# A Handbook on
# Computer Science

# 6

# Theory of Computation

## CONTENTS



# Regular Languages and Finite Automata

1

## Introduction

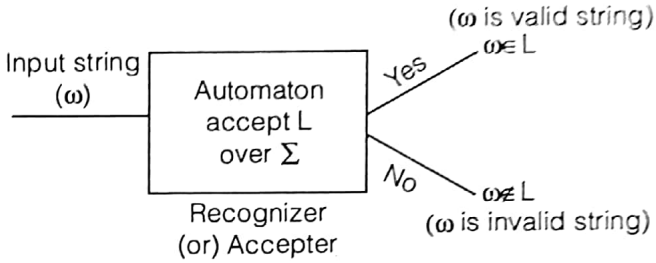
- **Alphabet ($\Sigma$)** : An Alphabet is a set of finite number of symbols.
  ***Example*** : $\Sigma = \{a, b\}$ is alphabet with two symbols *a* and *b*.
- **String** : A string is any sequence of symbols over the given alphabet $\Sigma$.
  ***Example*** : *abb* is a string over $\Sigma = \{a, b\}$
- Empty string ($\varepsilon$ or $\lambda$): Empty string is a string with no symbol over any alphabet. Empty string is also called as "Null String".
  $|\varepsilon| = 0$ (i.e, empty string length is 0)
- **Length of a string** : The number of symbols in the sequence of given string is called "length of a string".
  ***Example*** : $|abb| = 3$, for the string *abb* over $\Sigma = \{a, b\}$
- **Substring of a string** : Any sequence of zero or more consecutive symbols of the given string over an alphabet is called a "Substring of a string".
  ***Example*** : For string *abb* over $\Sigma = \{a, b\}$, the possible substrings are: $\varepsilon$, *a*, *b*, *ab*, *bb* and *abb*.
- **Prefix of a string** : Any substring with the sequence of beginning symbols of a given string is called a "Prefix".
  ***Example*** : For string "*abb*", the possible prefixes of abb are: $\varepsilon$, *a*, *ab*, *abb*.
- **Suffix of a string**: Any substring with the sequence of trailing (ending) symbols of a given string is called a "Suffix".
  ***Example*** : For string *abb*, the possible suffixed are: $\varepsilon$, *b*, *bb*, *abb*.
- **Power of an Alphabet:**
  Consider $\Sigma = \{a, b\}$. The following are powers of an alphabet over $\Sigma$.
  $\Sigma^0 = \{\varepsilon\}$ : zero length string
  $\Sigma^1 = \{a, b\}$ : set of 1-length strings
  $\Sigma^2 = \{aa, ab, ba, bb\}$ : set of 2-length strings
  $\Sigma^*$ = the set of all strings over $\Sigma = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$
  $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots = \Sigma^* - \Sigma^0 = \Sigma^* - \{\epsilon\}$
- **Language** : A set of strings over the given alphabet $\Sigma$ is called a "language" (or collection of strings).
  ***Example*** : Let $\Sigma = \{a, b\}$. Then Language L = {*ab*, *aab*}

- **Grammar :** Grammar Contain set of rules to generate the strings of a language. Grammar is a set of 4 tuples. Grammar $G = \{V, T, P, S\}$ where $V$ is set of non-terminals, $T$ is set of terminals, $P$ is set of rules and $S$ is start symbol ($S \in V$).

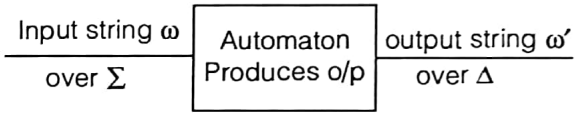
  Each rule of the grammar appears as : $X \rightarrow Y$, where $X$ and $Y$ are any sequence of terminals and non-terminals.

- **Automaton:**

  **(*i*) Automaton Acts as Recognizer or Accepter:** An automaton is a machine that accept or recognizes the strings of a language (L) over an input alphabet $\Sigma$.




  **(*ii*) Automaton Acts as producer or enumerator or transducer:** An automaton can also produce the output over any alphabet $D$ for the given input alphabet $\Sigma$.




- **Formal Languages:** Formal language is a set of all strings where each string is restricted over a particular forms of a string with the given input.

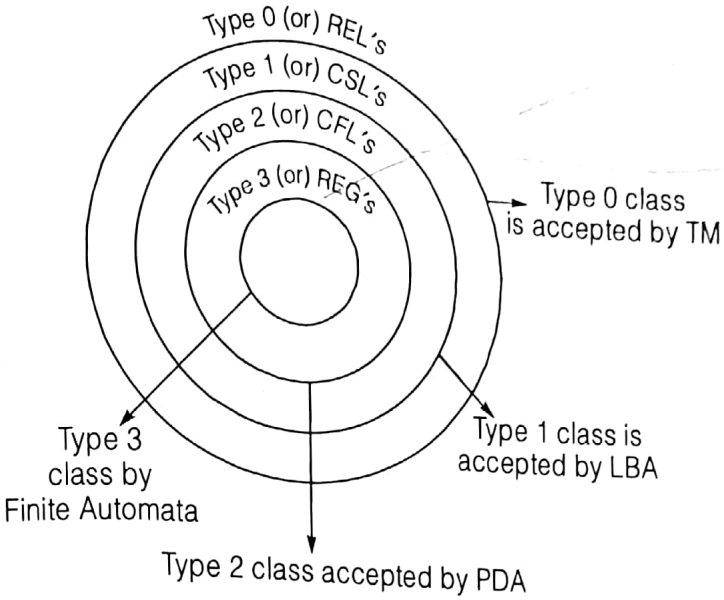
- Equivalence of language, grammar and automata

| | Formal Languages | Grammars | Automata |
|---|---|---|---|
| 1. | Regular Languages | Regular grammars | Finite Automata |
| 2. | Context Free Languages | Context Free Grammars | Push down Automata |
| 3. | Context Sensitive Languages | Context Sensitive Grammars | Linear Bound Automata |
| 4. | Recursive Enumerable Languages | Recursive enumerable grammar (unrestricted) | Turing Machines |

Every formal language can be generated by equivalent grammar and be accepted by the equivalent automaton as shown in above table.

