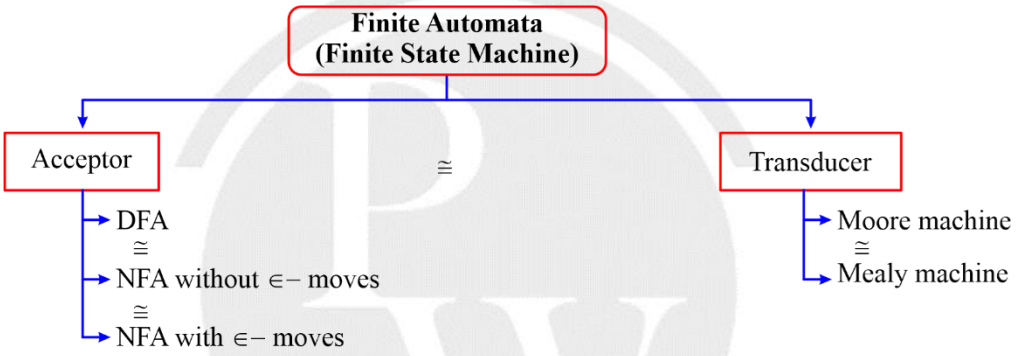
Example:

Sum of present bit and previous bit.

Mealy Machine



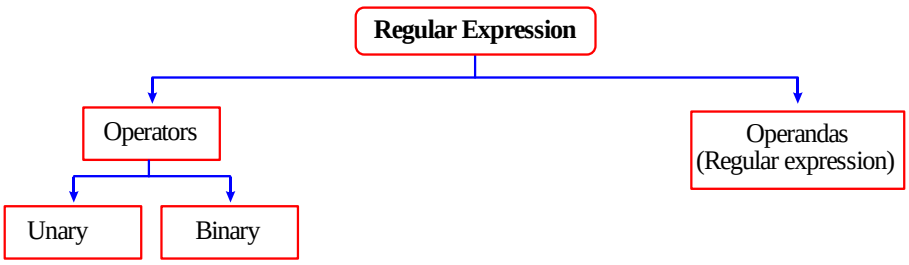
2.11 Classification of Finite State Machine (FSM)



2.12 Regular Expression

Definition:

- Regular expression represents a regular language.
- It describes a regular set.
- L (regular expression) is regular set.
- It is a kind of declarative way to represents a regular language.
- Regular expression generates a regular set.
- It uses 4 operators to represent a regular language.



2.13 Operators of Regular Expression

1. OR (+)
2. Concatenation(.)

} Binary Operator

3. Kleenestar (∗)
4. Kleenestar (+)

} Unary Operator

Regular Expression	Equivalent Regular Set
a + b	L(a + b) = {a, b}
a + a = a	L(a + a) = {a}
a + ϕ = a	L(a + ϕ) = {a}
a + ∈ = ∈ + a	L(a + ∈) = {a, ∈}
∈ + ∈ = ∈	L(∈+∈) = {∈}
ϕ + ϕ = ϕ	L(ϕ + ϕ) = {}
a · b = ab	L (a · b) = {ab}
a · ∈ = a	L (a · ∈) = {a}
∈ · ∈ = ∈	L(∈·∈) = {∈}
ϕ · ϕ = ϕ	L(ϕ · ϕ) = {ϕ}
a · a = aa = a^2	L (a · a) = aa = {a^2}

2.14 Kleene star/ kleene closure / closure

R* (Kleene closure of R):

$$R^* = R^\circ + R^1 + R^2 + R^3 + \ldots = \in + R + RR + RRR + \ldots.$$

Example:

$a^* = \{a^0, a^1, a^2, a^3, \ldots\} = \{\in, a, aa, aaa, \ldots\}$ = Set of all strings over a.

$\phi^* = \phi^0 + \phi^1 + \phi^2 + \phi^3 + \ldots. = \in + \phi + \phi + \phi + \ldots. = \in + \phi = \in$

2.15 Positive Closure (Kleeme Plus)

R^+ (Positive Closure of R):

$R^+ = R^1 + R^2 + R^3 + \ldots..$

$R^* = R^{\geq 0}$ i.e. repeart R any number of time

$R^+ = R^{\geq 1}$ i.e. repeat R atleast 1 time.

$R^* = R^+ + R^0$

Example:

(1) $a^+ = \{a, aa, aaa, \ldots\} = \{a^n \mid n \geq 1\}$

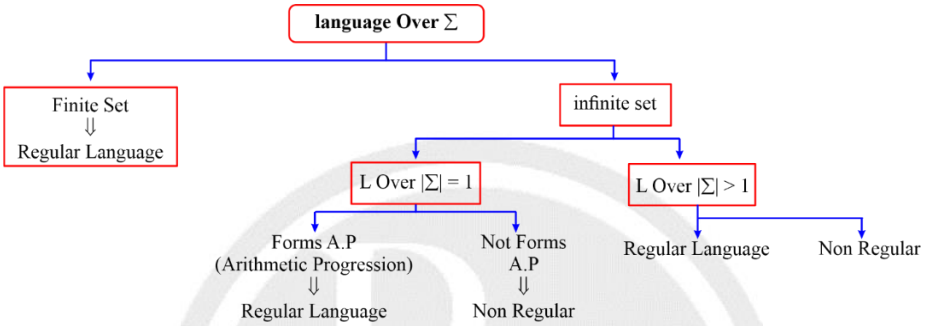
(2) $\in^+ = \in = \in^*$

(3) $\phi^+ = \phi$

Properties :

		OR	Concatenation
1.	Identity	ϕ	$\in$
2.	Associative	Yes	Yes
3.	Commutative	Yes	No
4.	Annihilatior	Σ^*	ϕ

2.16 Identification of Regular Languages



1. $L=\{a^nb^m \mid n, m \geq 0\} = a^*b^*$ (Regular language)
2. $L= \{a^nb^m \mid m<n<10\}$ (Finite Set, Regular language)
3. $L=\{a^nb^m \mid m > n > 10\}$ (Non Regular language)
4. $L= \{a^nb^m \mid m = n, m < 10\}$ (Regular language)
5. $L=\{ a^nb^m \mid m = n, m > 10\}$ (Non Regular language)
6. $L=\{a^mb^n \mid \gcd(m, n) = 1\}$ (Non Regular language)
7. $L=\{a^mb^n \mid LCM(m, n) = 1\}$ (Regular language)
8. $L= \{a^nb^n \mid n >= 0\}$ (Non Regular language)
9. $L=\{a^nb^{2n} \mid n \geq 0\}$ (Non Regular language)
10. $L=\{a^mb^n \mid m=even, n =odd\}$ (Regular language)
11. $L=\{a^mb^n \mid m = n = even\}$ (Non Regular language)

    OR

    $L = \{a^{2n}b^{2n} \mid n = even\}$
12. $L=\{a^*\}$ over $\Sigma= \{a\}$

    = Regular language
13. $L=\{a^{2n} \mid n >= 0\}$ over $\Sigma = \{a\}$.

