UNIT-2

Regular Grammar

A regular grammar G is said to be regular (type 3) if all its productions are type 3 productions.

A production of the form $A \to \alpha$ or $A \to aB$, where $A, B \in V_N$ and $a \in \Sigma$,is called type-3 production.

A production of the type $S \to \epsilon$ is allowed in type-3 grammar but in this case S should not appear on the right hand side of any production.

Regular Expressions

1. Any terminal symbol (i.e. an element of Σ), ϵ and ϕ are regular expressions. When we view a in Σ as a regular expression, we denote it by $\boldsymbol{a}$.
2. The union of any two regular expressions $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ written as $\boldsymbol{R_1} + \boldsymbol{R_2}$ is also a regular expression.
3. The concatenation of any two regular expressions $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ written as $\boldsymbol{R_1 R_2}$ is also a regular expression.
4. The iteration (or closure) of a regular expression $\boldsymbol{R}$, written as $\boldsymbol{R}^*$, is also a regular expression.
5. If R is a regular expression, then (R) is also a regular expression.
6. The regular expression over Σ are precisely those obtained recursively by the application of the rules 1 to 5 once or several times.

Operations in Regular Expressions

1. $R^+ \cup R^* = R^*$
2. $R^+ \cap R^* = R^+$
3. $R^*.R^+ = R^+$
4. $(R^*)^* = R^*$
5. $(R^*)^+ = R^*$
6. $(R^+)^* = R^*$
7. $(a^* + b^*)^* = (a + b)^*$
8. $(a^* + b)^* = (a + b)^*$
9. $(a + b^*)^* = (a + b)^*$
10. $(a.b)^* = (a + b)^*$
11. $(a^*.b^*)^* = (a + b)^*$
12. $(a^*.b)^* = (a + b)^*$
13. $(a.b^*)^* = (a + b)^*$

Pumping Lemma for Regular Sets

<u>Theorem</u> :

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton with n states. Let L be the regular set accepted by M. Let $w \in L$ and $|w| \geq m$. If $m \geq n$, then there exists x, y, z such that $w = xyz, y \neq \epsilon$ and $xy^iz \in L$ for each $i \geq 0$.

Application of Pumping Lemma

The theorem can be used to prove that certain sets are not regular.

Steps :

1. Assume that L is regular. Let n be the number of stats in the corresponding finite automaton.
2. Choose a string w such that $|w| \geq n$. Use pumping lemma to write $w = xyz$ with $|xy| \leq n$ and $|y| > 0$.
3. Find the suitable integer i such that $xy^iz \notin L$. This contradicts our assumption. Hence L is not regular.

Regular Sets and Regular Grammars

Regular Set

Any set represented by a regular expression is called a *regular set*.

Context Free Languages

A grammar G is Context Free Grammar (CFG) if its every production is of the form $A \to \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup \Sigma)^*$.

A language generated by Context Free Grammar is called as Context Free Language. It has applications in many programming languages, in particular most arithmetic expressions are generated by CFG.

CFL are applied in parser design. They are also useful for describing block structures in programming languages.

Derivation

The process of deriving a string is known as derivation.

Leftmost Derivation

A derivation $A \overset{*}{\Rightarrow} w$ is called a *leftmost derivation* if we apply production only to the leftmost variable at every step.

Rightmost Derivation

A derivation $A \overset{*}{\Rightarrow} w$ is called a *rightmost derivation* if we apply production only to the rightmost variable at every step.

Derivation Tree / Syntax Tree / Parse Tree

The derivations in a CFG can be represented using trees. Such trees representing derivations are called derivation trees.

OR

The graphical representation of a derivation is known as a derivation tree.