- **Chomsky Hierarchy :**

All formal languages are divided into 4 classes by chomsky and those class hierarchy known as "Chomsky-Hierarchy".




Type 3 ⊂ Type 2 ⊂ Type 1 ⊂ Type 0.

**Types of Automaton :**

(*i*) **Finite Automaton (FA):** An automaton that accepts a regular language is called a "FA".

**Types of Finite Automaton:**

(a) Deterministic FA(DFA)

(b) Non-deterministic FA(NFA or NDFA)

**L(DFA) ≅ L(NFA)**

i.e., both types of finite automata have same expressive power.

(*ii*) **Push Down Automaton(PDA):** An automaton that accepts a context free language is called a "PDA".

**Types of PDA:**

(a) Deterministic PDA(DPDA).

(b) Non-deterministic PDA(NPDA or PDA).

L(DPDA) < L(NPDA) i.e., DCFL's ⊂ CFL's

Class of languages accepted by DPDA's are proper subset of class of languages accepted by NPDA's.

The class of NPDA's have more expressive power than the class of DPDA's

(*iii*) **Linear Bound Automaton (LBA):** An automaton that accepts a context sensitive language is called a "LBA".

(*iv*) **Turing Machine (TM):** An automaton that accepted a recursive enumerable language is called a "TM". TM can accept a REL and TM can enumerate a REL.

Types of TM:

(a) Deterministic TM (DTM)

(b) Non-deterministic TM (NTM)

L(NTM) ≡ L(DTM)

- **Types of Grammars:**

  **(*i*) Regular Grammar(RG) or type-3:** A grammar is regular grammar iff it is either left linear grammar or right linear grammar. Regular grammar is a grammar where all rules are restricted as following.

  Either $V \to VT^* \mid T^*$ [Left linear grammar].

  or

  $V \to T^*V \mid T^*$ [Right linear grammar]

  where V is any variable and T any terminal.

  **(*ii*) Context Free Grammar (CFG) or Type-2:** Every rule of CFG is restricted as $V \to (V \cup T)^*$

  **(*iii*) Context Sensitive Grammar (CSG) or Type-1:**

  Rule : $X \to Y$

  Where $|X| \leq |Y|$, $X \in (V \cup T)^*$ and $Y \in (V \cup T)^*$ and X must contain atleast one variable.

  If NULL productions are present in the grammar, it may or may not be CSG.

  **(*iv*) Recursive Enumerable Grammar (REG) or Type-0:**

  Rule : $X \to Y$

  Where $X \in (V \cup T)^*$, $Y \in (V \cup T)^*$ and X must contain atleast one variable.

- **Equivalences of Language, Automata & Grammar :**

  **(*i*) Regular Language (Type-3 language or finite state language):**

  ≅

  Finite Automaton [ DFA ≅ NFA ≅ ε-NFA]

  ≅

  Regular Grammar (Type-3 grammar)

  ≅

  Left linear Grammar

  ≅

  Right linear Grammar

  ≅

  Regular Expression

(*ii*) Context Free language (Type-2 language):

≅

Push Down Automata [NPDA or PDA]

≅

Context Free Grammar

(*iii*) CSG ≅ LBA ≅ CSL (Type-1 language)

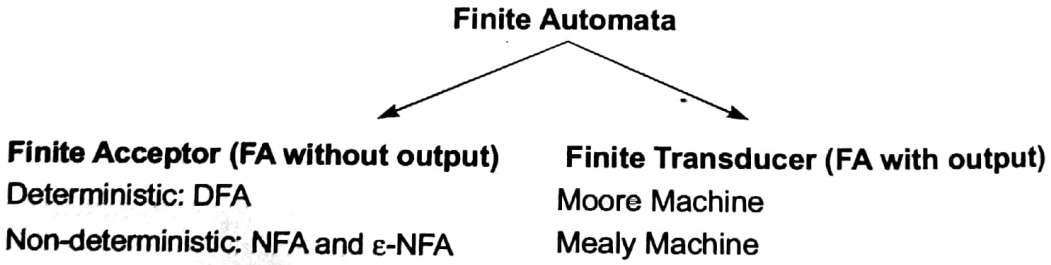
(*iv*) REG ≅ TM ≅ REL (Type-0 language)

- **Expressive Power of Automata:** Expressive power of a machine is the class or set of languages accepted by the particular type of machines. FA < DPDA < PDA < LBA < TM. FA is less powerful than any other machine and TM is more powerful than any other machine.
  - (*i*) Type 3 class ⊂ Type 2 class ⊂ Type 1 class ⊂ Type 0 class
  - (*ii*) **FA** ≅ TM with read only tape ≅ TM with unidirectional tape ≅ TM with finite tape ≅ PDA with finite stack
  - (*iii*) PDA ≅ FA with stack
  - (*iv*) TM ≅ PDA with additional stack ≅ FA with two stacks
- **Applications of Automata:**
  - (*i*) **Finite automata can be used in the following cases:**
    - (a) To design a lexical analysis of a compiler
    - (b) To recognize the pattern using regular expressions
    - (c) To design the combination and sequential circuits using Moore/ mealy machines.
  - (*ii*) **PDA can be used in the following cases:**
    - (a) To design the parsing phase of a compiler (syntax analysis)
    - (b) To implement stack applications
    - (c) To evaluate arithmetic expressions
    - (d) To solve the Tower of Hanoi problem
  - (*iii*) **Linear Bounded Automata can be used in the following cases:**
    - (a) To construct syntactic parse trees for semantic analysis of the compiler.
    - (b) To implement genetic programming.
  - (*iv*) **Turing Machine can be used in the following cases:**
    - (a) To solve any recursively enumerable problem
    - (b) To understand complexity theory
    - (c) To implement neural nets
    - (d) To implement artificial Intelligence
    - (e) To implement Robotic applications

**Remember:**

- NFA has same power as of DFA
- DPDA has less power than NPDA
- DTM has same power as of NTM
- Mealy and Moore machines have same power
- **Mealy machine:** Output depends on current state and current input
- **Moore machine:** Output depends on current state only.
- Finite automaton does not have infinite memory required for comparison.
- Every finite language is regular but converse need not be true.
- Language may be infinite but variables, terminals and production rules are finite for every grammar.
- Regular grammar is either left linear or right linear or both.
- Every NFA can be converted to DFA and then to minimal DFA
- If NFA has $n$-states then the equivalent DFA has atmost $2^n$ states.
- Let $w$ be a string of length $n$. Then number of possible substring including $\varepsilon$ is atmost $\frac{n(n+1)}{2}+1$.
- Number of prefixes for the given $n$-length string = $(n+1)$
- Number of suffixes for the given $n$-length string = $(n+1)$

## Finite Automaton (FA)




- **Finite automata is categorized into two types:**
  - (*i*) **Finite automata with output:** This is of two types Mealy machines and Moore machines.
    - (a) In Mealy machine the output is associated with transition (state and i/p symbol)
    - (b) In the Moore machine output is associated with the state.