    $L= a^{2n}=(aa)^*$

    = Regular language

14. L= $\{a^{Prime}\}$ over $\sum = \{a\}$

    = Non regular language

15. L=$\{ a^{n^2} \mid n \geq 0\}$ over $\sum = \{a\}$

    L={epsilon, a, aaaa, a^9, a^{16}...}, FA Not possible for L. So, it's Non-Regular language.

16. L= $\{ a^{2^n} \mid n > 0\}$over $\sum = \{a\}$

    Non Regular language

17. L= $\{ a^{2^n} \mid n \leq 10\}$

    = Regular language

18. L=$\{a^{n!} \mid n \geq 100\}$ over S = {a}

    = Non Regular language

19. L=$\{ a^{n^n} \mid n \geq 10\}$ over $\sum = \{a\}$

    = Non Regular language

20. L = $\{a^{Prime}\}^*$ over $\sum = \{a\}$

    L = complement of {a} = $\sum^* - \{a\} = \{\in, a^2, a^3\ a^4\ , a^5, a^6, a^7, ...\}$= Regular language

21. L = $\{a^{prime} \mid$ prime <100$\}$ is finite language (regular)

22. L = $\{w \# w \mid w \in a^*\}$

    L = $\{a^n \# a^n\}$

    dependency

    = Non Regular language

23. L = $\{w \# w \mid w \in (a+b)^*\}$

    = Non Regular language

24. L = $\{w \# w^R \mid w \in a^*\}$ is non regular

    L = $\{a^n \# a^n\}$

25. L = $\{w\ x\ w \mid w \in \{a, b\}^*, x \in \{a, b\}^+\}$

    Put w = $\in$, and x = $(a + b)^+$

    L = $(a + b)^+$

    = Regular language

26. L = $\{x\ w\ w \mid w \in \{a, b\}^*\ x \in \{a, b\}^+\}$

$L = (a + b)^+$

= Regular language

27. $L = \{w\ w^R\ x \mid w \in \{a, b\}^*, x \in \{a, b\}^+\}$

$L = (a + b)^+$

= Regular language

28. $L = \{w\ x\ w^R \mid w \in (a + b)^*, x \in \{a, b\}^+\}$

$L = (a + b)^+$

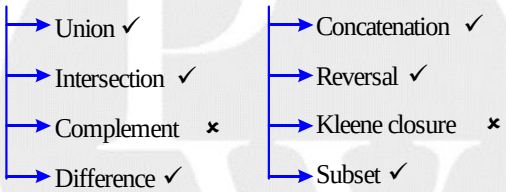
= Regular language

29. $L = \{x\ w\ w^R \mid w \in \{a, b\}^*, x \in \{a, b\}^+\}$

$L = (a + b)^+$

2.17 Closure Properties of Regular Languages

1. Closure properties for finite languages:



2.18 Table for FINITE sets

		Finite sets	Closed/Not Closed
(1)	Union ($\cup$)	$F_1 \cup F_2 \Rightarrow$ Finite	✓
(2)	Intersection ($\cap$)	$F_1 \cap F_2 \Rightarrow$ Finite	✓
(3)	Complement ($\overline{L}$)	$\overline{F} \Rightarrow$ NOT finite	×
(4)	$L_1 - L_2$	$F_1 - F_2 \Rightarrow$ Finite	✓
(5)	$L_1 . L_2$	$F_1 \cdot F_2 \Rightarrow$ Finite	✓
(6)	L^*, L^+	F^*, $F^+ \Rightarrow$ May or may not be finite	×
(7)	Subset (L)	Subset (Finite set) $\Rightarrow$ Finite	✓

2.18.1 Table for Infinite sets

		Infinite sets
(1)	Union ($\cup$)	Infinite $\cup$ Infinite $\Rightarrow$ Infinite
(2)	Intersection ($\cap$)	Infinite $\cap$ Infinite $\Rightarrow$ Need not be infinite
(3)	Complement ($\overline{L}$)	Complement of Infinite $\Rightarrow$ May or may not be infinite
(4)	$L_1 - L_2$	Need not be infinite
(5)	$L_1 . L_2$	Infinite
(6)	L^*, L^+	Infinite
(7)	Subset (L)	Need not be infinite

2.19 Closure Properties of Regulars

$L_i \rightarrow$ Regular

1.	$\cup$	2.	$\cap$
3.	$\overline{L}$	4.	$L_1 - L_2$
5.	$L_1 \cdot L_2$	6.	L^{Rev}
7.	L^+	8.	L^+
9.	Subset (L) is not closed for regular languages	10.	Prefix (L)
11.	Suffix (L)	12.	Substring (L)
13.	Substitution (L)	14.	Homomorphism (L)
15.	$\in$-free homomorphism (L)	16.	h^{-1} (L) (Inverse homomorphism (L))
17.	L_1/L_2 (Quotient)	18.	Symmetric difference
19.	Half (L)	20.	Second half (L)
21.	One-third (L) [1/3 (L)]	22.	Middle 1/3 (L)
23.	Last 1/3 (L)	24.	New $(L_1, L_2) = \text{Suffix}(L_1 \cup L_2^+)$
25.	Finite Union	26.	Finite Intersection
27.	Finite Difference	28.	Finite Concatenation

29.	Finite Subset	30.	Finite Substitution
31.	Infinite Union	32.	Infinite $\cap$
33.	Infinite Difference	34.	Infinite Concatenation
35.	Infinite Subset	36.	Infinite Substitution

- **Out of the given 36 closure properties, how many are closed for regular languages?**

  Subset operation, and 6 infinite operations are not closed, remaining all are closed for regular languages.

  Out of 36 operations total 7 operations are not closed.

