Turing Machine

Module 3

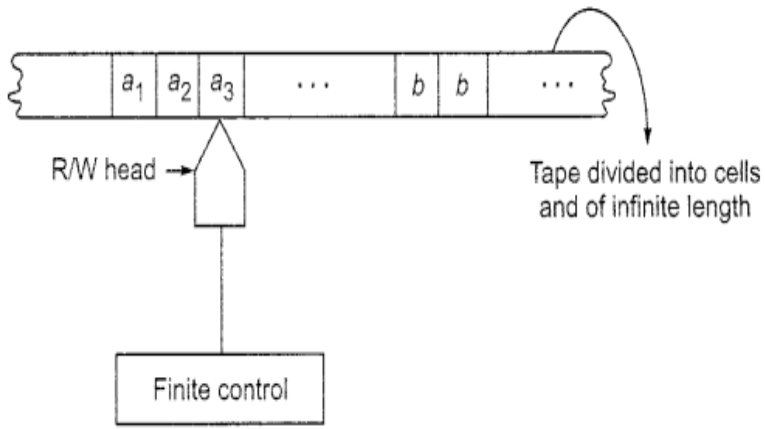
Dr. A K Yadav

April 28, 2022

Outline

Turing Machine

- A Turing Machine's storage can be visualized as a single, one dimensional array of cells, each of which can hold a single symbol.
- This array extends infinitely in both directions.
- Information can be read and changed in any order, such storage device is called **Tape**.
- Turing Machine has neither an input file nor any special output mechanism, whatever input and output is required will be done on machine's tape.



Turing Machine

- A Turing machine M is defined by
  M=(Q, $\Sigma, \tau, \delta, q_0, \square, F$)
  where, Q = set of internal states
  Σ = Input alphabet; $\Sigma \subseteq \tau - \{\square\}$
  τ = finite set of symbols called tape alphabet
  δ = transition function
  $Q \times \tau \rightarrow Q \times \tau \times \{L, R\}$
  $\square \in \tau$ = special symbol called blank
  $q_0 \in Q$ =initial state
  $F \subseteq Q$ = set of final states

Turing Machine

- A Turing machine M is defined by
  M=(Q, $\Sigma, \tau, \delta, q_0, \square, F$)
  where, Q = set of internal states
  Σ = Input alphabet; $\Sigma \subseteq \tau - \{\square\}$
  τ = finite set of symbols called tape alphabet
  δ = transition function
  $Q \times \tau \rightarrow Q \times \tau \times \{L, R\}$
  $\square \in \tau$ = special symbol called blank
  $q_0 \in Q$ =initial state
  $F \subseteq Q$ = set of final states

Turing Machine

- The current state of the control unit and the current tape symbol being read determines the new state of the control unit and new tape symbol which replaces the old one and move the head L or R.
- $\delta(q_0, a) = (q_1, b, R)$

- 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 the accepting states.
- A Turing machine is said to **halt** whenever it reaches a configuration for which δ is not defined.
  $\Longrightarrow$ No transitions are defined for any final state, so the Turing machine will halt whenever it enters a final state.

Turing Machine

- Consider the Turing machine defined by: Q=$\{q_0, q_1\}$, Σ=$\{0,1\}$, $\tau = \{0, 1, \Box\}$, F =$\{q_1\}$ and
  $\delta(q_0, 0) = (q_0, 1, R)$
  $\delta(q_0, 1) = (q_0, 0, R)$
  $\delta(q_0, \Box) = (q_1, \Box, L)$

Turing Machine

- Consider the Turing machine defined by: Q=$\{q_0, q_1\}$, Σ=$\{0,1\}$, $\tau = \{0, 1, \Box\}$, F =$\{q_1\}$ and
  $\delta(q_0, 0) = (q_0, 1, R)$
  $\delta(q_0, 1) = (q_0, 0, R)$
  $\delta(q_0, \Box) = (q_1, \Box, L)$
- Q=$\{q_0, q_1, q_2\}$, Σ=$\{a,b\}$, $\tau = \{a, b, \Box\}$

Turing Machine

- Consider the Turing machine defined by: Q=$\{q_0, q_1\}$, Σ=$\{0,1\}$, $\tau = \{0, 1, \Box\}$, F =$\{q_1\}$ and
  $\delta(q_0, 0) = (q_0, 1, R)$
  $\delta(q_0, 1) = (q_0, 0, R)$
  $\delta(q_0, \Box) = (q_1, \Box, L)$
- Q=$\{q_0, q_1, q_2\}$, Σ=$\{a,b\}$, $\tau = \{a, b, \Box\}$
  Let F be empty. Define δ by:
  $\delta(q_0, a) = (q_1, a, R)$
  $\delta(q_0, b) = (q_1, a, R)$
  $\delta(q_0, \Box) = (q_1, \Box, L)$
  $\delta(q_1, a) = (q_0, a, L)$
  $\delta(q_1, b) = (q_0, b, L)$
  $\delta(q_1, \Box) = (q_2, \Box, R)$

Representation of Turing Machines

Turing machine can be represented by tree ways:

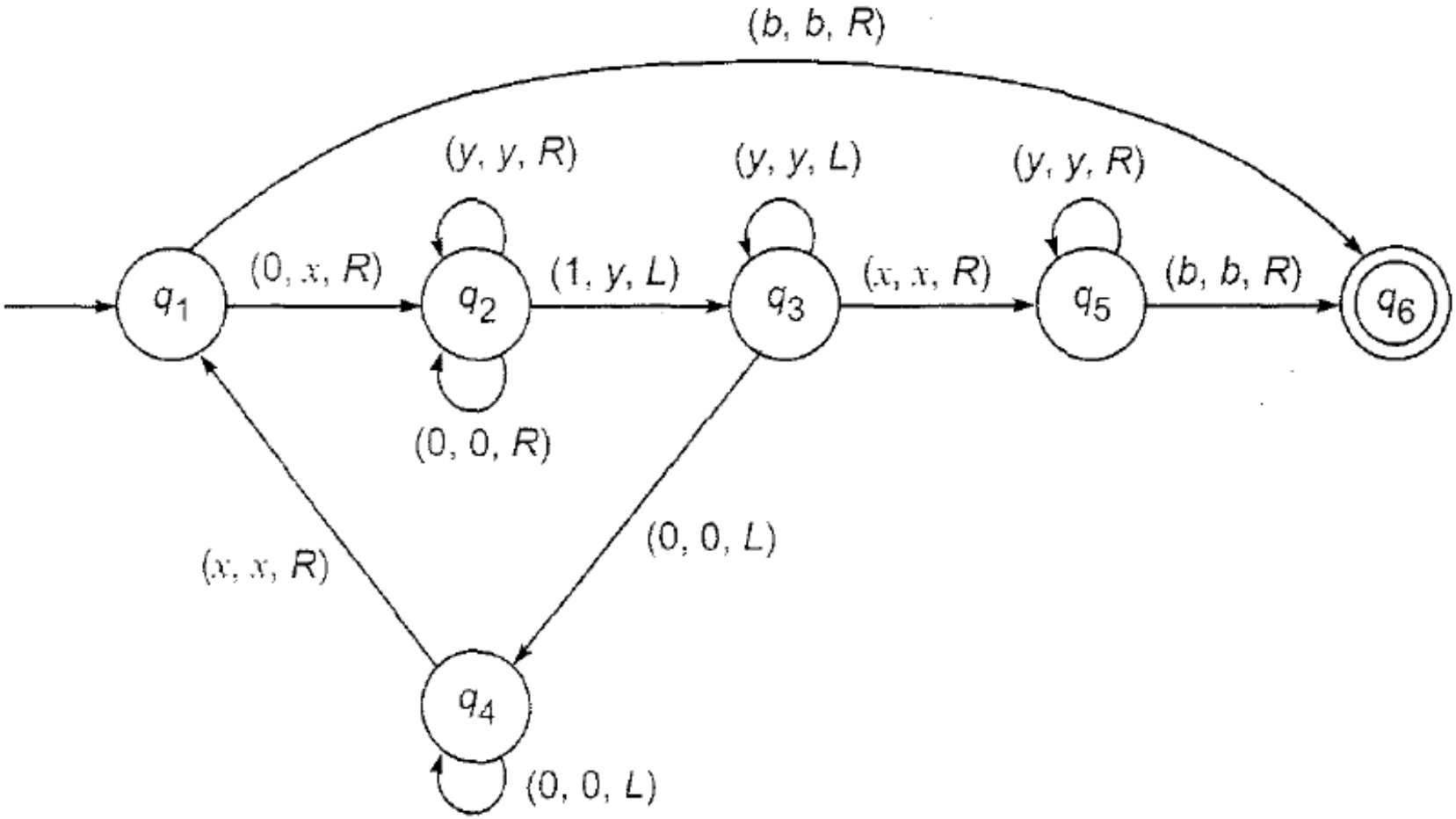
- Instantaneous Descriptions (ID) using move-relations
  - $\delta(q_1, x_i) = (q_2, y, R)$
    $x_1 x_2 \ldots q_1 x_i \ldots x_n \vdash x_1 x_2 \ldots y q_2 x_{i+1} \ldots x_n$
  - $\delta(q_1, x_i) = (q_2, y, L)$
    $x_1 x_2 \ldots q_1 x_i \ldots x_n \vdash x_1 x_2 \ldots q_2 x_{i-1} y \ldots x_n$
  - $\delta(q_1, x_n) = (q_2, y, R)$
    $x_1 x_2 \ldots q_1 x_n \vdash x_1 x_2 \ldots y q_2 \square$
    Because the tape is of infinite length having $\square$.
  - $\delta(q_1, x_1) = (q_2, y, L)$
    $q_1 x_1 x_2 \ldots x_n \vdash q_2 y x_2 \ldots x_n$
    Because we prevent the machine from going off the left-hand end of the tape
- Transition table

