FORMAL LANGUAGES AND AUTOMATA THEORY

Module 5

Rijin IK

Assistant Professor
Department of Computer Science and Engineering
Vimal Jyothi Engineering College
Chemperi

November 15, 2023

Outline

Course Outcomes

After the completion of the course the student will be able to

1. Classify a given formal language into Regular, Context-Free, Context Sensitive, Recursive or Recursively Enumerable. [Cognitive knowledge level: Understand]
2. Explain a formal representation of a given regular language as a finite state automaton, regular grammar, regular expression and Myhill-Nerode relation. [Cognitive knowledge level: Understand]
3. Design a Pushdown Automaton and a Context-Free Grammar for a given context-free language. [Cognitive knowledge level : Apply]
4. Design Turing machines as language acceptors or transducers. [Cognitive knowledge level: Apply]
5. Explain the notion of decidability. [Cognitive knowledge level: Understand]

Context Sensitive Grammar

Context Sensitive Languages

- A context sensitive grammar (CSG) is an unrestricted grammar in which all the productions are of the form

$$\alpha \rightarrow \beta$$

  Where $\alpha, \beta \in (V \cup T)^+$ *and* $|\alpha| \leq |\beta|$.
  - Where α, β are strings of terminals and non- terminals.
- A context sensitive grammar can never generate a language containing the empty string ϵ; $x \rightarrow \epsilon$ not allowed.
- A context sensitive grammar (CSG) can be converted into normal form where all productions are of the form,

$$\alpha A \beta \rightarrow \alpha \gamma \beta \text{ where } \gamma \neq \epsilon$$

- During derivation non-terminal A will be replaced by γ only when it is present in context of α and β
- This definition shows clearly one aspect of this type of grammar; it is **noncontracting**, in the sense that the length of successive sentential forms can never decrease

Context Sensitive Languages

Formal definition of Context Sensitive Grammar

- A context sensitive grammar $G = (V, T, P, S)$, where
  - V is a set of nonterminal symbols
  - T is a set of terminal symbols
  - S is the start symbol, and
  - P is a set of production rules, of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$ where A in V, α, $\beta \in (V \cup T)^*$and $\gamma \in (V \cup T)^+$

*The production $S \rightarrow \epsilon$ is also allowed if S is the start symbol and it does not appear on the right side of any production.

Context Sensitive Languages

Context Sensitive Language

- A language L is said to be context-sensitive if there exists a context-sensitive grammar G, such that $L = L(G)$.
- If G is a Context Sensitive Grammar then,

$$L(G) = \{w | (w \in \Sigma^*) \wedge (S \overset{+}{\underset{G}{\Longrightarrow}} w)\}$$

Context Sensitive Languages

Example. The following grammar(G) is context-sensitive

$$\begin{aligned} S &\rightarrow aTb|ab \\ aT &\rightarrow aaTb|ac \end{aligned}$$

$$L(G) = \{ab\} \cup \{a^n cb^n | n > 0\}$$

- This language is also a context-free.
- For example, Context free grammar($G1$) for this.

$$\begin{aligned} S &\rightarrow aTb|ab \\ T &\rightarrow aTb|c \end{aligned}$$

- Any context-free language is context sensitive
- Not all context-sensitive languages are context-free.

Context Sensitive Languages

Closure properties

- Context Sensitive Languages are closed under
  1. Union
  2. Intersection
  3. Complement
  4. Concatenation
  5. Kleene closure
  6. Reversal

Linear Bounded Automata (LBA)

Linear Bounded Automata is a single tape Turing Machine with two special tape symbols call them left marker $\vdash$ and right marker $\dashv$.

- It is allowed to read/write between these markers.
  - It should not replace the marker symbols by any other symbol.
  - It should not write on cells beyond the marker symbols.
- Context sensitive languages are recognised using linear bounded automata (LBA)
- Here input tape is restricted in size. A linear function is used for restricting the length of the input tape.