2.19.1 Operations over regular languages

Examples:

(1) $\left.\begin{matrix} L_1 = a^* \\ L_2 = a^+ \end{matrix}\right\} \Rightarrow L_1 \,.\, L_2 = a^*$

(2) $\left.\begin{matrix} L_1 = \phi \\ L_2 = \text{Any} \end{matrix}\right\} \Rightarrow L_1 + L_2 = L_2$

(3) $\left.\begin{matrix} L_1 = a^* \\ L_2 = b^* \end{matrix}\right\} \Rightarrow L_1 + L_2 = a^* + b^*$

(4) $\left.\begin{matrix} L_1 = \Sigma^* \\ L_2 = \text{Any language over same } \Sigma \end{matrix}\right\} \quad L_1 + L_2 = \Sigma^* = L_1$

(5) $\left.\begin{matrix} L_1 = a\Sigma^* \\ L_2 = b\Sigma^* \end{matrix} \quad \Sigma = \{a,b\}\right\} \quad \begin{aligned} L_1 + L_2 &= a\Sigma^* + b\Sigma^* \\ &= (a+b)\Sigma^* \\ &= \Sigma^+ \end{aligned}$

Note :

1. $\left.\begin{matrix} L_1 = \phi \\ L_2 = \text{Any} \end{matrix}\right\} \Rightarrow L_1 \cdot L_2 = \phi$

2. $\left.\begin{matrix} L_1 = \Sigma^* \\ L_2 = \text{Any} \end{matrix}\right\} \Rightarrow L_1 + L_2 = L_2$

2.19.2 Properties of regular languages

I. If both L_1 and L_2 are regular sets then $L_1 \cap L_2$ is Regular.

II. IF $L_1 \cap L_2$ is regular set then

- L_1 "need not be regular"

- L_2 "need not be regular".

III. If L is regular then $\overline{L}$ is regular.

IV. If $\overline{L}$ is regular then L is regular.

V. L is regular iff $\overline{L}$ is regular

VI. L is not regular iff $\overline{L}$ is not regular.

Example:

$a^n b^n$ is not regular and $\overline{a^n b^n}$ is not regular.

2.19.3 Arden's Lemma and Kleene Method

Arden's Lemma:

If R = Q + RP and P does not contain $\in$ then $\boxed{R = QP^*}$

$R = Q + RP$ ...(1)

$= Q + (Q + RP)\,P = Q + QP + RP^2$ ...(2)

Substitute R one more time in equation (2)

$R = Q + QP + QP^2 + RP^3$

If we do repetitive substitution infinite time, we will get $\boxed{R = QP^*}$

Kleene Method:

Kleene Method

→ Dynamic programming $O(n^3)$

→ $\boxed{R_{ij}^{k}}$ where *i* is the initial state, *j* is the final state
k represents number of states

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

Regular Grammar

→ It represents a regular set

→ It can be either left linear grammar (LLG) or right linear grammar (RLG)

LLG	RLG
Each production in LLG follows $V \rightarrow VT^*$ OR $V \rightarrow T^*$	Each production in RLG follows $V \rightarrow T^*V$ OR $V \rightarrow T^*$

2.20 Identify the Language Generated by Regular Grammar

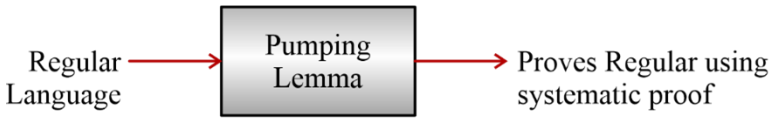
	Regular Grammar	Regular Language
1.	$S \rightarrow \in$	$L = \{\in\}$
2.	$S \rightarrow \in \mid a$	$L = \{\in, a\}$
3.	$S \rightarrow aa \mid abc \mid d$	$L = \{aa, abc, d\}$

4.	S → Aa A → b	By default S is a start symbol here. L = {ba}
5.	S → Aa	Useless production. It has no meaning. L = { } = ϕ
6.	$S \rightarrow \underset{\text{Useless}}{\boxed{Ab}} \mid b$	L = {b}
7.	↑ S → Aa | Ba A → a B → b Look from bottom to top	L = {aa, bb}
8.	↑ S → Aa A → ∈|b	L(A) = {∈, b} L = L(S) = L(A) · a = {a, ba}

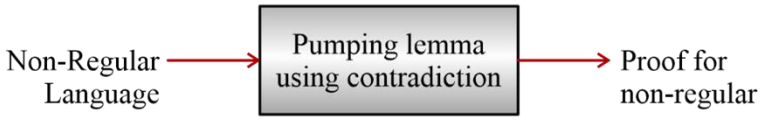
Regular Language	Regular Grammar
a^*	(I) S → Sa | ∈ OR (II) S → aS | ∈
a^+	(I) S → Sa | a OR (II) S → aS | a
$(a + b)^*$	(I) S → Sa | Sb | ∈ OR (II) S → aS | bS | ∈
$(a + b)^+$	(I) S → Sa | Sb | a | b OR (II) S → aS | bS | a | b
$(ab)^*$	(I) S → Sab | ∈ OR (II) S → abS | ∈

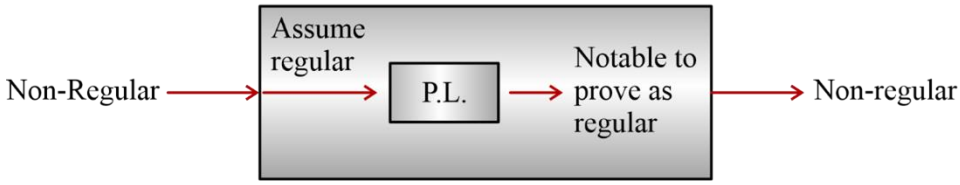
2.21 Pumping-Lemma (PL)

2.21.1 Pumping Lemma for Regular Languages



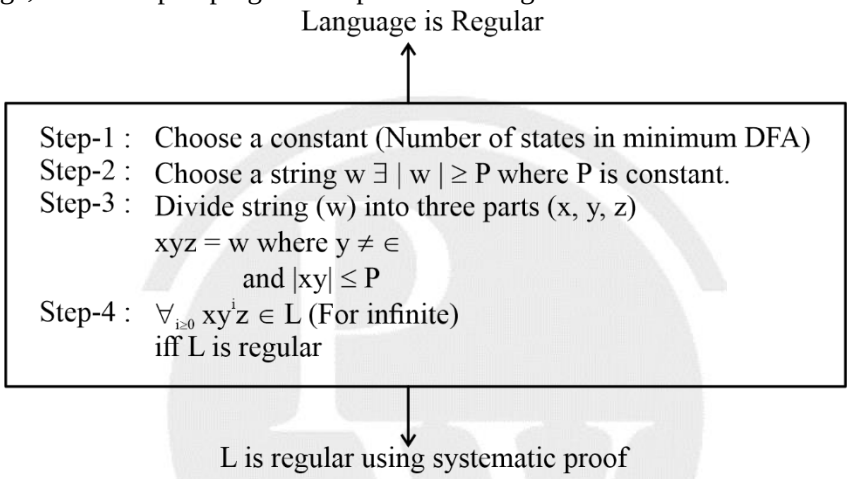
2.21.2 Pumping Lemma for Non-Regular Languages using contradiction





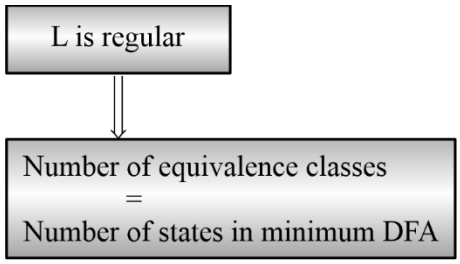
2.21.3 Proof for Regular Languages

- If L is Regular language, then how pumping lemma proves it as regular?