Definition :

A derivation tree (Also called a parse tree) for a CFG $G = (V_N, \Sigma, P, S)$ is a tree satisfying the following conditions :

1. Every vertex has a label which is variable or terminal or ϵ.
2. The root has label S.
3. The label of an internal vertex is a variable.
4. If the vertices $n_1, n_2, n_3, \dots, n_k$ written with labels $X_1, X_2, \dots, X_k$ are the sons of vertex n with label A, then $A \rightarrow X_1 X_2 \dots X_k$ is a production in P.
5. A vertex n is a leaf if its label is $a \in \Sigma$ or ϵ. n is the only son of its father if its label is ϵ.

Ambiguity in Context Free Grammars

Ambiguous String

A terminal string $w \in L(G)$ is ambiguous if there exists two or more derivation trees for w (or there exist two or more leftmost derivations of w).

Ambiguous Grammar

A context free grammar G is ambiguous if there exists some $w \in L(G)$, which is ambiguous.

Simplification of Grammar

The process of deleting and eliminating useless symbols, unit productions and null productions is known as *simplification of CFG.*

The production of the form $A \to \epsilon$ is known as *null production* or *empty production*. We try to remove them by replacing equivalent derivations.

The production of the form $A \to B$ where $A, B \in V_N, |A| = |B| = 1$, is known as *unit production.*

The variables which are not involved in the derivation of any string are known as *useless symbols*. For this –

- Select the variable which cannot be reached from the start symbol of the grammar and remove them along with their production.
- Select the variable which is reachable from the start symbol, but which does not derive any terminal and remove them along with their productions.

Normalization of CFG

Chomsky Normal Form

A context free grammar G is in Chomsky Normal Form if every production is of the form $A \to \alpha$ or $A \to BC$ and $S \to \epsilon$ is in G if $\epsilon \in L(G)$. When ϵ is in $L(G)$ we assume that S does not appear on the R.H.S. of any production.

For example :

Consider G whose productions are $S \to AB|\epsilon$, $A \to \alpha$, $B \to b$. Then G is in Chomsky Normal Form.

Remark :

For a grammar in CNF, the derivation tree has the following property : Every node has at most two descendants – either two internal vertices or a single leaf.

Greibach Normal Form

A context free grammar is in Greibach Normal Form if every production is of the form

$A \to a\alpha$, where $\alpha \in V_N^*$ and $a \in \Sigma$ (α can be ϵ also) and $S \to \epsilon$ is in G if $\epsilon \in L(G)$. When ϵ is in $L(G)$ we assume that S does not appear on the R.H.S. of any production.

For example :

Consider G whose productions are $S \to aAB|\epsilon$, $A \to bC$, $B \to b$, $C \to c$. Then G is in Greibach Normal Form.

Remark :

A grammar in GNF is a natural generalisation of a regular grammar. So, for a grammar in GNF or a regular grammar, we get a (single) terminal and a string of variables (possibly ϵ) on application of a production (with the exception of $S \to \epsilon$)

Pushdown Automaton

A pushdown automaton consists of

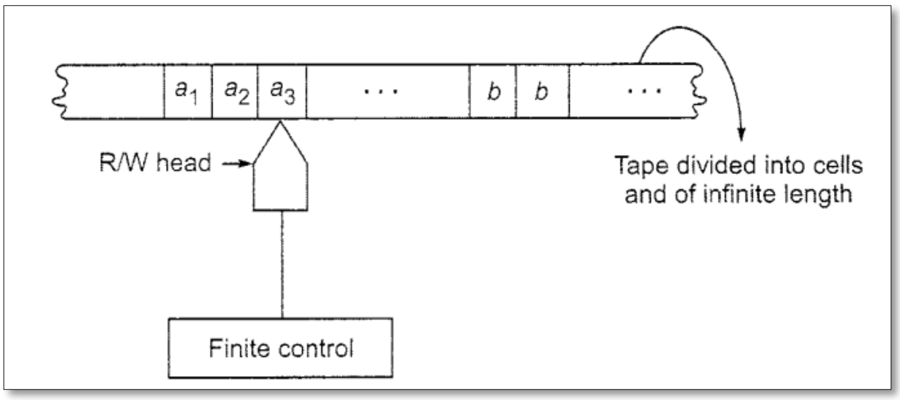
1. a finite non-empty set of states denoted by Q
2. a finite non-empty set of input symbols denoted by Σ
3. a finite non-empty set of pushdown symbols denoted by Γ
4. an initial state denoted by q_0
5. a special pushdown symbol called *initial symbol* on the pushdown store demoted by z_0
6. a set of final states F which is subset of Q and
7. a transition function δ from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the set of finite subsets of $Q \times \Gamma^*$.