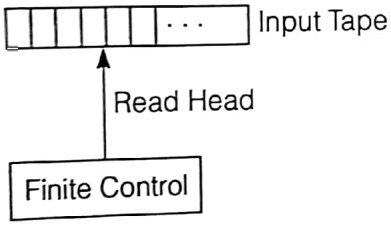
(ii) **Finite automata without output:** This is of two types DFA and NFA

(a) **DFA:** It is deterministic finite automata. For every input symbol in DFA there is an exactly one transition from each state of finite automata. It accepts all strings that are present in the language by halting in a final stage and rejects all strings that are not present in the language by halting in a non-final state.

(b) **NFA:** It is non deterministic finite automata. For every input symbol in NFA there is atleast one transition or no transitions from each state of finite automata. It accepts all strings that are present in the language by halting in a final state and rejects all strings not present in the language either by halting in a final state or by halting in a dead configuration.

(c) **ε-NFA:** It is a non deterministic finite automata which include ε transitions between states.

**Model of Finite Automata:**




(*i*) Input tape contains a string over the given input alphabet.

(*ii*) Read head is a pointer which reads one input symbol at a time and moves to the next symbol.

(*iii*) Finite control contains set of states and a transition function to take an action based on current state and input symbol scanned.

## Deterministic Finite Automata (DFA or FA)

For every input symbol of alphabet there is an exactly one transition from every state of finite automata is called as DFA.

DFA covers transitions for all valid and invalid strings in the given regular language.

For every valid string, DFA reaches to one of its Final state and for every invalid string it reaches to non-final state.

## Specifications of DFA

DFA = $(Q, \Sigma, \delta, q_0, F)$ is 5-tuple notation.

Where Q is set of finite states,

$\Sigma$ is input alphabet contains finite number of input symbols

$\delta$ is transition function defined over transitions of FA for state and input

$\delta : Q \times \Sigma \rightarrow Q$

$q_0$ is initial state or start state of DFA, $q_0 \in Q$

F is set of final states of DFA

## Compound FA (F1 × F2)

Let F1 and F2 are two DFA's. Then operations defined over FA are (F1 ∪ F2), (F1 ∩ F2), (F1 – F2), and (F2 – F1). Construction of compound FA for the given F1 and F2 as follows:

1. The number of states in compound FA (F1 × F2) is equal to m × n, where m is the number of states of F1 and n is the number of states of F2.
2. Initial state of compound FA is combination of initial state of F1 and initial state of F2.
3. Let Q1 = $\{q_0, q_1, q_2\}$ is set of states of F1, and Q2 = $\{q_a, q_b\}$ is set of states of F2. Let $q_0$ be the start state of F1 and $q_a$ be the start of F2. Then Q1 × Q2 = $\{(q_0, q_a), (q_0, q_b), (q_1, q_a), (q_1, q_b), (q_2, q_a), (q_2, q_b)\}$, where $(q_0, q_a)$ is an initial state of compound FA.
4. Let $q_2$ is final of F1 and $q_b$ is final of $F_2$. Final states depends on type of compound finite automata (F1 × F2)
   (a) F1 ∪ F2 final states: Make those states of (Q1 × Q2) as final, if final state of F1 ***or*** final state of F2 contains in any of the states. Such a FA accepts L(F1) ∪ L(F2).
   (b) F1 ∩ F2 final states: Make those states of (Q1 × Q2) as final, if only final state of F1 ***and*** final state of F2 contains in any of the states. Such a FA accepts L(F1) ∩ L(F2).
   (c) F1 – F2 final states: Make those states of (Q1 × Q2) as final, if only final state of F1 ***but not*** final state of F2 contains in any of the states. Such a FA accepts L(F1) – L(F2).
   (d) F2 – F1 final states: Make those states of (Q1× Q2) as final, if only final state of F2 ***but not*** final state of F1 contains in any of the states. Such a FA accepts L(F2) – L(F1).

## Non-deterministic Finite Automata (NFA or NDFA)

For every valid string there exists a path from initial state that reaches to final state. NFA covers transitions for all valid strings only in the given regular language.

### Specification of NFA without Null Moves

NFA = $(Q, \Sigma, \delta, q_0, F)$ is a 5-tuple notation.

Where Q is set of finite states,

$\Sigma$ is input alphabet contains finite number of input symbols

$\delta$ is transition function defined over transitions of NFA for state and input.

$\delta: Q \times \Sigma \rightarrow 2^Q$

## Epsilon NFA (ε-NFA) or NFA

For every valid string there exists a path from initial state that reaches to final state. ε-NFA is same as NFA but it can include epsilon transitions.

### Specification of ε-NFA

ε-NFA = $(Q, \Sigma, \delta, q_0, F)$ is a 5-tuple notation.

Where $\delta: Q \times \Sigma \cup \{\varepsilon\} \rightarrow 2^Q$.

## Regular Expressions

It is a declarative way of representation of a regular language. Regular expression is a compact formula that generates exactly those strings which are in a language.