2.22 Equivalence Classes

(1) L is regular iff L has finite number of equivalence classes.
(2) L is not regular iff L has infinite equivalence classes.

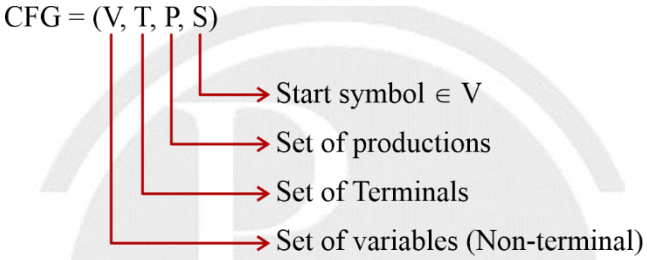


❐❐❐

3 PUSH DOWN AUTOMATA

3.1 Context Free Grammar

CFG: It represents a Context Free Language.



- Rule of each production in P:
  $V \rightarrow (V \cup T)^*$
  V = only 1 variable in LHS
- To derive a string, following derivations can be used.

1. **Linear Derivation:** Linear derivation is two types

   (a) Left Most Derivation (LMD)

   (b) Right Most Derivation (RMD)

2. **Non- linear Derivation:** Non- linear Derivation OR Parse Tree OR Derivation Tree

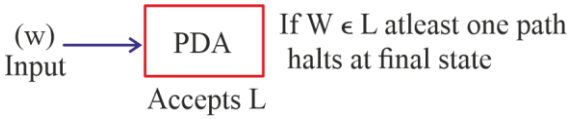
3.2 Types of Context Free Grammars

There are two types of CFG:

1. **Ambiguous CFG:** At least one string has more than one derivation.
2. **Unambiguous CFG:** Every string (w) generated by CFG has exactly one derivation.

3.3 Pushdown Automata (PDA)

PDA accepts context free language (CFL). PDA also called as NPDA.



3.4 PDA acceptance mechanisms

- PDA acceptance mechanisms are three types:
  1. PDA acceptance using final state.
  2. PDA acceptance using empty stack.
  3. PDA acceptance using both final state and Empty stack.
     All PDA acceptance mechanisms are equivalent.
- DPDA acceptance mechanism are two types:
  1. DPDA using final stack.
  2. DPDA using both final state and empty stack.

3.5 PDA configuration

PDA = $(Q, \sum, \delta, q_0, F, Z_0, \Gamma)$.

where Q = Set of states

$\sum$ = input alphabet

δ = Transition Function (PDA/NPDA $\delta : Q \times \sum \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$)

F = Set of Final state

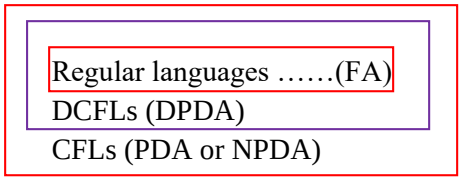
Z_0 = Bottom symbol or initially TOS

Γ = Stack Alphabet

- DPDA transition Function is [$\delta : Q \times \sum \times \Gamma \rightarrow Q \times \Gamma^*$]
- Difference between DPDA and PDA

DPDA	PDA
[1] $\delta : Q \times \sum \times \Gamma \rightarrow Q \times \Gamma^*$	$\delta : Q \times \sum \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$
[2] Every DPDA is PDA	Every PDA need not be DPDA
[3] DPDA acceptance with (a) Final state mechanism (b) Both Final and empty stack	PDA acceptance with (a) Final state mechanism (b) Empty stack mechanism (c) Both Empty stack and Final state

- Relation between Regular, DCFL and CFL.



I If L is regular language, then it is also DCFL and CFL.

II Every DCFL is CFL, but it need not be regular.

3.6 Closure Properties of CFLs

Operation	Closed / Not Closed
Union ($L_1 \cup L_2$)	✓
Intersection ($L_1 \cap L_2$)	×
Complement ($\overline{L}$)	×
Set difference (L_1-L_2)	×
Concatenation ($L_1.L_2$)	✓
Reversal (L^{Rev})	✓
Kleene Closure (L)	✓
Kleene Plus ($L^{\bullet}$)	✓
Subset (L)	×
Prefix (L)	✓
Suffix (L)	✓
Substring (L)	✓
Substitution (L)	✓
Homomorphism (L)	✓
$\in$ - free Homomorphism h(L)	✓
Inverse Homomorphism h^{-1} (L)	✓
quotient (L_1, L_2) = L_1/L_2	×
Symmetric difference (L_1, L_2)	×
Finite Union	✓
Finite Intersection	×
Finite difference	×
Finite concatenation	✓
Finite subset	✓
Finite substitution	✓
Infinite Union	×
Infinite intersection	×
Infinite difference	×
Infinite concatenation	×
Infinite Subset	×
Infinite Substitution	×
Union with Regular (L U Regular)	✓
L U Regular	✓
L – Reg	✓
Reg – L	×

Note :(i) CFL ∩ CFL = Need not be CFL

(ii) CFL ∩ Regular = CFL (Need not be DCFL)

(iii) CFL ∩ DCFL = May or may not be CFL

(iv) CFL ∩ Finite = Finite

(v) CFL ∩ infinite = Need not be CFL

3.7 Closure Properties of DCFLs

Operation	Closed / Not Closed
Union ($L_1 \cup L_2$)	×
Intersection ($L_1 \cap L_2$)	×
Complement ($\overline{L}$)	✓
Set difference (L_1-L_2)	×
Concatenation ($L_1.L_2$)	×
Reversal (L^{Rev})	×
Kleene Closure (L) = L*	×
Kleene Plus (L) = L^+	×
Subset (L)	×
Prefix (L)	✓
Suffix (L)	×
Substring (L)	×
Substitution (L)	×
Homomorphism (L)	×
∈ - free Homomorphism h(L)	×
Inverse Homomorphism h^{-1} (L)	✓
quotient (L_1, L_2)	×
Symmetric difference (L_1, L_2)	×
Finite Union	×
Finite Intersection	×
Finite difference	×
Finite concatenation	×
Finite subset	✓
Finite substitution	×
Infinite Union	×
Infinite intersection	×
Infinite difference	×
Infinite concatenation	×
Infinite Subset	×
Infinite Substitution	×
Union with Regular (L U Regular)	✓
L U Regular	✓
L – Regular	✓
Regular – L	✓