Symbolically, a PDA is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$.

UNIT – 3

Q. Explain the Turing Machine model with the help of diagram.

The Turing Machine can be thought of as finite control connected to a R/W (read/write) head. It has one tape which is divided into a number of cells. The block diagram of the basic model for the Turing Machine is given below.



Turing Machine Model

Each cell can store only one symbol. The input to and the output from the finite state automaton are affected by the R/W head which can examine one cell at a time. In one move, the machine examines the present symbol under the R/W head on the tape and the present state of an automaton to determine

(i) a new symbol to be written on the tape in the cell under the R/W head,

(ii) a motion of the R/W head along the tape: either the head moves one cell left (L) or one cell right (R),

(iii) the next state of the automaton, and

(iv) whether to halt or not.

Q. Define : Turing Machine.

Definition

A Turing Machine M is a 7-tuple, namely $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$, where

- Q is a finite nonempty set of states.
- Γ is a finite nonempty set of tape symbols.
- $b \in \Gamma$ is the blank.
- Σ is a nonempty set of input symbols and is a subset of Γ.
- δ is the transition function mapping (q, x) onto (q', y, D) where D denotes the direction of movement of R/W head: D = L or R according as the movement is to the left or right.
- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is the set of final states.

Note :

(1) The acceptability of a string is decided by the reachability from the initial state to some final state. So the final states are also called as accepting states.
(2) δ may not be defined for some elements of $Q \times \Gamma$.

Q. Write a short note on Representation of a Turing Machine by Instantaneous Descriptions.

Q. Write a short note on Representation of a Turing Machine by Transition Table.

Q. Write a short note on Representation of a Turing Machine by Transition Diagram.

Representation of Turing Machines

We can describe a Turing Machine using

(i) Instantaneous Descriptions
(ii) Transition Table
(iii) Transition Diagrams

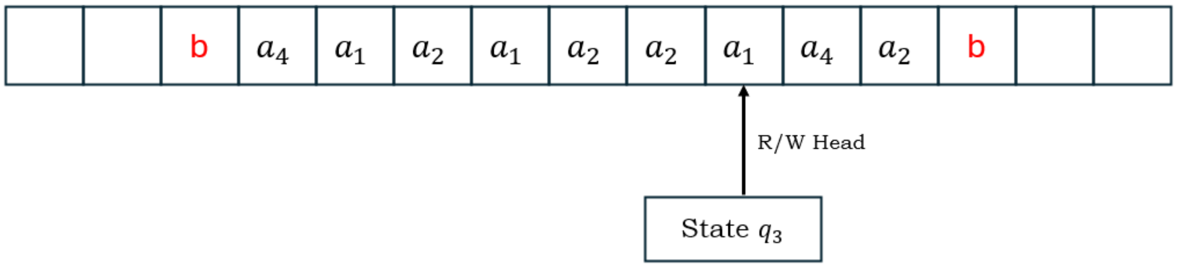
(i) Representation by Instantaneous Descriptions

Definition

An Instantaneous Description of a Turing machine M is a string $\alpha\beta\gamma$, where β is the present state of M, the entire input string is split as $\alpha\gamma$, the first symbol of γ is the current symbol a under the R/W head and γ has all the subsequent symbols of the input string, and the string α is the substring of the input string formed by all the symbols to the left of a.

For Example :

Consider a snapshot of Turing Machine as shown in the following diagram.