Linear Bounded Automata (LBA)

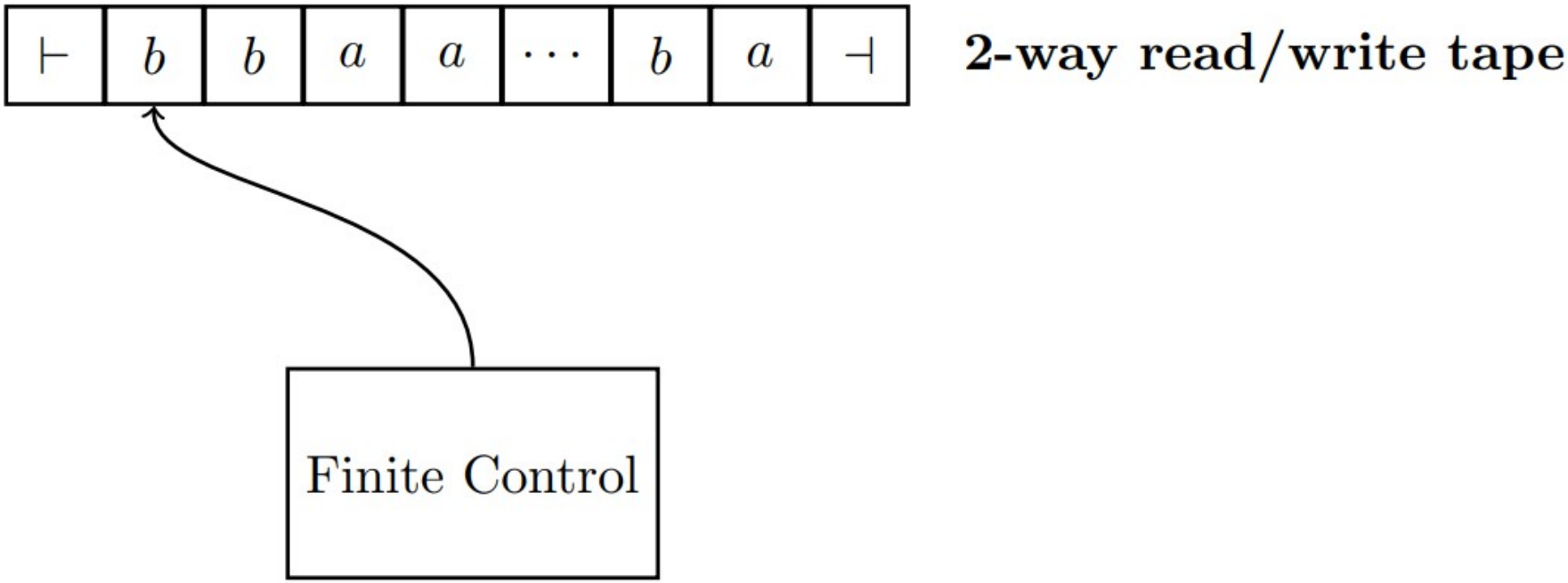


Figure: Linear Bounded Automata (LBA)

Linear Bounded Automata (LBA)

Read/ write tape:

- Read/ write head can move to right/left, one cell at a time.
- The tape is initally assumed to be store the input delimited by $\dashv$ and the blank symbol B, which is followed by blank symbol ($\dashv$)in all the cells until the cell containing $\vdash$
- If the input size is n, the number of cells in the tape which the machine can use is bounded by a linear function of n.
- The machine can scan the input in right/left directions any number of rounds

Finite Control:

- Control unit with a finite set of states which can remember finite amount of information about the processed input.

Linear Bounded Automata (LBA)

Formal definition

- Formally Linear Bounded Automata is a non-deterministic Turing Machine, $M = (Q, \Sigma, \Gamma, \delta, B, \vdash, \dashv, q_0, t, r)$
  - Q is set of all states
  - Σ is set of all terminals
  - Γ is set of all tape alphabets $\Sigma \subset \Gamma$
  - δ is set of transitions $(Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{R, L\})$
  - B is blank symbol $(B \in \Gamma)$
  - $\vdash$ is left marker and $\dashv$ is right marker ($\vdash$ *and* $\dashv \in \Gamma$)
  - q_0 is the initial state $(q_0 \in Q)$
  - t is accept state $(t \in Q)$
  - r is reject state $(r \in Q)$

Linear Bounded Automata (LBA)

Example: Consider an LBA defined as,

$$
\begin{aligned}
Q &= \{q0, q1, q2, q3, q4\} \\
\Sigma &= \{a, b, c\} \\
\Gamma &= \{a, b, c, B\} \\
q0 &= q0
\end{aligned}
$$

$$
\begin{array}{lll}
\delta(q0, \vdash) = (q1, \vdash, R) & \delta(q1, \vdash) = (q1, \vdash, R) & \delta(q1, B) = (q1, B, R) \\
\delta(q1, a) = (q2, B, R) & \delta(q2, a) = (q2, a, R) & \delta(q2, B) = (q2, B, R) \\
\delta(q2, b) = (q3, B, R) & \delta(q3, b) = (q3, b, R) & \delta(q3, B) = (q3, B, R) \\
\delta(q3, c) = (q4, B, L) & \delta(q4, c) = (q4, c, L) & \delta(q4, b) = (q4, b, L) \\
\delta(q4, a) = (q4, a, L) & \delta(q4, B) = (q4, B, L) & \delta(q4, \dashv) = (q1, \vdash, R)
\end{array}
$$