Representation of Turing Machines

Present state	Tape symbol		
	b	0	1
$\rightarrow q_1$	$1Lq_2$	$0Rq_1$	
q_2	bRq_3	$0Lq_2$	$1Lq_2$
q_3		bRq_4	bRq_5
q_4	$0Rq_5$	$0Rq_4$	$1Rq_4$
(q_5)	$0Lq_2$		

- Transition diagram (transition graph)

Representation of Turing Machines



Language acceptability of Turing Machine

- Consider the Turing machine $M = (Q, \Sigma, \tau, \delta, q_0, \Box, F)$. A string $w \in \Sigma^*$ is said to be accepted by M if $q_o w \vdash^* \alpha_1 q \alpha_2$ for some $q \in F$ and $\alpha_1, \alpha_2 \in \tau^*$
- M does not accept w if the machine M either halts in a non-accepting state or does not halt.
- There are other equivalent definitions of acceptance by the Turing machine, we will not discuss them now.

Design of TM

The basic guidelines for designing a Turing machine:

- The fundamental objective in scanning a symbol by the R/W head is to know what to do in the future. The machine must remember the past symbols scanned. The Turing machine can remember this by going to the next unique state.
- The number of states must be minimized. This can be achieved by changing the states only when there is a change in the written symbol or when there is a change in the movement of the R/W head.

Design of Turing Machine

Question: For $\Sigma = \{a,b\}$, design a Turing machine that accepts $L = \{a^n b^n : n \geq 1\}$

Solution:

- Start with left most 'a', replace it by 'x'
- Travel righ to find left most 'b', replace it by 'y'.
- Move left again to find left most 'a', replace by 'x, then again right to left most 'b', replace by 'y'.
- Continue moving right and left till no 'a' and 'b' remains, then the string must be in L.

Design of Turing Machine

Q=$\{q_0, q_1, q_2, q_3, q_4\}$, F=$\{q_4\}$
$\Sigma = \{a, b\}$, $\tau = \{a, b, x, y, \Box\}$
$\delta(q_0, a) = (q_1, x, R) \implies$ replaces 'a' by 'x'
$\delta(q_1, a) = (q_1, a, R) \implies$ move right
$\delta(q_1, y) = (q_1, y, R) \implies$ move right
$\delta(q_1, b) = (q_2, y, L) \implies$ 'a' paired with 'b'
Move left to find 'x'
$\delta(q_2, y) = (q_2, y, L) \implies$ move left
$\delta(q_2, a) = (q_2, a, L) \implies$ move left
$\delta(q_2, x) = (q_0, x, R) \implies$ placed at first 'a'
Check for all 'a' and 'b' are replaced
$\delta(q_0, y) = (q_3, y, R)$
$\delta(q_3, y) = (q_3, y, R)$
$\delta(q_3, \Box) = (q_4, \Box, R)$
For input 'aabb'

$q_0aabb \vdash xq_1abb \vdash xaq_1bb \vdash xq_2ayb \vdash q_2xayb \vdash xq_0ayb \vdash xxq_1yb \vdash xxyq_1b \vdash$

$xxq_2yy \vdash xq_2xyy \vdash xxq_0yy \vdash xxyq_3y \vdash xxyyq_3 \vdash xxyy\Box q_4\Box$

Design of Turing machine

Question: Design a Turing Machine that accepts L=$\{a^n b^n c^n : n \geq 1\}$

Turing Computable

A function 'f' with domain 'D' is said to be Turing computable or just computable, if there exists some Turing machine M=(Q, $\Sigma, \tau, \delta, q_0, \square$, F) such that $q_0 w \vdash_M^* q_f f(w), q_f \in F$ for all w$\in D$.

Turing Computable

Example

Given two positive integers 'x' and 'y'. Design a Turing machine that computes x+y

Solution: Using Unary notation in which any positive integer 'x' is represented by w(x) $\in \{1\}^+$ such that |w(x)|=x.

So, the required machine is: $q_0w(x)0w(y) \vdash *q_fw(x+y)0$

Steps: Move the separating 0 to right end of w(y)

Let M=(Q,$\Sigma, \tau, q_0, \Box, F$)

Q=$\{q_0, q_1, q_2, q_3, q_4\}$, F=$\{q_4\}$

$\delta(q_0, 1) = (q_0, 1, R)$, $\delta(q_0, 0) = (q_1, 1, R)$, $\delta(q_1, 1) = (q_1, 1, R)$,

$\delta(q_1, \Box) = (q_2, \Box, L)$, $\delta(q_2, 1) = (q_3, 0, L)$, $\delta(q_3, 1) = (q_3, 1, L)$

$\delta(q_3, \Box) = (q_4, \Box, R)$

Turing Computable

Example

Given two positive integers 'x' and 'y'. Design a Turing machine that computes x+y

Solution: Using Unary notation in which any positive integer 'x' is represented by w(x) $\in \{1\}^+$ such that |w(x)|=x.