Note :

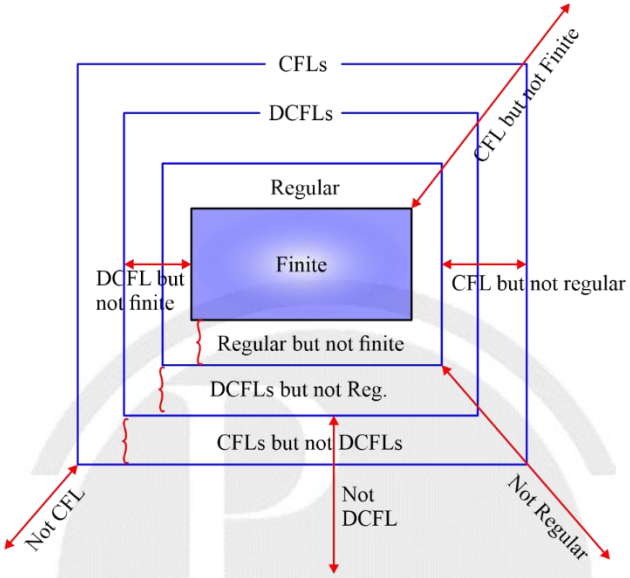
DCFL ∪ Regular : DCFL
DCFL ∩ Regular : DCFL
DCFL - Regular : DCFL
Regular - DCFL : DCFL
DCFL ∪ CFL : CFL (need not be DCFL)
DCFL ∩ CFL : Need not be CFL
DCFL - CFL : Need not be CFL
CFL - DCFL : Need not be CFL
DCFL ∪ Finite : DCFL
DCFL ∩ Finite : Finite
DCFL – Finite : DCFL
Finite - DCFL : Finite

3.8 Comparison of Regular Grammars and CFGs

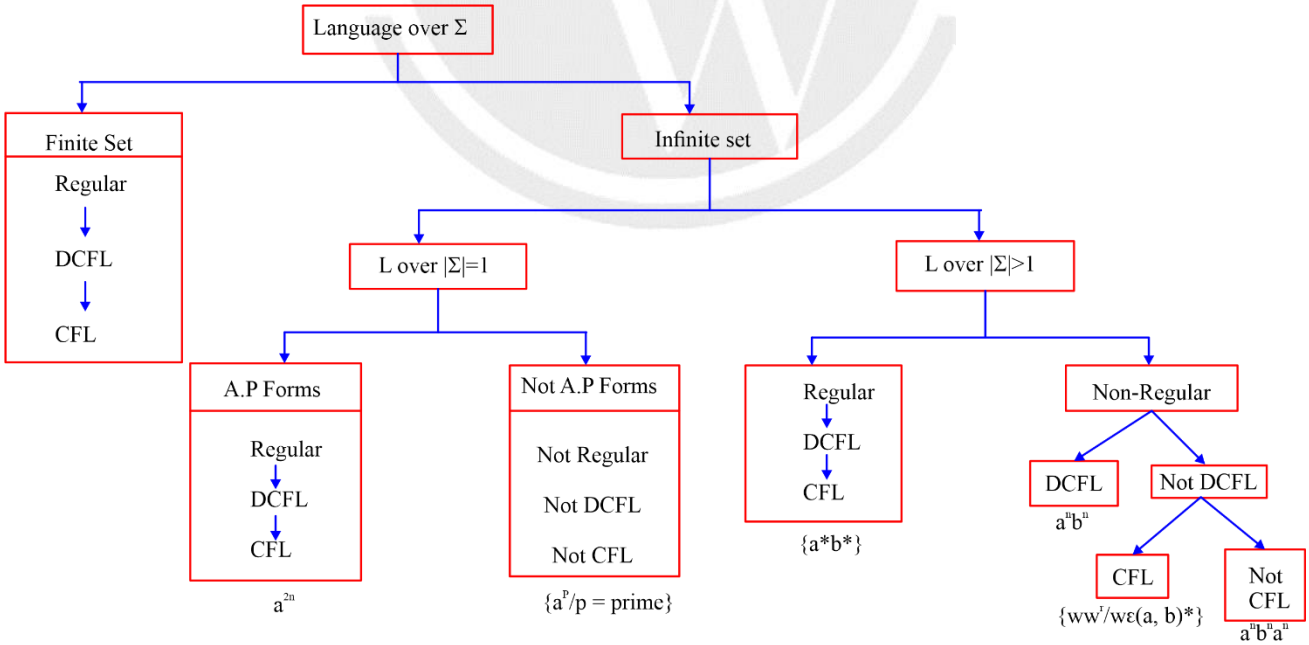
- CFL and CFG both are equivalent. CFL ≅ CFG
- LLG: Left Linear Grammar, RLG: Right Linear Grammar, RG: Regular Grammar, LG: Linear Grammar, CFG: Context Free Grammar.
- Every LLG is RG
- Every RLG is RG
- Every RG is LG
- Every RG is CFG
- Every LG is CFG
- Every RG need not be RLG
- Every RG need not be LLG
- Every LG need not be RG
- Every CFG need not be RLG
- Every CFG need not be LG

3.9 Context Free Languages and DCFLs

I. Comparison of various languages



II. Identification of Regulars, DCFLs, and CFLs



[1] $L = \{a^n b^n c^k \mid n, m, k \geq 0\}$
$= a^* b^* c^*$
= Regular

[2] $L = \{a^n b^n \mid n \geq 0\}$
L = DCFL but not regular
$S \rightarrow aSb \mid \in$

[3] $L = \{a^n b^{2n} \mid n \geq 0\}$
L = DCFL but not regular
$S \rightarrow aSbb \mid \in$

[4] $L = \{a^{2n} b^n \mid n \geq 0\}$
L = DCFL
$S \rightarrow aaSb \mid \in$

[5] $L = \{a^{2n} b^{2n} \mid n \geq 0\}$
L = DCFL
$S \rightarrow aaSbb \mid \in$

[6] $L = \{ a^n b^n c^* \}$ Assume always $n \geq 0$ in all examples
OR
$= \{ a^m b^n c^* \mid m = n\}$
DCFL but not regular