In the above, the transition $\delta(q3, b) = (q3, b, R)$ means,LBA on state q3, head points to symbol b, remains in state q3, replaces b with b and head turns towards right by one cell

Linear Bounded Automata (LBA)

Example:1A.Design Linear Bounded Automata for the language $L = \{a^n b^n c^n | n \geq 1\}$.

$$
\begin{aligned}
Q &= \{q0, q1, q2, q3, t, r\} \\
\Sigma &= \{a, b, c\} \\
\Gamma &= \{a, b, c, X, Y, Z\} \\
q0 &= q0 \\
t = t(\textit{accept state}) &\quad , \quad r = r(\textit{reject state})
\end{aligned}
$$

$$
\begin{array}{lll}
\delta(q_0, \vdash) = (q_0, \vdash, R) & \delta(q_0, a) = (q_1, X, R) & \delta(q_1, a) = (q_1, a, R) \\
\delta(q_1, b) = (q_2, Y, R) & \delta(q_2, b) = (q_2, b, R) & \delta(q_2, c) = (q_3, Z, L) \\
\delta(q_3, b) = (q_3, b, L) & \delta(q_3, Y) = (q|_3, Y, L) & \delta(q_3, a) = (q_3, a, L) \\
\delta(q_3, X) = (q_0, X, R) & \delta(q_2, Z) = (q_2, Z, R) & \delta(q_3, Z) = (q_3, Z, L) \\
\delta(q_1, Y) = (q_1, Y, R) & \delta(q_0, Y) = (q_0, Y, R) & \delta(q_0, Z) = (q_0, Z, R) \\
\delta(q_0, B) = (t, B, L) & \delta(q_0, b) = (r, b, R) & \delta(q_0, c) = (r, c, R) \\
\delta(q_1, c) = (r, c, R) & \delta(q_2, B) = (r, B, L) &
\end{array}
$$

Turing Machine (TM)

Class	Grammars	Languages	Automaton
Type-0	Unrestricted	Recursive Enumerable	Turing Machine
Type-1	Context Sensitive	Context Sensitive	Linear-Bound
Type-2	Context Free	Context Free	Pushdown
Type-3	Regular	Regular	Finite

Figure: Chomsky Hierarchy

From the diagram, unrestricted grammars produce unrestricted languages (Type-0 languages). These Type-0 languages are recognised using Turing machines

Turing Machine (TM)

Example of unrestricted grammars:

$$\begin{aligned} S &\rightarrow ACaB \\ CaBc &\rightarrow aaC \\ CB &\rightarrow DB \\ aD &\rightarrow Da \\ aEC &\rightarrow Ea \\ AE &\rightarrow \epsilon \end{aligned}$$

Here LHS and RHS can contain any set of terminals and non terminals.

Turing Machine (TM)