So, the required machine is: $q_0w(x)0w(y) \vdash *q_fw(x+y)0$

Steps: Move the separating 0 to right end of w(y)

Let M=(Q,$\Sigma, \tau, q_0, \Box, F$)

Q=$\{q_0, q_1, q_2, q_3, q_4\}$, F=$\{q_4\}$

$\delta(q_0, 1) = (q_0, 1, R)$, $\delta(q_0, 0) = (q_1, 1, R)$, $\delta(q_1, 1) = (q_1, 1, R)$,

$\delta(q_1, \Box) = (q_2, \Box, L)$, $\delta(q_2, 1) = (q_3, 0, L)$, $\delta(q_3, 1) = (q_3, 1, L)$

$\delta(q_3, \Box) = (q_4, \Box, R)$

Adding 111 to 11

$q_0111011 \vdash 1q_011011 \vdash 11q_01011 \vdash 111q_0011 \vdash 1111q_111 \vdash \ldots$

Variation of TM

- Turing Machine with Stationary Head
- Storage in the State
- Multiple Track Turing Machine
- Subroutines
- Multitape Turing Machines
- Nondeterministic Turing Machines

Universal TM

A universal Turing machine is a Turing machine T_u that works as follows:

- It is assumed to receive an input string of the form $e(T)e(z)$, where T is an arbitrary TM, z is a string over the input alphabet of T , and e is an encoding function whose values are strings in $\{0,1\}^*$. The computation performed by T_u on this input string satisfies two properties:
  1. T_u accepts the string $e(T)e(z)$ if and only if T accepts z.
  2. If T accepts z and produces output y, then T_u produces output $e(y)$.

Encoding Function

We assume that there is an infinite set $\mathbb{S} = \{a_1, a_2, a_3, \dots\}$ of symbols, where $a_1 = \Delta = blank$, such that the tape alphabet of every Turing machine T is a subset of $\mathbb{S}$.
If $T = (Q, \Sigma, \tau, q_0, \delta)$ is a TM and z is a string, we define the strings $e(T)$ and $e(z)$ as follows:

- First we assign numbers to each state, tape symbol, and tape head direction of T.
- Each tape symbol, including Δ, is an element a_i of $\mathbb{S}$, and it is assigned the number $n(a_i) = i$.
- The accepting state h_a, the rejecting state h_r, and the initial state q_0 are assigned the numbers $n(h_a) = 1$, $n(h_r) = 2$, and $n(q_0) = 3$.
- The other elements $q \in Q$ are assigned distinct numbers $n(q)$, each at least 4.

Encoding Function

- We don't require the numbers to be consecutive, and the order is not important.
- The three directions R, L, and S are assigned the numbers $n(R) = 1, n(L) = 2$, and $n(S) = 3$
- For each move m of T of the form $\delta(p, a) = (q, b, D)$

$$e(m) = 1^{n(p)}01^{n(a)}01^{n(q)}01^{n(b)}01^{n(D)}0$$

- List the moves of T in any order as $m_1, \ldots, m_k$. and define $e(T)$ as:

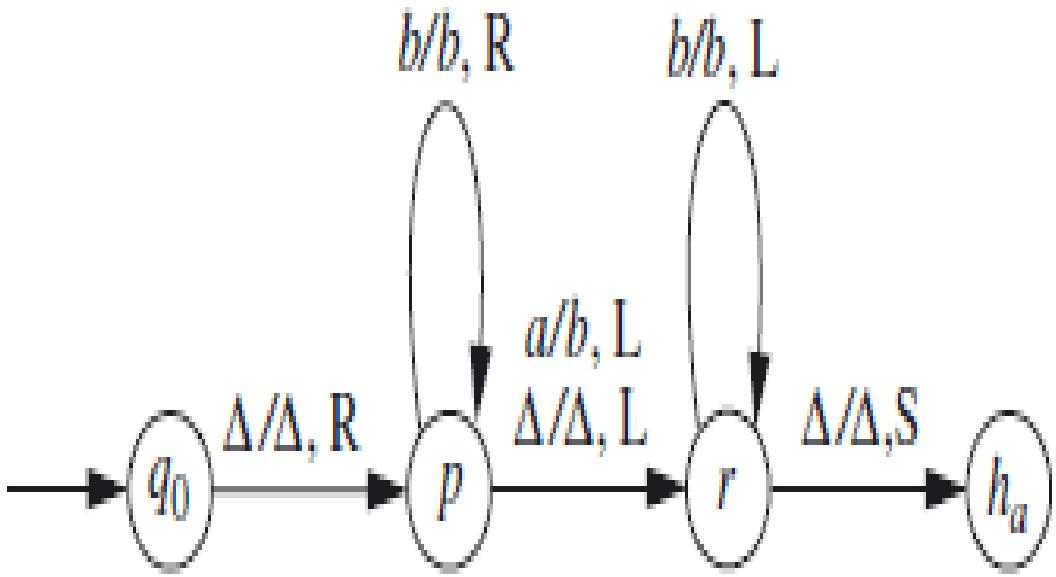
$$e(T) = e(m_1)0e(m_2)0\ldots)0(m_k)0$$

- If $z = z_1 z_2 \ldots z_j$ is a string, where each $z_i \in \mathbb{S}$

$$e(z) = 01^{n(z_1)}01^{n(z_2)}0\ldots01^{n(z_j)}0$$

Encoding Function

- The input to UTM will be $e(T)e(z)$
- Example: Let T be the TM shown in below figure, which transforms an input string of a's and b's by changing the leftmost a, if there is one, to b.