[7] $L = \{ a^n b^* c^n \}$ DCFL

[8] $L = \{ a^* b^n c^n \}$ DCFL

[9] $L = \{ a^m b^n c^* \mid m \neq n \}$
DCFL

[10] $L = \{ a^m b^n c^* \mid m < n \}$
DCFL

[11] $L = \{ a^m b^n c^* \mid m > n \}$
DCFL

[12] $L = \{ a^m b^n c^* \mid c \leq n \}$
DCFL

[13] $L = \{ a^m b^n c^* \mid m \geq n \}$
DCFL

[14] $L = \{ a^m b^n c^{m+n} \mid m, n \geq 0 \}$ is DCFL

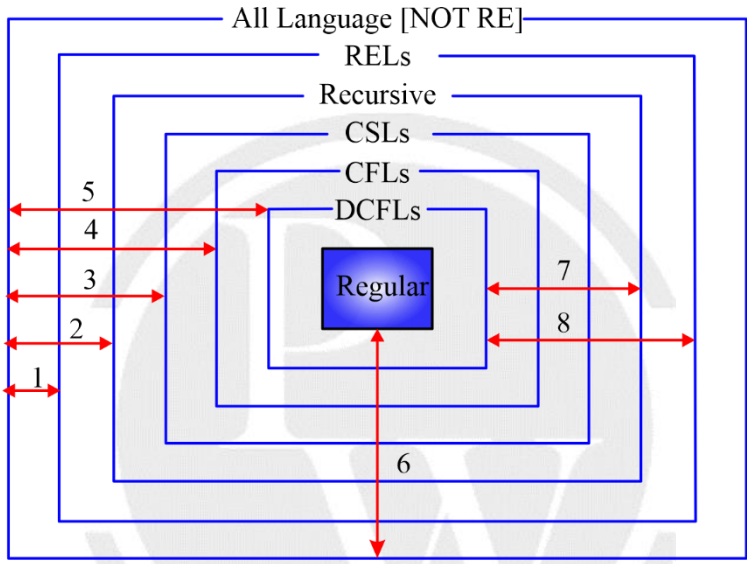
[15] L = { $a^{m+n}b^{n+k}c^{k+m}$ |m, n ≥ 0 } is CFL

[16] L = { ww^r |w belongs to {a, b}* } is CFL but not DCFL

❐❐❐

4 TURNING MACHINE

4.1 Classification of Languages



1. All languages which are not RELs
2. All not recursive languages
3. All not CSLs
4. All not CFLs
5. All not DCFLs
6. All not regulars
7. All recursive languages which are not DCFLs
8. All RELs which are not DCFLs

4.2 There are Two types of TM

(a) DTM (Deterministic TM)
(b) NTM (Non- Deterministic TM)

DTM

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \left\{ \underset{\text{Left}}{L}, \underset{\text{Right}}{R} \right\}$$

NTM

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$$

4.3 Unrestricted Grammar (UG) and Context Sensitive Grammar (CSG)

Unrestricted grammar (UG) also called as

- RE grammar
- Phase structure grammar

Context sensitive grammar (CSG) is also called as

- Non contracting grammar
- Bound restricted grammar

4.4 Grammars

Left Linear Grammar (LLG): $V \rightarrow VT^* \mid T^*$

Right Linear Grammar (RLG): $V \rightarrow T^*V \mid T^*$

Linear Grammar (LG): $V \rightarrow T^*VT^* \mid T^*$

Context Free Grammar (CFG): $LHS \rightarrow RHS, \quad |LHS| \leq |RHS|$

Unrestricted Grammar (UG): $LHS \rightarrow RHS$

4.5 Equivalence of various TMs

TM ≅ Single tape TM

TM ≅ One-way infinite tape TM

TM ≅ Two-way infinite tape TM

TM ≅ Multi tape and multi head TM

TM ≅ Universal TM

TM ≅ Two stack PDA

TM ≅ Multi stack PDA

TM ≅ FA with two stacks

TM ≅ FA + R/W tape + Bidirectional head

4.6 Restrictions on TM

(1) If TM tape is read only tape, this TM accepts regular.

TM ≅ FA (TM with read only tape)

(2) If TM head is unidirectional then L(TM) = Regular.

(3) If TM tape is read only and unidirectional head then L(TM) = Regular.

(4) If TM always halts then L(TM) is recursive language.

(5) If TM always halts and uses linear bound tape then L(TM) is CSL.

4.7 LBA, HTM and TM

HTM: It is a TM that always halts for every input.
TM: It halts for every valid string and for invalid strings either halts at "non final state" or "never halts".
LBA: It is HTM but length of the tape we use linearly bounded.

- LBA accepts CSL languages.
- HTM accepts recursive languages.
- TM accepts Recursive Enumerable (RE) languages.

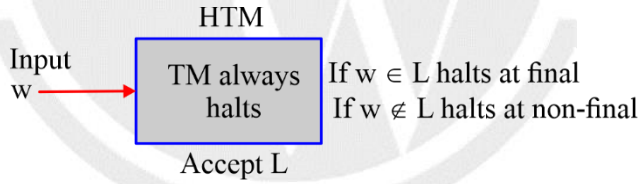
4.8 Recursively Enumerable Language

It is also called as:

- Enumerable language
- TM recognizable language
- TM Enumerable language
- Partially decidable language
- Semi-decidable language

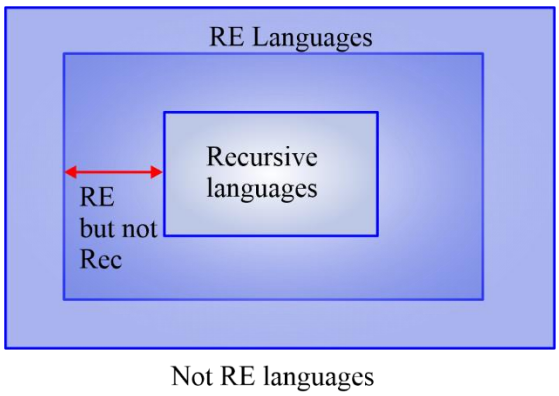
4.9 Recursive Language

- Recursive language is acceptable by HTM, and hence acceptable by TM.
- Recursive also called as decidable language.
- Recursive also called as Turing decidable language.
  If TM always halts, then TM is called as HTM.



4.10 Difference between Recursive and REL

- All Recursive languages are RE languages.