So, the input string is $a_4a_1a_2a_1a_2a_2a_1a_4a_2$.

$$\therefore \alpha\gamma = a_4a_1a_2a_1a_2a_2a_1a_4a_2$$

The present state is q_3. $\therefore \beta = q_3$

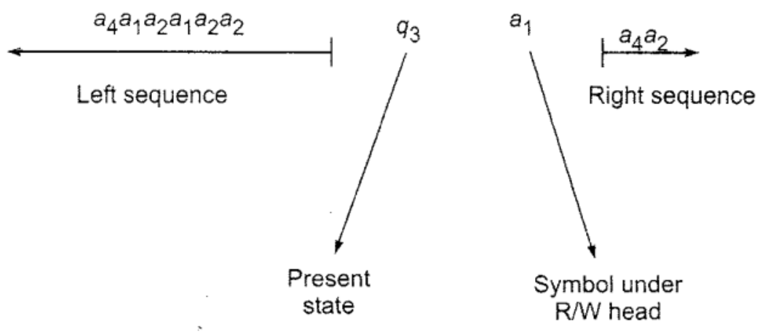
The nonblank symbols to the left of a_1 form the α.

$$\therefore \alpha = a_4a_1a_2a_1a_2a_2$$

The present symbol under the R/W head is a_1. So, the symbol a_1 and symbols to the right of a_1 form the γ.

$$\therefore \gamma = a_1a_4a_2.$$

Thus, the Instantaneous Description is as follows :



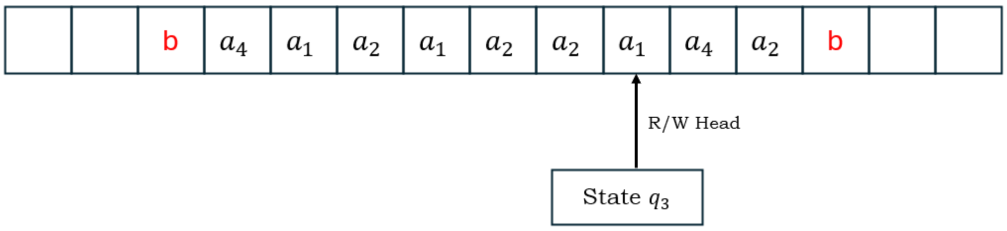
Moves in Turing Machine

The $\delta(q_3, a_1)$ induces change in ID of the Turing Machine called a Move.

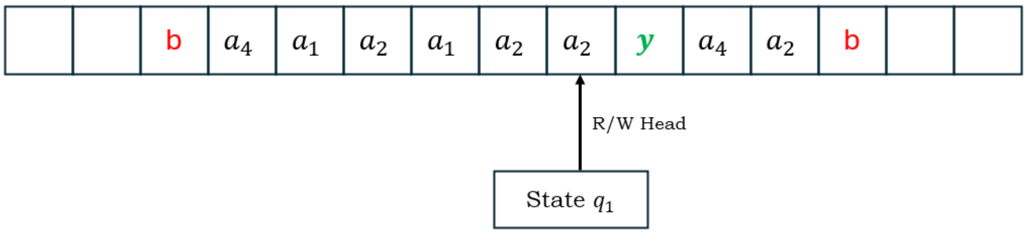
<u>Case-I : Turing Machine moves to Left</u>

If $\delta(q_3, a_1) = (q_1, y, L)$ then

ID before processing : $a_4\ a_1\ a_2\ a_1\ a_2\ a_2\ \boldsymbol{q_3}\ a_1\ a_4\ a_2$



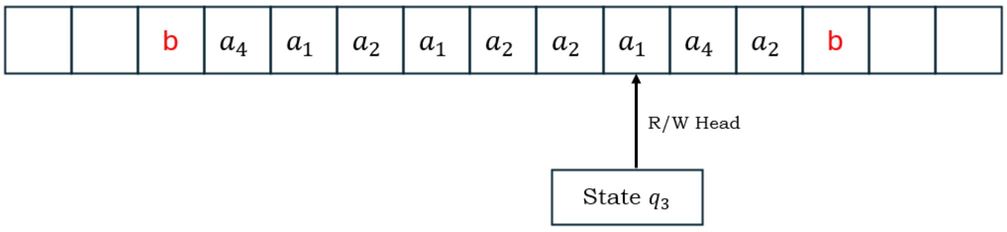
ID after processing : $a_4\ a_1\ a_2\ a_1\ a_2\ \boldsymbol{q_1}\ a_2\ y\ a_4\ a_2$