- **Operators of Regular Expressions:** The operators in regular expressions are: * is kleene closure, · is concatenation operator and + is union operator.
- **Important Notations of Regular Expressions:** Some of the important notations compared to regular sets are:

| Language | | Regular expression |
|---|---|---|
| { } | → | ϕ |
| {ε} | → | ε |
| {a} | → | a |
| {a, b} | → | a+b |
| {a, b, c} | → | a+b+c |
| {a, b, c,... z} | → | a+b+c... + z |

- **Properties of Regular Expression Operators:**
  (a) Union operator satisfies commutative property and associative property.
  (b) The concatenation operator satisfies associative property but not commutative property
  (c) Both left and right distributive property of concatenation over union are satisfied. $r(s + t) = rs + rt$ and $(s + t)r = sr + tr$
  (d) Both left and right distributive properties of union over concatenation are not satisfied. $st + r \neq (s + r)(t + r)$ and $r + st \neq (r + s)(r + t)$
  (e) Union operator satisfies idempotent property but the concatenation operator does not satisfy. i.e. $r + r = r$ but $r \cdot r \neq r$.
  (f) **Identity property:**
  $R + \phi = \phi + R = R$: $\phi$ acts as identity element with respect to union operation
  $\varepsilon.R = R.\varepsilon = R$: Epsilon ($\varepsilon$) acts as identity element with respect to concatenation.

- **Equivalences of Languages (or regular expressions):**
  (a) $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
  (b) $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$
  (c) $L(r^*) = (L(r))^*$
  (d) $r_1 \cdot r_2 \neq r_2 \cdot r_1$
  (e) $r_1(r_2 r_3) = (r_1 r_2)r_3$
  (f) $r_1(r_2 + r_3) = r_1 r_2 + r_1 r_3$
  (g) $r_1 + r_2 r_3 \neq (r_1 + r_2)(r_1 + r_3)$
  (h) $\phi + r = r$
  (i) $\phi.r = \phi = r.\phi$
  (j) $\lambda r = r\lambda = r$
  (k) $\lambda^* = \lambda$
  (l) $\phi^* = \lambda$
  (m) $r + r = r$
  (n) $r.r \neq r$
  (o) $r^*.r^* = r^*$
  (p) $r^*.r^+ = r^+$
  (q) $(r^*)^* = r^*$
  (r) $(r)^* = r^*$

(s) $\lambda + rr^* = r^* = \lambda + r^+$

(t) $p(qp)^* = (pq)^*p$

(u) $(p + q)^* = (p^*q^*)^* = (p^*+q^*)^*$

(v) $(p + q)^*p^*q^* = (p + q)^*$

## Closure Properties of Regular Languages

- The set of regular languages is closed under the union operation.
- Finite union is closed for regular languages but infinite union is not closed.
- The set of regular languages is closed under the intersection operation.
- Regular languages are closed under finite intersection but not under infinite intersection.
- The set of regular languages is closed under the difference operation.
- The set of regular languages is closed under the complement operation.
- The set of regular languages is closed under the concatenation operation.
- The set of regular languages is closed under the kleene closure operation.
- The set of regular languages is closed under the positive closure operation.
- The set of regular languages is closed under the prefix operation as well as suffix operation.

  Init(L) = Prefix(L) = {u | uv is in regular language L}

  Suffix(L) = {v | uv is in regular language L}
- The set of regular languages is closed under the reversal operation.
- Half of a Regular Language is also regular: Let L is a regular language. Then Half(L) = {u | uv ∈ L and |u| = |v|}.
- One third of a Regular Language is also regular: Let L is a regular language. Then onethird(L) = {u | uvw ∈ L and |u| = |v| = |w|}.
- Quotient of two Regular Languages is also regular: L1/L2 = {x | xy ∈ L1 for some y ∈ L2}.
- Subsequence of a Regular Language is also regular language: Subsequence(L) = {w | w is obtained by removing symbols from anywhere of a string in L}.
- Sub-word of a Regular Language is also regular: Sub-word(L) = {v | for some u and for some w, uvw is in L}.
- Homomorphism of two Regular Languages is also regular.
- Inverse Hommorphism of two Regular languages is also regular.
- Substitution of two Regular Languages is also regular.

- Shuffle of two Regular Languages: Shuffle($L_1$, $L_2$)
  $= \{x_1y_1x_2y_2...x_ky_k \mid x_1x_2...x_k$ is in $L_1$ and $y_1y_2...y_k$ is in $L_2\}$ is also regular.
- Symmetric difference of two Regular Languages is also regular.
- If L is regular then $\sqrt{L}$ is also a Regular Language: $\sqrt{L} = \{w \mid ww \text{ in } L\}$.
- If L regular then Max(L) is also a Regular Language.
- If L is regular than Min(L) is also regular. Min(L) = {w | w ∈ L and there is no proper prefix of w is in L}.
- Regular language are not closed under subset, superset, infinite union as well as infinite intersection. i.e. a subset or superset of a regular language need not be regular. Infinite union and infinite intersections of regular languages need not be regular.

## Decision Properties of Regular Language

- **Membership:** It is decidable.
- **Emptiness (Is L empty?):** Checking whether the given regular language is empty or not. It is decidable.
- **Finiteness (Is L finite?):** Checking whether the given language is finite or not. It is decidable.
- **Equivalence of two Regular Languages (Is L1 ≡ L2?):** Checking whether the given two regular languages are equal or not. It is decidable.
- **Subset Operation over Two Regular Languages (Is L1 ⊆ L2?):** Checking whether the regular language L1 is subset of a regular language L2 or not. $(L1 \subseteq L2) \cong (L1 \cap \overline{L2} = \phi)$. It is decidable.

## Mealy and Moore Machines

Mealy and Moore machines are acts as language generators but not language recognizers. There are no final states in Moore and Mealy machine.

- **Mealy Machine:** In Finite automata, if output symbol is associated with transition (state and input) then such automata is called as Mealy machine.
- **Moore Machine:** In Finite automata if output symbol is associated with each state then such automata is called as Moore machine.

**Specification of Mealy and Moore Machines:**

$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is a 6-tuple notation.

Where,

$Q$ is set of finite states,

$\Sigma$ is input alphabet contains finite number of input symbols.

$\Delta$ is output alphabet which contains output symbols.

$\delta$ is transition function. $\delta : Q \times \Sigma \to Q$

$\lambda$ is output function: $\lambda : Q \to \Delta^*$ (Moore machine)

$\lambda : Q \times \Sigma \to \Delta^*$ (Mealy machine)

$q_0$ is initial state or start of Moore machine, $q_0 \in Q$

## Pumping Lemma

Pumping lemma can be used to prove that a given language is non-regular (using proof by contradiction). Then there exist a constant 'n' such that for every string 'w' in L of length 'n' or more the following holds. Let w has three parts x, y and z.