Note :

1. Union:
   - REL ∪ Finite ⇒ REL
   - REL ∪ Regular ⇒ REL
   - REL ∪ CFL ⇒ REL
   - REL ∪ Recursive ⇒ REL
   - $REL_1 \cup REL_2 \Rightarrow REL$
2. Intersection:
   - REL ∩ Finite ⇒ REL (Finite)
   - REL ∩ CFL ⇒ REL
   - REL ∩ Rec ⇒ REL
   - $REL_1 \cap REL_2 \Rightarrow REL$

4.11 Closure Properties of Recursive languages

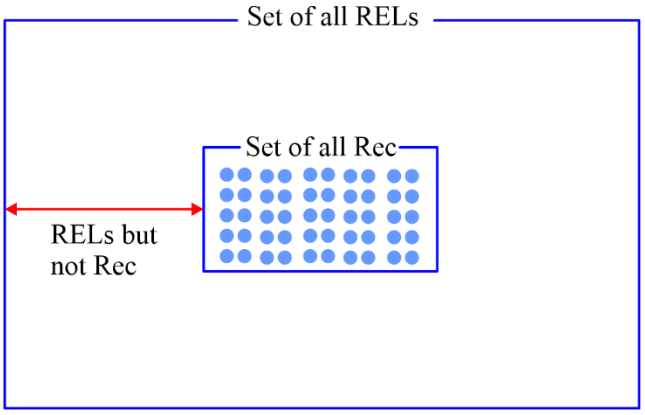
I. The following operations are not closed for recursive languages:

- Subset
- Substitution
- Homomorphism
- Finite substitution
- Infinite union
- Infinite intersection
- Infinite concatenation
- Infinite difference
- Infinite substitution

II. The following operations are closed for recursive languages:

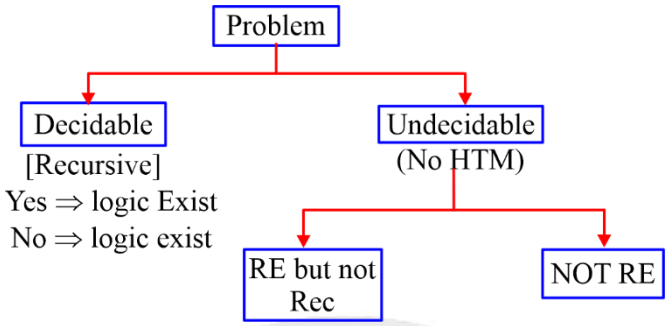
- Complement
- Difference
- Finite difference
- Infinite followed by ∪, ∩, -, •, ⊆, f
- Remember not closed operations

4.12 Complement of Recursive Vs Complement of REL



- Complement of Recursive set is Recursive.
- Complement of REL is either Recursive or non-REL.
- Complement of REL never be "REL which is not recursive".

4.13 Decidable and Undecidable



4.14 Decidable Vs Undecidable, RE Vs Not RE, Countable Vs Uncountable

[1] Decidable and Undecidable

↓ ↓

HTM exist (Rec language) | HTM not exist (not Rec language)

[2] RE and Not RE

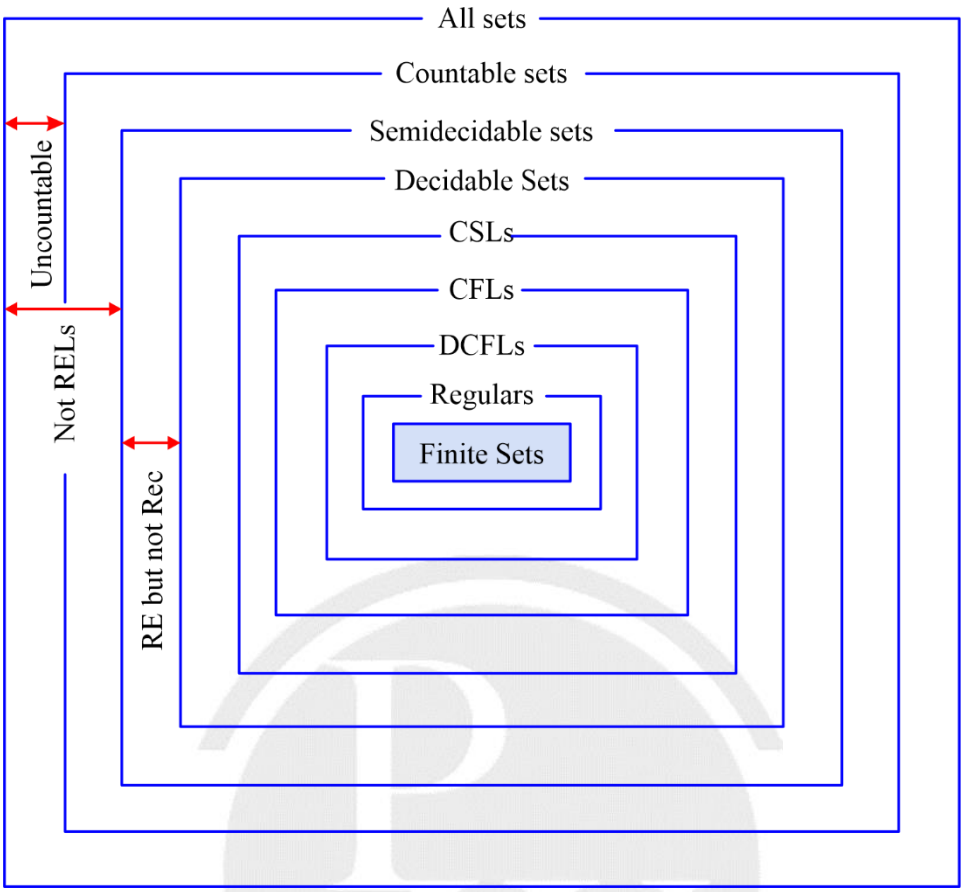
↓ ↓

TM exist | TM not exist

[3] Countable set and Uncountable set

↓ ↓

X is countable set iff $X \to f(X)$ is bijective where f(X) is known countable set. | (X is not countable set)



Note :

- If problem p is decidable then $\overline{p}$ is also decidable.
- If problem p is Undecidable then $\overline{p}$ is also UD.
- If problem p is RE but not recursive then $\overline{p}$ is not RE.
- If problem p is not RE then $\overline{p}$ is either "Not RE" or "RE but not Recursive".