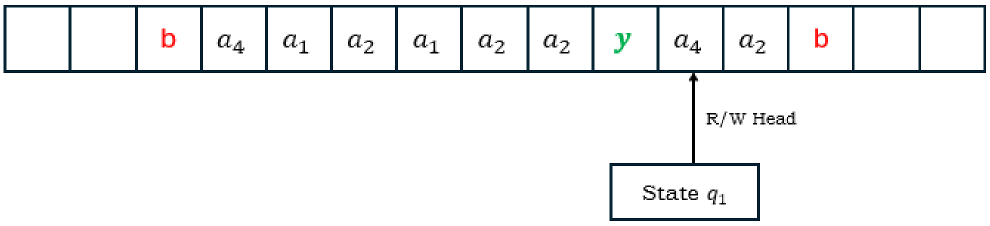
Case-II : Turing Machine moves to Right

If $\delta(q_3, a_1) = (q_1, y, R)$

ID before processing : $a_4\ a_1\ a_2\ a_1\ a_2\ a_2\ \boldsymbol{q_3}\ a_1\ a_4\ a_2$



ID after processing : $a_4\ a_1\ a_2\ a_1\ a_2\ a_2\ y\ \boldsymbol{q_1}\ a_4\ a_2$



(ii) Representation by Transition Table

In this type of representation of the Turing Machine, we give the definition of δ in the form of a table called as the Transition Table.

Suppose a Turing Machine has 3 states q_0, q_1 *and* q_2 where q_0 is the initial state and q_2 is the final state. The tape symbols are $0, 1, b$. Then the transition table will have 3 rows and 3 columns. The initial state is marked with $\rightarrow$ and the final state will be circled.

Present State	Tape Symbols		
	0	1	b
$\rightarrow q_0$			
q_1			
$\bigcirc\!\!\!\!\!\; q_2$			

Suppose we have $\delta(q_0, 0) = (q_1, y, R)$.

which means when if the symbols under the head is 0 and the present state is q_0 then –

(i) y is written in the current cell
(ii) the R/W head moves to the right and
(iii) the machine enters into the new state q_1

In order to represent this in the form of table we will write (q_1, y, R) in q_0 row and 0 column in the Transition Table.

Present State	Tape Symbols		
	0	1	b
$\rightarrow q_0$	(q_1, y, R)		
q_1			
$\bigcirc\!\!\!\!\!\; q_2$			

(iii) Representation by Transition Diagram

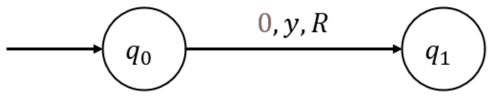
In this type of representation of the Turing Machine, states are represented by the vertices and directed edges are used to represent transition of states.

Suppose we have $\delta(q_0, 0) = (q_1, y, R)$.

which means when if the symbols under the head is 0 and the present state is q_0 then –

(i) y is written in the current cell
(ii) the R/W head moves to the right and
(iii) the machine enters into the new state q_1

Then, we represent it as follows :



The initial state is indicated by → and the final state is double circled.

String acceptability by a Turing Machine

A string is accepted by a Turing Machine if it halts in Accepting State.

A string is rejected by a Turing Machine if it halts in non-accepting state or if machine enters into an infinite loop.

Running Time of a Turing Machine

Let M be a Turing Machine and w an input string. The running time of M on input w, is the number of steps that M takes before halting. If M does not halt on an input string w, then the running time of M on w is infinite.