1. $w = xyz$, where $w \in L$ and $|w| \geq n$.
2. For all $i \geq 0$, $xy^i z \in L$
3. $|y| > 0$.
4. Optional condition $|xy| \leq n$.

***Examples:*** (Non regular languages)

$L = \{a^m b^m \mid i \geq 0\}$

$L = \{a^p \mid p \text{ is a prime}\}$

$L = \{a^m b^n \mid m > n\}$

$L = \{a^m b^n \mid m < n\}$

$L = \{a^m b^n \mid m! = n\}$

$L = \{w \mid w \varepsilon (a + b)^*\}$ and w has #a's equal to #b's}

$L = \{ww \mid w \varepsilon (a + b)^*\}$

$L = \left\{a^{n^2}\right\}$

$L = \left\{a^{2^n}\right\}$

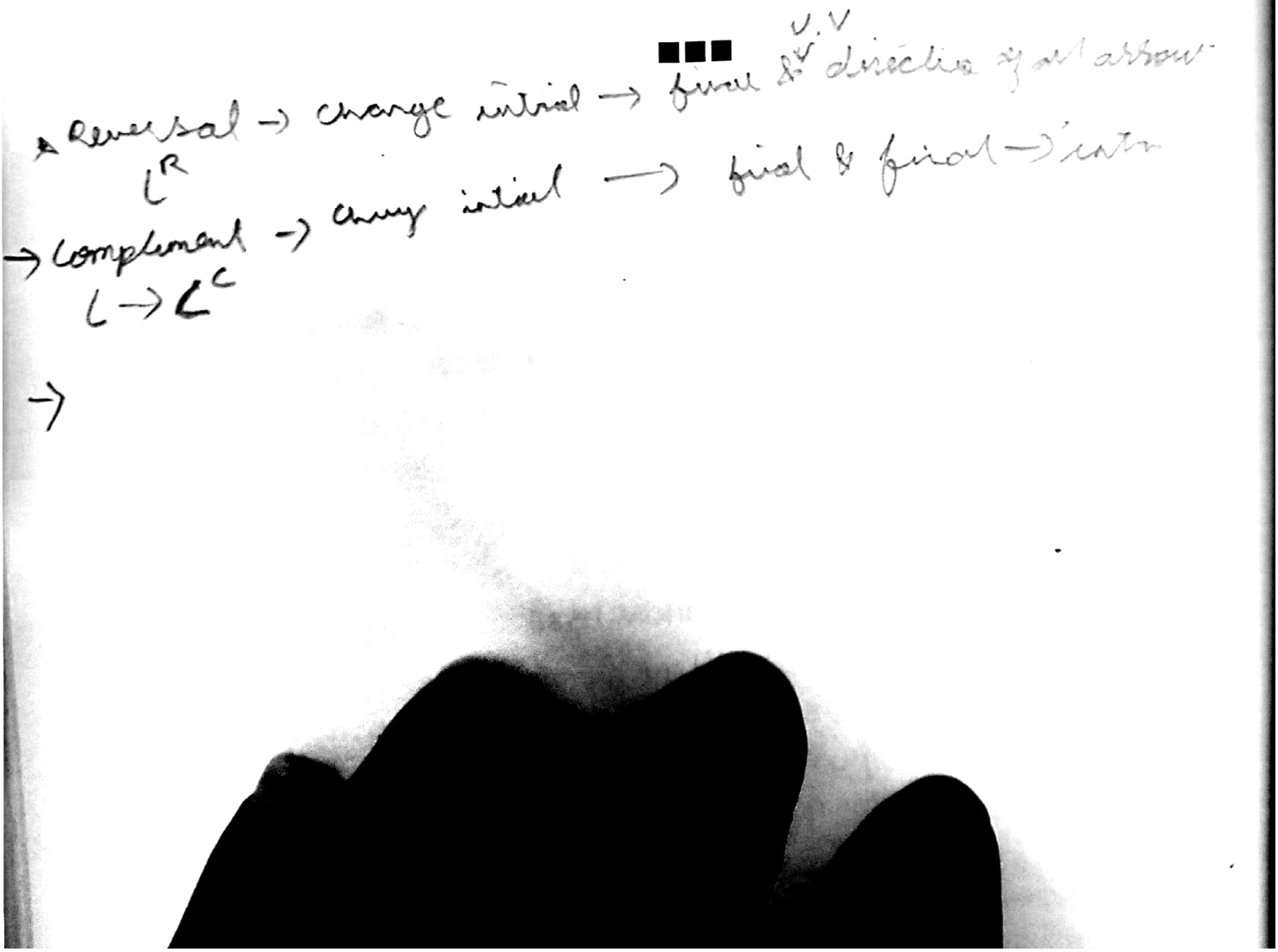
$L = \{a^n b^n c^n\}$

## Minimization of FA

Finite automata need to be minimized to reduce number of states of DFA.

- **Minimization methods:** Partition method (state equivalence), table filing method and Myhill-Nerode theorem.
- **Myhill – Nerode Theorem:** Let L be a regular language, then it contains finite number of equivalence classes. Union of all Myhill-Nerode equivalence classes gives universal language. All the strings of universal language partitioned into a finite number of Myhill-Nerode equivalence classes, if L is regular.

  The number of Myhill-Nerode equivalence classes in a regular language ≡ Number of states in the unique minimal DFA corresponding to that language.



# Push Down Automata and CFLs

2

## Context Free Grammar (CFG)

- Derivation of a String:
  - (*i*) **Left Most Derivation (LMD):** In each sentential form, the left most non-terminal is substituted with its productions to derive a string.
  - (*ii*) **Right Most Derivation (RMD):** In each sentential form, the right most non-terminal is substituted with its productions to derive a string.
  - (*iii*) **Parse tree:** Every intermediate node of a parse tree is a non-terminal which is represented by a Circle and all leaf nodes are terminal symbols.
- **Umambiguous grammar:** The grammar is called as unambiguous grammar if every string generated from the grammar has exactly one parse tree.