Encoding Function

- Solutions:We assume for simplicity that $n(a) = 2$ and $n(b) = 3$. By definition, $n(q_0) = 3$, and we let $n(p) = 4$ and $n(r) = 5$.
- If m is the move determined by the formula $\delta(q_0, \Delta) = (p, \Delta, R)$, then

$$e(m) = 1^3 0101^4 01010 = 111010111101010$$

- if we encode the moves in the order they appear in the diagram, left to right
  $e(T) = 1110101110101001111011101111101110100$
  $1111011011111011101100111101011111010110 0$
  $1111011011111011101100111110101011100$
- Let the string to be checked for acceptance is Δaab

$$e(z) = 0101101101110$$

Encoding Function

- Input to the UTM is $e(T)e(z)$
  $= 11101011101010011110111011110110100$
  $11110110111110110110011110101111010110 0$
  $1111101110111110111011001111101010101110001011011011 10$
- We will use three tapes. The first tape is for input and output and originally contains the string $e(T)e(z)$, where T is a TM and z is a string over the input alphabet of T.
- The second tape will correspond to the working tape of T , during the computation that simulates the computation of T on input z.
- The third tape will have only the encoded form of T's current state.
- T_u starts by transferring the string $e(z)$, except for the initial 0, from the end of tape 1 to tape 2, beginning in square 3.

Encoding Function

- Since T begins with its leftmost square blank, T_u writes 10, the encoded form of Δ, in squares 1 and 2.
- Square 0 is left blank, and the tape head begins on square 1.
- The second step is for T_u to write 111, the encoded form of the initial state q_0, on tape 3, beginning in square 1.
- $\Delta 11101011101010011110111011110111010011110110111110111011$
  $\Delta 10110110111\Delta$
  $\Delta 111\Delta$
- Now we simulate the UTM by finding the pattern for state q from tape 3 followed by code of the $0e(z_i)0$ from tape 2 under R/W head.

Encoding Function

- When pattern is found, copy 1st part as state on tape 3, replace $e(z_i)$ by 2nd part from tape 1 and move the R/W head as per the encoded value of direction in part 3 on tape 1.
- We repeat the above two steps until we found state $h_a = 1$
- In the last, when T halt with acceptance means on the 3rd tape $h_a = 1$, we erase the contents of 1st tape and copy the encoded output of UTM on 2nd tape to the 1st tape.

Church-Turing Thesis

To say that the Turing machine is a general model of computation means that any algorithmic procedure that can be carried out at all, by a human computer or a team of humans or an electronic computer, can be carried out by a TM. This statement was first formulated by Alonzo Church in the 1930s and is usually referred to as Church's thesis, or the Church-Turing thesis. It is not a mathematically precise statement that can be proved, because we do not have a precise definition of the term algorithmic procedure. By now, however, there is enough evidence for the thesis to have been generally accepted. Here is an informal summary of some of the evidence.

Church-Turing Thesis

- The nature of the model makes it seem likely that all the steps crucial to human computation can be carried out by a TM. Humans normally work with a two-dimensional sheet of paper, and a human computer may perhaps be able to transfer his attention to a location that is not immediately adjacent to the current one, but enhancements like these do not appear to change the types of computation that are possible. A TM tape could be organized so as to simulate two dimensions; one likely consequence would be that the TM would require more moves to do what a human could do in one.

Church-Turing Thesis

- Various enhancements of the TM model have been suggested in order to make the operation more like that of a human computer, or more convenient, or more efficient. The multitape TM discussed is an example. In each case, it has been shown that the computing power of the device is unchanged.
- Other theoretical models of computation have been proposed. These include abstract machines such as the ones with two stacks or with a queue, as well as machines that are more like modern computers. In addition, various notational systems (programming languages, grammars, and other formal mathematical systems) have been suggested as ways of describing or formulating computations. Again, in every case, the model has been shown to be equivalent to the Turing machine.

Church-Turing Thesis

- Since the introduction of the Turing machine, no one has suggested any type of computation that ought to be included in the category of "algorithmic procedure" and cannot be implemented on a TM.

Characteristic Function

For a language $L \subseteq \Sigma^*$, the characteristic function of L is the function $\chi_L : \Sigma^* \to \{0, 1\}$ defined by

$$\chi_L(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$$

- Computing the function χ_L and accepting the language L are two approaches to the question of whether an arbitrary string is in L or not.
- A TM computing χ_L indicates whether the input string is in L by producing output 1 or output 0.
- A TM accepting L indicates the same thing by accepting or not accepting the input.

Accepting a Language and Deciding a Language

- A Turing machine T with input alphabet Σ accepts a language $L \subseteq \Sigma^*$ if $L(T) = L$.
- T decides L if T computes the characteristic function $\chi_L : \Sigma^* \to \{0, 1\}$.
- A language L is recursively enumerable if there is a TM that accepts L, and L is recursive if there is a TM that decides L.
- Recursively enumerable languages are sometimes referred to as Turing-acceptable, and recursive languages are sometimes called Turing-decidable, or simply decidable.
- Every recursive language is recursively enumerable.