Time Complexity of Turing Machine

The time complexity of Turing Machine M is the function $T(n)$, n being the input size, where $T(n)$ is defined as the maximum of the running time of M over all inputs w of size n.

Turing Machine Construction Techniques

Construction of Turing Machine needs complete set of states used for storing information.

Turing machine with stationary head

- In the definition of the TM we define $\delta(q_i, a)$ as (q_j, y, D) where D = L or R. So, the head moves to the left or right after reading some input symbol say 'a'.

- Suppose we want to include the option that the head can continue to be in the same cell for some input symbol say 'a'. Then we define $\delta(q_i, a)$ as $\delta(q_j, y, S)$.
- This means that the TM on reading the input symbol 'a', changes the state to q_j and writes 'y' in the current cell in place of 'a' and continues to remain in the same cell.

Storage in the State

- We are using a state in FA, PDA or TM to 'remember' things. We can use state to store a symbol as well.
- So, the state becomes a pair (q_i, a) where q_i is the state (in the usual sense) and 'a' is the tape symbol stored in (q_i, a). So the new set of states becomes $Q \times \Gamma$.

Multiple Track Turing Machine

- In standard TM a single tape is used. In multiple track TM, a single tape is assumed to be divided into several tracks.
- Now the tape alphabet is required to consist of k-tuples of tape symbols where k represents the number of tracks.
- Hence the only difference between the standard TM and the TM with multiple tracks is the set of tape symbols.
- In case of the standard TM, tape symbols are elements of Γ. In the case of TM with multiple track, it is Γ^k.

Subroutines

- Subroutines are used in computer languages, when some task has to be done repeatedly. We can implement this in TM as well.
- First a TM for the subroutine is written. This will have initial state and a 'return' state.
- After reaching the return state, there is a temporary halt.
- For using a subroutine, moves are effected to enter the initial state for the subroutine (when the return state of the subroutine is reached) and to return to the main program of TM.

Variants of the Turing Machine

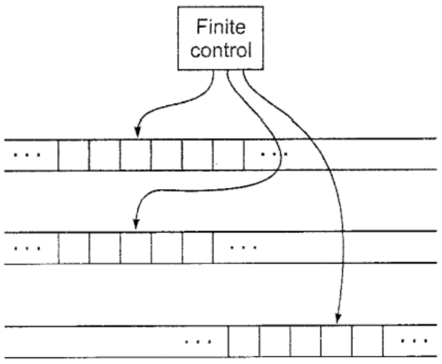
Multitape Turing Machine

A Multitape Turing Machine has a finite set Q of states, an initial state q_0, a subset F of Q called the set of final states, a set P of tape symbols, a new symbol b, not in P called the blank symbol. (We assume that $\Sigma \subseteq \Gamma$ and $b \notin \Sigma$.)

There are k tapes, each divided into cells. The first tape holds the input string w. Initially all the other tapes hold the blank symbol.

Initially, the head of the first tape (input tape) is at the left end of the input w. All the other heads can be placed at any cell initially.

δ is a partial function from $Q \times \Gamma^k$ into $Q \times \Gamma^k \times \{L, R, S\}^k$. We use implementation description to define δ.



In a typical move :

(i) M enters a new state
(ii) On each tape, a new symbol is written in the cell under the head.
(iii) Each tape heads move independently; some move to the left; some move to the right and the remaining do not move.

An initial ID has the initial state q_0, the input string w in the first step (input tape), empty strings of $b's$ in the remaining $k - 1$ tapes. An accepting ID has a final state, some strings in each of the k tapes.