  #parse tree's = # LMD's = # RMD's = 1
- **Ambiguous grammar:** The grammar is called as ambiguous grammar if there exists a string which is derived from the grammar with more than one parse tree.

  (#parse tree's = # LMD's = # RMD's) > 1
- **Inherently Ambiguous Context Free language:** For a given language if there is no equivalent unambiguous CFG then such language is called as inherently ambiguous context free language.

  $L = \{a^k b^l c^k\} \cup \{a^m b^m c^n\}$ is an example of an inherently ambiguous CFL.
- **Reduced CFG (non-redundant CFG):** Reduced CFG is a CFG without any useless symbols.
- **Simplified CFG:** Simplified CFG is a CFG without any null-productions, unit-productions and useless symbols.

  The following order applied over a given CFG to convert into simplified CFG.
  - (*i*) Null-productions elimination.
  - (*ii*) Unit-productions elimination.
  - (*iii*) Useless symbols elimination.

### Normal forms for CFG

**Types of CFG Normal forms:**

1. **Chomsky Normal Form:** Every CFG production of the CNF grammar contains either two non terminals or a single terminal in the RHS of the production.

$A \rightarrow BC$

$A \rightarrow a$

CNF is also called binary standard form (the parse tree in CNF is always a binary tree).

(*i*) The number of steps required in derivation of an $x$-length string from the given CNF – context free grammar is: $(2x - 1)$

(*ii*) Let G be the given CFG without '$\in$' and unit productions and let '$K$' be the maximum number of symbols on the right hand side of any production, then the equivalent CNF contain a **maximum of:** $(K - 1)|P| + |T|$ **productions**. Where, $|P|$ = the number of productions in '$G$', $|T|$ = the number of terminals, $K$ = maximum number of symbols on right hand side.

(*iii*) For the generation of '$l$' length yield, the minimum height of the derivation of tree for the given CNF context free grammar is $\lceil \log_2 l \rceil + 1$ and maximum height will be $l$.

2. **Greibach normal form:** Every CFG production of GNF grammar contains a single terminal (T) followed by any sequence of non-terminals (V*).

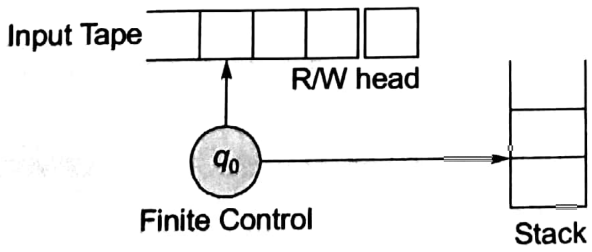
$V \rightarrow TV^*$

(*i*) In GNF context free grammar, the restrictions only on the positions, but not on length of right side of a production. $A \rightarrow a\alpha$, where $\alpha \in V$.

(*ii*) The number of steps required in derivation of an $x$-length string from the given GNF-context free grammar is: $x$

## Push Down Automata

### Configuration




- It contains finite control, input tape and a stack
- Finite control contains finite number of states.
- Read head or read pointer is used to read the input symbol from the input tape

- The symbols can be pushed or popped from the stack, which is used to keep track of previous symbols.
- For every string finite control starts from the initial state. If the string is valid finite control reaches to the final state at end of input.

## Acceptance Mechanisms

- Acceptance by final state (universal mechanism).
- Acceptance by empty stack (local mechanism).
- Acceptance by empty stack and final state.

**CFL ≅ PDA by empty stack ≅ PDA by final state ≅ PDA by empty stack and final state ≅ CFG.**

### Operations

The operations on push down automata are as follows:

- **PUSH:** Some finite sequence of symbols is pushed into the stack.
- **POP:** One symbol at the top of the stack is popped.
- **BIPASS:** i.e. do nothing i.e. stack contents unchanged while state may be changed
- **REPLACE:** Replace one symbol at the top of the stack.

### Specifications

PDA = $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a 7 tuple notation.

Where,

$Q$ is set of finite states.

$\Sigma$ is the input alphabet which contains finite number of input symbols.

$\Gamma$ is the stack alphabet which contains a finite number of stack symbols.

$\delta$ is a transition function.

(*i*) (DPDA) $\delta : Q \times (\Sigma \cup \{\in\}) \times \Gamma \to Q \times \Gamma^*$

(*ii*) (NPDA) $\delta : Q \times (\Sigma \cup \{\in\}) \times \Gamma \to 2^{Q \times \Gamma^*}$ (only finite sub-sets)

$q_0$ is a single start state $q_0 \in Q$.

$Z_0$ is the start stack symbol $Z_0 \in \Gamma$.

F is the set of final states $F \subseteq Q$. If acceptance is by empty stack method then F is not specified.

## Context Free languages (Non-Regular PDA Languages)

1. $L = \{ a^n b^n \mid n \geq 1\}$
2. $L = \{a^n b^n c^m \mid m,n \geq 1\}$
3. $L = \{a^m\ b^{m+n}\ c^n \mid m, n \geq 1\}$
4. $L = \{a^n\ b^{3n} \mid n \geq 1\}$
5. $L = \{w \in (a + b)^* \mid$ no. of a's$(w)$ = no. of b's $(w)\}$
6. $L$ = {all strings of balanced parenthesis}
7. $L = \{ww^R \mid w \in (a + b)^+\}$
8. $L = \{wcw^R \mid w \in (a + b)^+\}$
9. $L = \{a^m\ b^m\ c^n\ d^n \mid m, n \geq 0\}$
10. $L = \{a^m\ b^n\ c^n\ d^m \mid m, n \geq 0\}$

## Closure Properties of CFL's

- The context free languages are closed under:
  - (*i*) Union operation
  - (*ii*) Concatenation
  - (*iii*) Kleene closure
  - (*iv*) Reversal operation
  - (*v*) Homomorphism
  - (*vi*) Inverse homomorphism
  - (*vii*) Substitution
  - (*viii*) Init or prefix operation
  - (*ix*) Quotient with regular language.
  - (*x*) Cycle operation
  - (*x*) Union with regular language
  - (*xi*) Intersection with regular language
  - (*xii*) Difference with regular language
- **The context free languages are not closed under:**
  - (*i*) Intersection
  - (*ii*) Complement
  - (*iii*) Difference, symmetric difference (XOR), NAND, NOR and any other operation which can be reduced to intersection and complement.
  - (*iv*) Subset
  - (*v*) Superset
  - (*vi*) Infinite union

## Decision Properties of CFL's

1. **Test for membership:** Decidable.
2. **Test for Emptiness:** Decidable.
3. **Test for finiteness:** Decidable.