Accepting a Language and Deciding a Language

- The main difference is that in recursively enumerable language the machine halts for input strings which are in language L. but for input strings which are not in L, it may halt or may not halt. When we come to recursive language it always halt whether it is accepted by the machine or not.
- If $L \subseteq \Sigma^*$ is accepted by a TM T that halts on every input string, then L is recursive.
- If $L \subseteq \Sigma^*$ is accepted by a TM T that halts on every input string x when $x \in L$ and may or may not halt when $x \notin L$ then L is recursively enumerable.

Unrestricted Grammars

- The unrestricted grammars correspond to recursively enumerable languages in the same way that CFGs correspond to languages accepted by PDAs and regular grammars to those accepted by DFAs
- An unrestricted grammar is a 4-tuple $G = (V, \Sigma, P, S)$, where V and Σ are disjoint sets of variables and terminals, respectively. S is an element of V called the start symbol, and P is a set of productions of the form $\alpha \rightarrow \beta$ where $\alpha, \beta \in (V \cup \Sigma)^*$ and α contains at least one variable
- For every unrestricted grammar G, there is a Turing machine T with $L(T) = L(G)$.
- For every Turing machine T with input alphabet Σ, there is an unrestricted grammar G generating the language $L(T) \subseteq \Sigma^*$.

Unrestricted Grammars

- A context-sensitive grammar (CSG) is an unrestricted grammar in which no production is length-decreasing.
- In other words, every production is of the form $\alpha \rightarrow \beta$, where $|\beta| \geq |\alpha|$.
- No variable is allowed in β whose value is null.
- A language is a context-sensitive language (CSL) if it can be generated by a context-sensitive grammar.
- Example Write the production for the language $L = \{a^n b^n c^n | n \geq 1\}$

Unrestricted Grammars

- Solution: $S \to SABC|\mathbb{A}BC$,
  $BA \to AB$,
  $CA \to AC$,
  $CB \to BC$,
  $\mathbb{A} \to a$,
  $aA \to aa$,
  $aB \to ab$,
  $bB \to bb$,
  $bC \to bc$,
  $cC \to cc$.

Linear Bounded Automata

- This model is important because (a) the set of context-sensitive languages is accepted by the model and (b) the infinite storage is restricted in size but not in accessibility to the storage in comparison with the Turing machine model.
- It is called the linear bounded 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 described formally by the following set format:
  $M = (Q, \Sigma, \tau, \delta, q_0, b, \S, \$, F)$