Non Deterministic Turing Machines

A nondeterministic Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$ where

- Q is a finite non-empty set of states.
- Γ is a finite non-empty set of tape symbols.
- b ∈ Γ is called blank symbol.
- Σ is a nonempty subset of Γ, called the set of input symbols. We assume that $b \notin \Sigma$.
- q_0 is the initial state.
- δ is the partial function from $Q \times \Gamma$ into the power set of $Q \times \Gamma \times \{L, R\}$

Linear Bound Automata

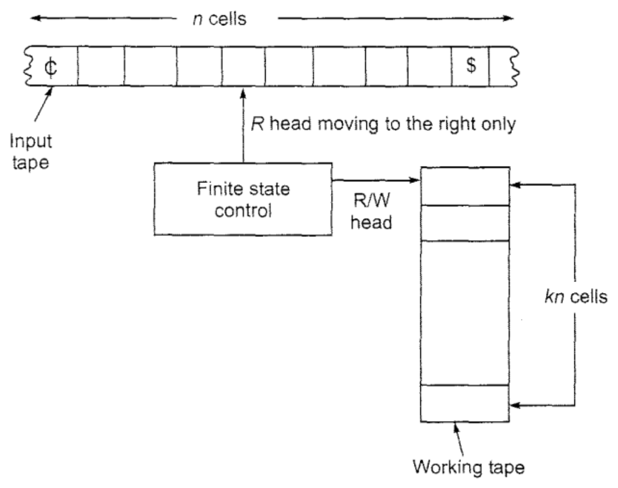


Fig. 9.11 Model of linear bounded automaton.

Linear Bound Automaton is a restricted form of Turing Machine. It accepts the set of context-sensitive languages. It is called as the linear bound automaton (LBA) because a linear function is used to restrict (to bound) the length of the tape.

A linear bounded automaton is a non-deterministic Turing Machine which has a single tape whose length is not infinite but bounded by a linear function of the length of the input string. The models can be descried formally by the following set format :

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

1. Q is a finite nonempty set of states.
2. Σ is a nonempty set of input symbols and is a subset of Γ.
3. Γ is a finite nonempty set of tape symbols.
4. δ is the transition function mapping (q, x) onto (q', y, D) where D denotes the direction of movement of R/W head: $D = L$ or R according as the movement is to the left or right.
5. $q_0 \in Q$ is the initial state
6. b ∈ Γ is the blank
7. ¢ is a called the left-hand marker which is entered in the left most cell of the input tape and prevents the R/W head from getting off the leftmost cell of the tape.
8. $ is called the right-end marker which is entered in the rightmost cell of the input tape and prevents the R/W head from getting off the right end of the tape. (Both end markers should not appear on any other cell within the input tape and the R/W head should not print any other symbol over both the endmarkers.)
9. $F \subseteq Q$ is the set of final states.

Linear Bounded Automata and Languages

A linear bounded automaton M accepts a string w, if after starting at initial state with R/W head reading the left-end marker, M halts over the right-end marker in a final state. Otherwise, w is rejected.

Some Definitions

Procedure

A *procedure* for solving a problem is a finite sequence of instructions which can be mechanically carried out given any input.

Algorithm

An *algorithm* is a procedure that terminates after a finite number of steps for any input.

Recursive Set

A set X is recursive if we have an algorithm, to determine whether a given element belongs to X or not.

Recursively Enumerable Set

A recursively enumerable set is a set X for which we have a procedure to determine whether a given element belongs to X or not. It is clear that recursive set is recursively enumerable.

Recursively Enumerable Language

A language $L \subseteq \Sigma^*$ is recursively enumerable if there exist a TM M, such that $L = T(M)$.

Recursive Language

A language $L \subseteq \Sigma^*$ is recursive if there exist some TM M, that satisfies the following two conditions :

(i) If $w \in L$ then M accepts w (that is, reaches an accepting state on processing w) and halts.

(ii) If $w \notin L$ then M eventually halts, without reaching an accepting state.

NOTE :

This definition formalizes the notion of an 'algorithm'. An algorithm, in usual sense, is a well-defined sequence of steps that always terminates and produces an answer. The conditions (i) and (ii) of definition assure that the TM always halts, accepting w under condition (i) and not accepting under condition (ii).

So a TM, defining a recursive language always halts eventually just as an algorithm eventually terminates.

Definition

A problem with two answers (Yes/No) is decidable if the corresponding language is recursive. In this case, the language L is also called *decidable*.