## Non-CFLs: Examples

(a) $L = \{a^n b^n c^n \mid n \geq 0\}$

(b) $L = \{a^n b^m c^n d^m \mid m, n \geq 0\}$

(c) $L = \{a^{n^2} \mid n \geq 0\}$

(d) $L = \left\{a^{2^n} \mid n \geq 0\right\}$

(e) $L = \left\{a^n \mid n \text{ is prime}\right\}$

(f) $L = \left\{a^{n!} \mid n \geq 0\right\}$

(g) $L = \left\{a^{2^{n^2}} \mid n \geq 0\right\}$

(h) $L = \{a^i b^j c^k \mid i \leq j \leq k\}$

(i) $L = \{w w^R w^R \mid w \in (a + b)^*\}$

(j) $L = \{ww \mid w \in (a + b)^*\}$

(k) $L = \{0^p \mid p \text{ is composite}\}$

(l) $L = \{0^i 1^j \mid j = i^2\}$

(m) $L = \{a^n b^n c^i \mid i \leq n\}$

(n) $L = \{0^i 1^j 0^k \mid j = \max(i, k)\}$

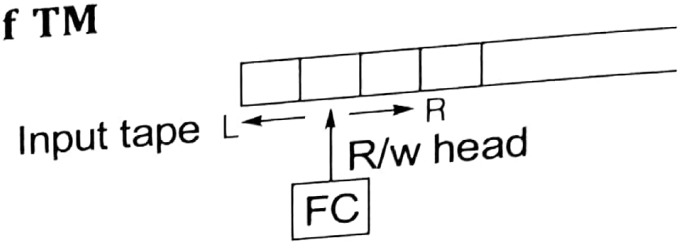
(o) $L = \{a^n b^n c^i \mid i \neq n\}$

(p) $L = \{ww^R w \mid w (a + b)^*\}$

■■■

## Turing Machine

### Configuration of TM




R/W head can move in any direction but it moves at a time only one cell, this capability is called "turn around capability"

### Specification

The turing machine can be represented as a 7 tuple as follows:

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

Where
- $Q$ = set of states.
- $\Sigma$ = input alphabet
- $\Gamma$ = tape alphabet
- $\delta$ = transition function; for DTM $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- $B$ = blank symbol
- $F$ = final states $F \subseteq Q$

### Closure Properties of Recursive Languages

- **Recursive languages are closed under the following properties:**
  - (a) Union operation
  - (b) Intersection operation
  - (c) Complement
  - (d) Concatenation operation
  - (e) Kleene closure
  - (f) Positive closure
  - (g) Difference, symmetric difference, XOR, NAND, NOR and any other operation which can be reduced to intersection and complement.
  - (h) Reversal (transpose) operation
- **Recursive languages are not closed under following properties:**
  - (a) Homomorphism of a recursive language.
  - (b) Substitution of a recursive language.

## Closure Propterties of Recursive Enumerable Languages

- Recursive Enumerable languages are closed under the following properties:
  (a) Union operation
  (b) Intersection operation
  (c) Concatenation operation
  (d) Kleene closure
  (e) Reversal operation
  (f) Positive closure
- Recursive enumerable languages are not closed under following properties:
  (a) Complement
  (b) Difference, symmetric difference, XOR, NAND, NOR and any other operation which can be reduced to intersection and complement.

## Decidability Problems for Recursive Languages

1. Membership problem: Decidable.
2. Emptiness problem: Undecidable.
3. Finiteness problem: Undecidable.
4. Equivalence problem: Undecidable.
5. Subset Problem: Undecidable.
6. Ambiguity problem: Undecidable.
7. Regularity problem: Undecidable.
8. Universality problem $L = \Sigma^*$? : Undecidable.
9. Disjointedness problem : Undecidable.

## Some decidability problems related to halting turing machines (REC languages)

1. Halting problem of a (halting) turing machine: Decidable.
2. Empty word halting problem for halting turing machine : Decidable.
3. State entry problem for halting turing machine : Decidable.

## Decidability Problems for Recursively Enumerable Languages

1. RE membership problem: Undecidable.
2. Emptiness problem: Undecidable.
3. Finiteness problem: Undecidable.
4. Equivalence problem: Undecidable.

5. Subset Problem: Undecidable.
6. Universality problem L = $\Sigma^*$?: Undecidable.
7. Ambiguity problem: Undecidable.
8. Regularity problem: Undecidable.
9. Disjointedness problem: Undecidable.

## Some decidability problems related to turing machines (RE languages)

1. Halting problem of a turing machine: Undecidable.
2. Empty word halting problem: Undecidable.
3. State entry problem: Undecidable.
4. Post correspondence problem (PCP): Undecidable.
5. Modified post correspondence problem (MPCP): Undecidable.

- **Post correspondence problem:** An instance of PCP consist of 2 lists, $A = w_1 \ldots w_k$ and $B = x_1 \ldots x_2$ of strings over some alphabet $\Sigma$. This instance of PCP has a solution if there is any sequence of integers $i_1$, $i_2 \ldots\ldots i_m$, with $m \geq 1$, such that $w_{i1}, w_{i2} \ldots\ldots w_{im} = x_{i1}, x_{i2} \ldots x_{im}$.
  The sequence $i_1, i_2 \ldots\ldots i_m$ is a solution to this instance of PCP.
- **Modified post correspondence problem:** In modified PCP there is an additional requirement on a solution that the first pair on $A$ and $B$ lists must be the first pair in the solution. More formally, an instance of modified PCP is two lists $A = w_1\, w_{i1}\, w_{i2} \ldots\ldots w_{im} = x_1\, x_{i1}\, x_{i2} \ldots\ldots x_{im}$.