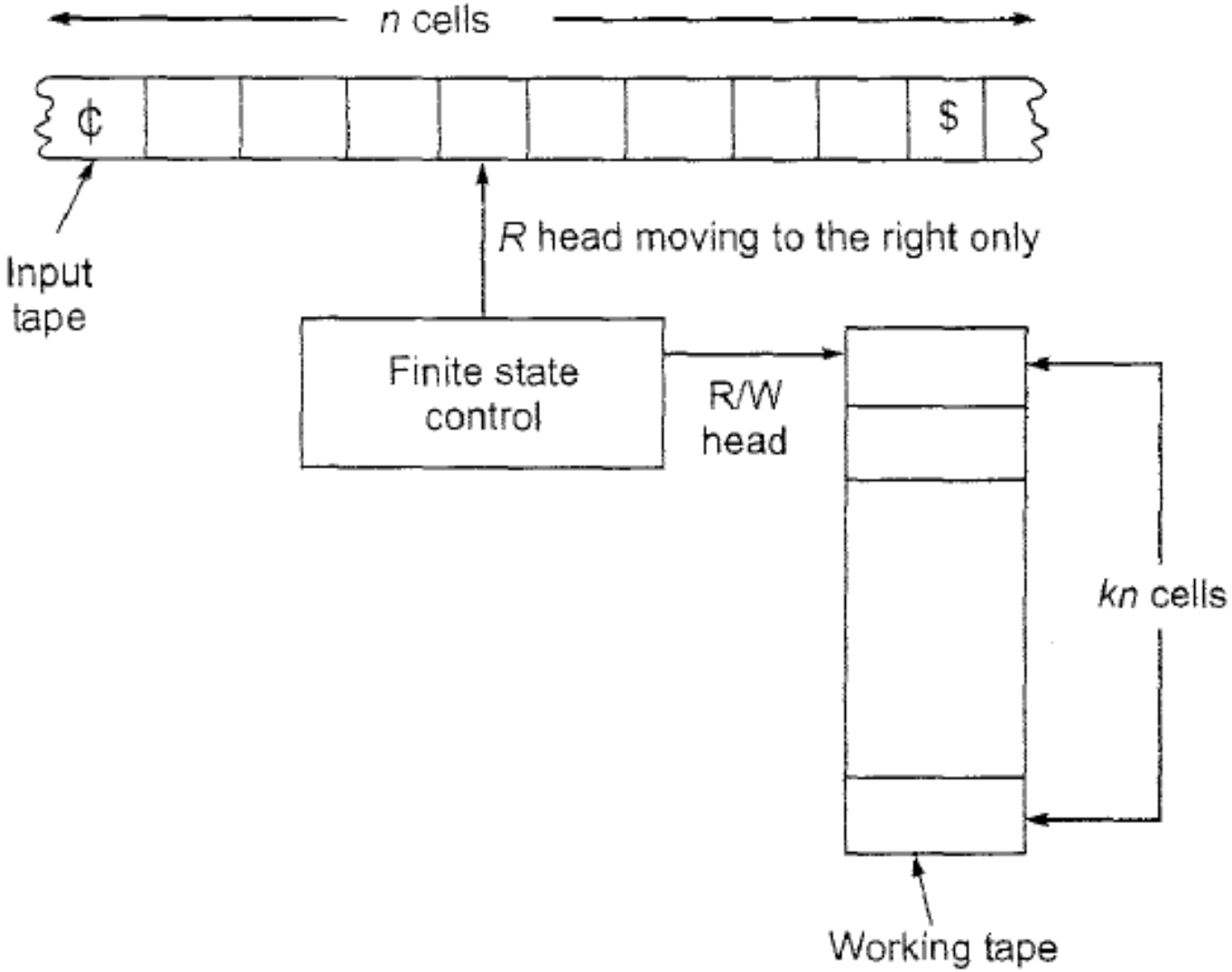
Linear Bounded Automata

- All the symbols have the same meaning as in the basic model of Turing machines with the difference that the input alphabet Σ contains two more special symbols $\S$ and \$ also.
- $\S$ is called the left-end marker which is entered in the leftmost cell of the input tape and prevents the R/W head from getting off the left end of the tape.
- \$ 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 the 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 end-markers.
- Let us consider the input string w with $|w| = n - 2$.

Linear Bounded Automata

- The input string w can be recognized by an LBA if it can also be recognized by a Turing machine using no more than kn cells of input tape, where k is a constant specified in the description of LBA.
- The value of k does not depend on the input string but is purely a property of the machine.
- Whenever we process any string in LBA, we shall assume that the input string is enclosed within the end-markers § and \$.
- The model of LBA can be represented by the block below diagram:

Linear Bounded Automata



Linear Bounded Automata

- There are two tapes: one is called the input tape, and the other is working tape.
- On the input tape the head never prints and never moves to the left.
- On the working tape the head can move in any direction Left or Right and can modify the contents in any way, without any restriction.
- In the case of LBA, an ID is denoted by (q, w, i), where $q \in Q, w \in \tau$ and i is some integer between 1 and n.
- The transition of IDs is similar except that i changes to $i - 1$ if the R/W head moves to the left and to $i + 1$ if the head moves to the right.

Linear Bounded Automata

- The language accepted by LBA is defined as the set

$$\{w \in \{\Sigma - \{\S, \$\}\}^* | (q_0, \S w \$, 1) \vdash^* (q, \alpha, i)\}$$

  for some $q \in F$ and for some integer i between 1 and n, $\alpha \in \tau^*$.

- As a null string can be represented either by the absence of input string or by a completely blank tape, an LBA may accept the null string.
- A linear bounded automaton M accepts a string w if, after starting at the initial state with R/W head reading the left-endmarker, M halts over the right-endmarker in a final state. Otherwise, w is rejected

Linear Bounded Automata

- The set of strings accepted by non-deterministic LBA is the set of strings generated by the context sensitive grammars, excluding the null strings.
- If L is a context-sensitive language, then L is accepted by a linear bounded automaton and vice versa.
- Exercise; Design the LBA for the language $L = \{a^n b^n c^n | n \geq 1\}$

Construction of Grammar Corresponding to TM

- For understanding the construction. we have to note that a transition of ID corresponds to a production.
- We enclose IDs within brackets. So acceptance of w by M corresponds to the transformation of initial ID $[q_0, \S w\$]$ into $[q_f\, b]$.
- Also, the 'length' of ID may change if the R/W head reaches the left-end or the right-end, i.e. when the left-hand side or the right-hand side bracket is reached.
- So we get productions corresponding to transition of IDs with (i) Left move (ii) Right move, and (iii) end-markers.
- Right move:

Construction of Grammar Corresponding to TM

1. $$\delta(q_i, a_j) = (q_l, a_k, R)$$

   ID $a_m q_i a_j a_{m+1} \vdash a_m a_k q_l a_{m+1}$
   leads to the production

   $$q_i a_j \rightarrow a_k q_l$$

2. When at the right-end and right move. When the R/W head moves to the right of], the length increases. That is

   $$\delta(q_i,]) = (q_i, \square, R)$$

   ID $a_m q_i] \vdash a_m q_i \square]$
   Corresponding to this we have a production

   $$q_i] \rightarrow q_i \square]$$

   for all $q_i \in Q$

Construction of Grammar Corresponding to TM

3. When $\square$ occurs to the left of], it can be deleted. This is achieved by the production

$$a_j \square] \rightarrow a_j]$$

for all $a_j \in \tau$

- Left move:

1.

$$\delta(q_i, a_j) = (q_l, a_k, L)$$

ID $a_m q_i a_j \vdash q_l a_m a_k$ leads to the production

$$a_m q_i a_j \rightarrow q_l a_m a_k$$

for all $a_m \in \tau$

2. When at the left-end and left move

$$\delta(q_i, a_j) = (q_l, a_k, L)$$

ID $[q_i a_j \vdash [q_l \square a_k$
leads to the production

$$[q_i a_j \rightarrow [q_l \square a_k$$

Construction of Grammar Corresponding to TM

3. When $\square$ occurs next to the left-bracket, it can be deleted. This is achieved by including the production

$$[\square \rightarrow [$$

- Introduction of end-markers: For introducing end-markers for the input string, the following productions are included, where q_0 is the initial state and q_f is the final state:
  1. $a_i \rightarrow [q_0 \S a_i$ for $a_i \in \tau, a_i \neq \square$
  2. $a_i \rightarrow a_i \$]$ for $a_i \in \tau, a_i \neq \square$
  3. For removing the brackets from $[q_f \square]$, we include the production

$$[q_f \square] \rightarrow S$$

- To get the required grammar, reverse the arrows of the productions obtained above.
- The productions we get can be called inverse productions.
- The new grammar is called the generative grammar.

Construction of Grammar Corresponding to LBA

- A linear bounded automaton M accepts a string w if, after starting at the 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.
- The production rules for the generative grammar are constructed as in the case of Turing machines.
- The following additional productions are needed in the case of LBA:
  1. $a_i q_f \$ \rightarrow q_f \$$ for all $a_i \in \tau$
  2. $\S q_f \$ \rightarrow \S q_f$
  3. $\S q_f \rightarrow q_f$

Exercise: Find the grammar generating the set accepted by a linear bounded automaton M whose transition table is given below:

Construction of Grammar Corresponding to LBA

Present state	Tape input symbol			
	¢	$	0	1
$\rightarrow q_1$	$¢Rq_1$		$1Lq_2$	$0Rq_2$
q_2	$¢Rq_4$		$1Rq_3$	$1Lq_1$
q_3		$\$Lq_1$	$1Rq_3$	$1Rq_3$
(q_4)		Halt	$0Lq_4$	$0Rq_4$

CYK Algorithm

- The algorithm is called the CYK algorithm, after its originators J. Cocke, D. H. Younger, and T. Kasami.
- The algorithm works only if the grammar is in Chomsky normal form and succeeds by breaking one problem into a sequence of smaller ones in the following way.
- Assume that we have a grammar G = (V, T, S, P) in Chomsky normal form and a string $w = a_1 a_2 \ldots a_n$.
- Define substrings $w_{ij} = a_i \ldots a_j$
- Define subsets of V as $V_{ij} = \{A \in V : A \Rightarrow^* w_{ij}\}$
- Clearly, $w \in L(G)$ if and only if $S \in V_{1n}$.
- To compute V_{ii}, observe that $A \in V_{ii}$ if and only if G contains a production $A \rightarrow a_i$.
- Therefore, V_{ii} can be computed for all $1 \leq i \leq n$ by inspection of w and the productions of the grammar.

CYK Algorithm

- To compute V_{ij} for $i < j$, A derives w_{ij} if and only if there is a production $A \rightarrow BC$, with $B \Rightarrow^* w_{ik}$ and $C \Rightarrow^* w_{k+1j}$ for some k with $i \leq k < j$.
- In other words,
  $V_{ij} = \cup_{k=i\ldots j-1}\{A : A \rightarrow BC, B \in V_{ik}, C \in V_{k+1j}$
  $\Rightarrow \{A_1 | A_1 \rightarrow \{V_{ii}\}\{V_{i+1j}\}\} \cup \{A_2 | A_2 \rightarrow \{V_{ii+1}\}\{V_{i+2j}\}\} \cup$
  $\cdots \cup \{A_l | A_l \rightarrow \{V_{ik}\}\{V_{k+1j}\}\} \cup \cdots \cup \{A_{n-1} | A_{n-1} \rightarrow$
  $\{V_{ij-2}\}\{V_{j-1j}\}\} \cup \{A_n | A_n \rightarrow \{V_{ij-1}\}\{V_{jj}\}\}$
- Compute all the V_{ij} using the above eq. as:
  1. Compute $V_{11}, V_{22}, \ldots, V_{nn}$
  2. Compute $V_{12}, V_{23}, \ldots, V_{n-1n}$
  3. Compute $V_{13}, V_{24}, \ldots, V_{n-2n}$
  4. $\ldots$
  5. Compute V_{1n}
- If $S \in V_{1n}$ then $w \in L(G)$ otherwise $w \notin L(G)$

CYK Algorithm

Exercise: Determine whether the string $w = aabbb$ is in the language generated by the grammar:

$$S \to AB$$

$$A \to BB|a$$

$$B \to AB|b$$

Questions?

+91 9911375598, akyadav1@amity.edu

Thank you.