4.15 Decision Properties Table

- D: Decidable
- UD: Undecidable

	FA	DPDA	PDA	LBA/HTM	TM
H (Halting)	D	D	D	D	UD
M (Membership)	D	D	D	D	UD
E_m (Emptiness)	D	D	D	UD	UD
F (Finiteness)	D	D	D	UD	UD

T (Totality)	D	D	UD	UD	UD
E_q (Equivalence)	D	D	UD	UD	UD
D (Disjoint)	D	UD	UD	UD	UD
S (Set Containment)	D	UD	UD	UD	UD

4.16 Decidable problem for DFA / NFA/ FA/ Regular

(1) Halting problem for FA / Reg/ DFA / NFA is decidable.
(2) Non-halting problem for FA / Reg/ DFA / NFA is decidable.
(3) Membership problem for FA / Reg/ DFA / NFA is decidable.
(4) Non-membership problem for FA / Reg/ DFA / NFA is decidable.
(5) Emptiness for FA / Reg/ DFA / NFA is decidable.
(6) Non-emptiness problem for FA / Reg/ DFA / NFA is decidable.
(7) Fitness problem for FA / Reg/ DFA / NFA is decidable.
(8) Non-fitness problem for FA / Reg/ DFA / NFA is decidable.
(9) Totality problem for FA / Reg/ DFA / NFA is decidable.
(10) Non-totality problem for FA / Reg/ DFA / NFA is decidable.
(11) Equivalence problem for FA / Reg/ DFA / NFA is decidable.
(12) Non-equivalence problem for FA / Reg/ DFA / NFA is decidable.
(13) Disjointness problem is decidable for FA / Reg/ DFA / NFA.
(14) N0n-disjointness problem is decidable for FA / Reg/ DFA / NFA.
(15) Set containment problem for FA / Reg/ DFA / NFA is decidable
(16) Non-set containment problem for FA / Reg/ DFA / NFA is decidable.

4.17 Decidable problems for CFLs/DCFLs

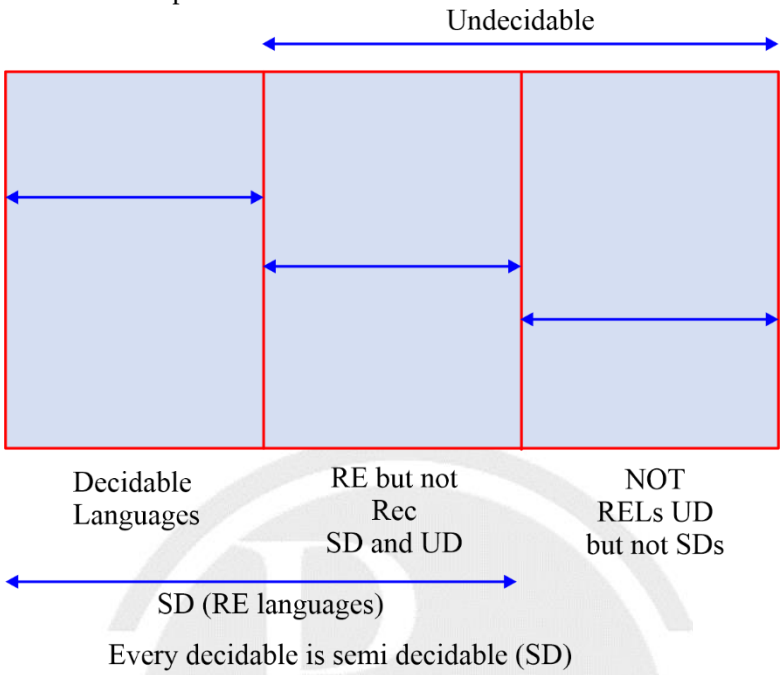
	Problems	DCFLs	CFLs
	Halting	D	D
	Non-Halting	D	D
CYK algo	Membership	D	D
CYK algo	Non-membership	D	D
Simplification algo	Emptiness	D	D
Simplification algo	Non-emptiness	D	D
Dependancy graph	Finiteness	D	D
Dependancy graph	Non-finiteness	D	D
	Totality	D	UD
	Non-totality	D	UD
	Equivalence	D	UD
	Disjointness	UD	UD
	Non-disjointness	UD	UD
	Set containment	UD	UD
	Non-set containment	UD	UD

4.18 Decidability problems for Recursive languages

	Problems	**Recursive**
1.	Halting	D
2.	Non-Halting	D
3.	Membership	D
4.	Non-membership	D
5.	Emptiness	UD [Not REL]
6.	Non-emptiness	UD [RE but not Rec] [SD but UD]
7.	Finiteness	UD [Not RE]
8.	Non-finiteness	UD [Not RE]
9.	Totality	UD [Not RE]
10.	Non-totality	UD [RE but not Rec] [SD but UD]
11.	Equivalence	UD [Not RE]
12.	Non-equivalence	UD [RE but not Rec]
13.	Disjointness	UD [Not RE]
14.	Non-disjointness	UD [RE but not Rec]
15.	Set containment	UD [Not RE]
16.	Non-set containment	UD [RE but not Rec]

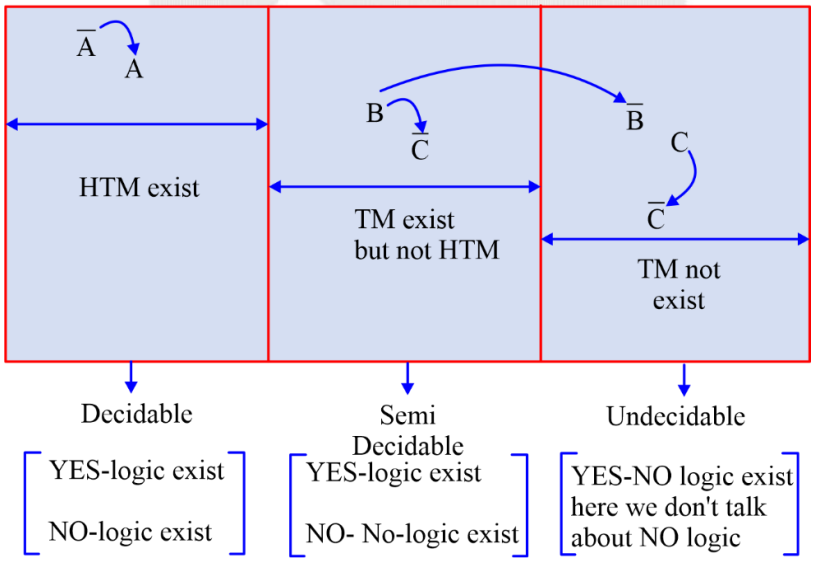
4.19 Classification of Languages based on Decidability

All RE languages can be classified into 3 important classes.



Every decidable is semi decidable (SD)

4.19.1 Decidability Vs Turing Machine



4.20 Decidable languages

(1) Finite set $\Rightarrow$ Decidable

(2) $\sum = \{a, b\} \Rightarrow$ Decidable

(3) $\sum \cong$ Set of finite number of symbols $\Rightarrow$ Decidable

(4) $\sum^*$ over alphabet $\sum = \{a, b\} \Rightarrow$ Decidable

(5) $\{M \mid M$ is DFA, M accepts $ab\} \Rightarrow$ Decidable

(6) {TM | Number of states in TM= 2} ⇒ Decidable

(7) {TM | TM reaches state q within 100 steps} ⇒ Decidable

(8) { TM | TM accepts REL} ⇒ Decidable

❐❐❐❐