**Remember:**

- ***Rice's theorem***: Every (non-trivial) question asked about RE languages is undecidable.
- NTM ≡ DTM
- L is RE iff there exist a TM to accept it
- L is REC iff there exist a TM to accept it and TM halts for all $w \in \Sigma^*$
- Both RE and REC are countable.
- Only REC has membership algorithm but no membership algorithm for RE
- If L is REC than $\bar{L}$ is also REC
- If both L and $\bar{L}$ are RE then both L and $\bar{L}$ will be REC.
- Recursive languages is turing decidable.
- Recursive enumerable language is turing recognizable.

## Identification of Formal Languages

1. $a^{n^2} \rightarrow$ CSL
2. $a^n b^n c^n \rightarrow$ CSL
3. $\{ww \mid w \in (0, 1)^*\} \rightarrow$ CSL
4. $\{a^n b^m c^p \mid n = m = p\} \rightarrow$ CSL
5. $\{ww^R w \mid w \in (0, 1)^*\} \rightarrow$ CSL
6. $\{ww^R w^R \mid w \in (0, 1)^*\} \rightarrow$ CSL
7. $\{xww \mid x\, w \in (0, 1)^+\} \rightarrow$ CSL
8. $\{w \mid n_a(w) = n_b(w) = n_c(w)\} \rightarrow$ CSL
9. $\left\{w \left| \frac{n_a(w)}{n_b(w)} = n_c(w) \right.\right\} \rightarrow$ CSL
10. $\{a^n b^m c^n d^m\} \rightarrow$ CSL
11. $\{a^n b^m c^p \mid n = m \text{ and } m = p\} \rightarrow$ CSL
12. $a^n b^{n^2} \rightarrow$ CSL
13. $\{a^n b^{n+m} c^p\} \rightarrow$ DCFL
14. $\{w \mid n_a(w) + n_b(w) = n_c(w)\} \rightarrow$ DCFL
15. $\{a^m b^n c^m\} \rightarrow$ DCFL
16. $\{a^m b^n c^n\} \rightarrow$ DCFL
17. $\{a^n b^{n+m} c^m\} \rightarrow$ DCFL
18. $\{a^n b^n c^m d^{nj}\} \rightarrow$ DCFL
19. $\{a^n b^m c^m c^n\} \rightarrow$ DCFL
20. $\{w x w^R \mid w \in \{0,1\}^*, x \in \{0, 1\}\} \rightarrow$ DCFL
21. $\{ww^R \mid w \in \{0,1\}^*\} \rightarrow$ CFL
22. $\{a^m b^n c^p \mid m = n \text{ or } n = p\} \rightarrow$ CFL
23. $\{wxw^R \mid x, w \in (0,1)^*\} \rightarrow$ Regular
24. $\{wx\, w \mid x,\ w \in (0,1)^*\} \rightarrow$ Regular
25. $\{wx\, w^R \mid x,\ w \in (0,1)^+\} \rightarrow$ Regular
26. $a^* \rightarrow$ Regular
27. $a^* b^* \rightarrow$ Regular
28. $\{a^n b^n \mid n \leq 10000\} \rightarrow$ Regular
29. $\{a^n b^n c^n \mid n = 10\} \rightarrow$ Regular
30. $\left\{a^{n^2} / n \geq 0\right\}^* \rightarrow$ Regular
31. $\{xww \mid x,\ w \in (0, 1)^*\} \rightarrow$ Regular

■■■

# Undecidability

4

## Undecidablity/Decidability Table for Formal Languages

| Problem | Regular Language | DCFL | CFL | Recursive Language | Recursively Enumerable Language |
|---|---|---|---|---|---|
| 1.Is W in L? (member ship problem) | ✓ | ✓ | ✓ | ✓ | × |
| 2.Is L = $\phi$? (emptiness problem) | ✓ | ✓ | ✓ | × | × |
| 3.Is L = Finite? (Finiteness problem) | ✓ | ✓ | ✓ | × | × |
| 4.Is L1 = L2? (equivalence problem) | ✓ | ✓ | × | × | × |
| 5.Is L1 $\subseteq$ L2? (subset problem) | ✓ | × | × | × | × |
| 6.Is 'L' regular? (regularity problem) | ✓ | ✓ | × | × | × |
| 7.Is L ambiguous (ambiguity problem) | ✓[1] | × | × | × | × |
| 8.Is L = $\Sigma^*$ (Universality problem) | ✓ | ✓ | × | × | × |
| 9.Is L1 $\cap$ L2 = $\phi$? (disjointedness prob.) | ✓ | × | × | × | × |
| 10.Is L = R? Where, R is given regular language | ✓ | ✓ | × | × | × |
| 11.Is the intersection of two languages is also a language of the same type? | ✓[1] | × | × | ✓[1] | ✓[1] |
| 12.Is the compliment of a language is also a language of the same type. | ✓[1] | ✓[1] | × | ✓[1] | × |

## Closure Properties Table

| Property | Regular | DCFL | CFL | CSL | REC | RE |
|---|---|---|---|---|---|---|
| $\cup$ | ✓ | ☒ | ✓ | ✓ | ✓ | ✓ |
| $\cap$ | ✓ | ☒ | ☒ | ✓ | ✓ | ✓ |
| $L^C$ | ✓ | ✓ | ☒ | ☒ | ✓ | ☒ |
| $\bullet$ | ✓ | ☒ | ✓ | ✓ | ✓ | ✓ |
| $L^*, L^+$ | ✓ | ☒ | ✓ | ✓ | ✓ | ✓ |
| $L^R$ | ✓ | ☒ | ✓ | No information | ✓ No information | ✓ No information |
| $h(L)$ | ✓ | ☒ | ✓ | | | |
| $h^{-1}(L)$ | ✓ | ✓ | ✓ | | | |
| $L_1 \cap R$ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| $L_1 - R$ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| $L_1 \cup R$ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| $L_1 - L_2$ | ✓ | ☒ | ☒ | ☒ | ✓ | ☒ |

## Some Decidable Problems Related to Theory Machines:

1. State of TM after k(finite) steps? → Decidable
2. Will a string w be accepted by TM after k(finite) steps? → Decidable
3. Will a TM hang within k(finite) steps? → Decidable
4. Whether a given TM will ever make a left move? → Decidable
5. Whether a given TM will print some non-blank character? → Decidable
6. PCP and MPCP problems on unary alphabet → Decidable.

■■■