Definition

A problem/language is *undecidable* if it is not decidable.

Church-Turing Thesis

In 1936, A method named as lambda-calculus was created by Alonzo Church in which the Church numerals are well defined, i.e. the encoding of natural numbers. Also in 1936, Turing machines (earlier called theoretical model for machines) was created by Alan Turing, that is used for manipulating the symbols of string with the help of tape.

Turing machine is defined as an abstract representation of a computing device such as hardware in computers. Alan Turing proposed Logical Computing Machines (LCMs), i.e. Turing's expressions for Turing Machines. This was done to define algorithms properly. So, Church made a mechanical method named as '*M*' for manipulation of strings by using logic and mathematics.

This method *M* must pass the following statements:

- Number of instructions in M must be finite.
- Output should be produced after performing finite number of steps.
- It should not be imaginary, i.e. can be made in real life.
- It should not require any complex understanding.

Using these statements Church proposed a hypothesis called Church's Turing thesis that can be stated as: "The assumption that the intuitive notion of computable functions can be identified with partial recursive functions."

In 1930, this statement was first formulated by Alonzo Church and is usually referred to as Church's thesis, or the Church-Turing thesis. However, this hypothesis cannot be proved.

The recursive functions can be computable after taking following assumptions:

1. Each and every function must be computable.
2. Let 'F' be the computable function and after performing some elementary operations to 'F', it will transform a new function 'G' then this function 'G' automatically becomes the computable function.
3. If any functions that follow above two assumptions must be states as computable function.

Difference between Turing Machine and Universal Turing Machine

Turing Machine was first described by Alan Turing in the year 1936. It was primarily invented to investigate the computability of a given problem. It accepts type-0 grammar which is Recursively Enumerable language. The Turing machine has a tape of infinite length where we can perform read and write operations. The infinite cells of the Turing machine can contain input symbols and blanks. It has a head pointer that can move in any direction, it points to the cell where the input is being read.

Universal Turing Machine simulates a Turing Machine. Universal Turing Machine can be considered as a subset of all the Turing machines, it can match or surpass other Turing machines including itself. Universal Turing Machine is like a single Turing Machine that has a solution to all problems that is computable. It contains a Turing Machine description as input along with an input string, runs the Turing Machine on the input and returns a result.

	Turing Machine	**Universal Turing Machine**
1	It is a mathematical model of computation it manipulates symbols on the tape according to the rules defined	Universal Turing Machine is like a single Turing Machine that has a solution to all problem that is computable
2	A program can be compared to a Turing Machine	Programmable Turing Machine is called Universal Turing Machine
3	Turing machine's temporary storage is tape. The infinite cells of the Turing machine can contain input symbols and blanks.	Universal Turing Machine contains Turing Machine description as input along with an input string, runs the Turing Machine on the input and returns the result.
4	Turing machines help us understand the fundamental limitations of mechanical computation power	Although developed for theoretical reasons, it helped in the development of stored program computers
5	A Turing machine is a formal model of a computer with a fixed program	Universal Turing Machine provides a solution to problems that are computable
6	It does not minimize the space complexity	It minimizes space complexity
7,	Transition function which Turing Machine performs is defined as: δ X T -> Q X T X {L,R}, where δ is the transition function	The transition function is $Q \times T \rightarrow Q \times T \times \{L, R\}$, where Q is a finite set of states, T is the tape of the alphabet
8	In the set theory point of view, all Turing machines form a set of the all the device that accepts type 0 grammar	Universal Turing Machine is a subset of all the Turing Machines

Halting Problem

The Halting problem – Given a program/algorithm will ever halt or not?

Halting means that the program on certain input will accept it and halt or reject it and halt and it would never go into an infinite loop.

Basically, halting means terminating. So can we have an algorithm that will tell that the given program will halt or not. In terms of Turing machine, will it terminate when run on some machine with some particular given input string.

The answer is no we cannot design a generalized algorithm which can appropriately say that given a program will ever halt or not?