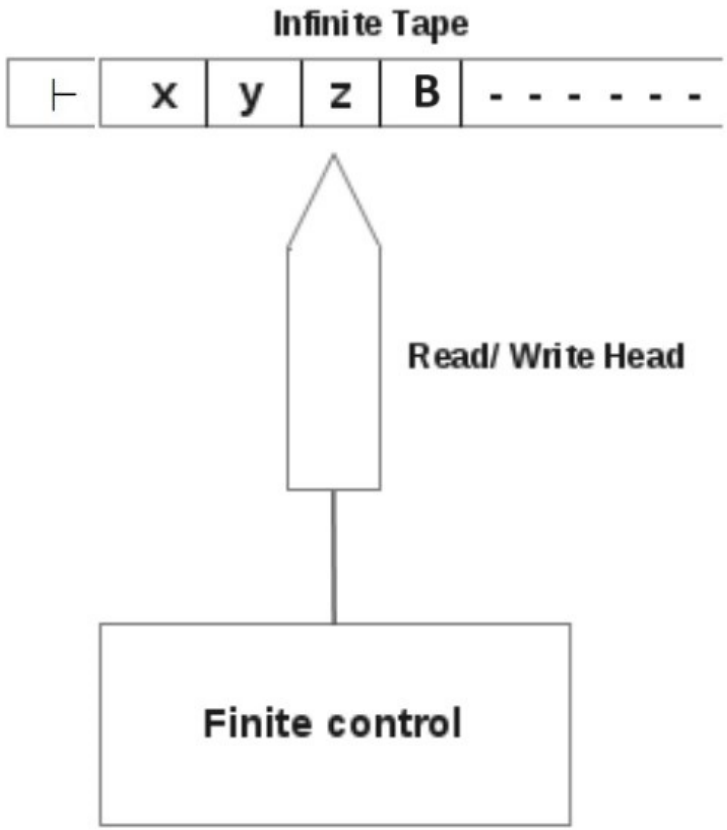


Figure: Turing Machine (TM)

The basic model has a finite control, an input tape that is divided into cells, and a tape head that scans one cell of the tape at a time.

Turing Machine (TM)

- Tape
  - The tape has a leftmost cell but is infinite to the right.
  - Each cell of the tape may hold exactly one of a finite number of tape symbols.
  - Initially, the n leftmost cells, for some finite $n \geq 0$, hold the input, which is a string of symbols chosen from a subset of the tape symbols called the input symbols.
  - The remaining infinity of cells each hold the blank, which is a special tape symbol that is not an input symbol.
- Finite control
  - At a particular instant, finite control is in a state. States are divided into,
    - Start state ($q0$),
    - Halt state (h),
    - Intermediate states ($q1, q2, ...$).
- Tape head
  - Tape head points to one of the tape cells. It communicates between finite control and tape. It can move left , right or remain stationary.

Turing Machine (TM)

Moves of Turing Machine

In one move the Turing machine, depending upon the symbol scanned by the tape head and the state of the finite control,

1. changes state
2. prints a symbol on the tape cell scanned, replacing what was written there, and
3. moves its head left or right one cell.

The difference between a Turing machine and a two-way finite automaton lies in the former's ability to change symbols on its tape.

Turing Machine (TM)

Formal definition-Turing Machine (TM)

Turing Machine M is denoted by the 9-tuple:

$M = (Q, \Sigma, \Gamma, \delta, B, \vdash, q_0, t, r)$

- Q is set of all states
- Σ is a finite set (the input alphabet)
- Γ is set of all tape alphabets $\Sigma \subset \Gamma$
- δ is set of transitions $(Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{R, L\})$
- B is blank symbol $(B \in \Gamma)$
- $\vdash$ is left marker $(\vdash \in \Gamma)$
- q_0 is the initial state $(q_0 \in Q)$
- t is accept state $(t \in Q)$
- r is reject state $(r \in Q)$

Intuitively, $\delta(p, a) = (q, b, d)$ means, "When in state p scanning symbol a, write b on that tape cell, move the head in direction d(L/R), and enter state q. The symbols Land R stand for left and right, respectively.

Turing Machine (TM)

Instantaneous description (ID):

- Instantaneous description of the Turing machine M is denoted by $\alpha_1 q \alpha_2$
- Here q, the current state of M, is in Q; $\alpha_1\alpha_2$ is the string in Γ^* that is the contents of the tape up to the rightmost nonblank symbol or the symbol to the left of the head, whichever is rightmost. (Observe that the blank B may occur in $\alpha_1\alpha_2$.)
- The tape head is assumed to be scanning the leftmost symbol of α_2, or if $\alpha_2 = \epsilon$, the head is scanning a blank.

Turing Machine (TM)

Define a move in TM

- Let $X_1X_2\ldots X_{i-1}qX_i\ldots X_n$ be an ID.
- The left move is: if $\delta(q, X_i) = (p, Y, L)$,if $i > 1$ then
  - $X_1X_2\ldots X_{i-1}qX_i\ldots X_n \vdash X_1X_2\ldots X_{i-2}pX_{i-1}YX_{i+1}\ldots X_n$.
- The right move is: if $\delta(q, X_i) = (p, Y, R)$,if $i > 1$ then
  - $X_1X_2\ldots X_{i-1}qX_i\ldots X_n \vdash X_1X_2\ldots X_{i-1}YpX_{i+1}\ldots X_n$.

Turing Machine (TM)

Language accepted by a turing machine
The language of a Turing machine M, denoted L(M), is the set of all strings that M accepts:

$$L(M) = \{w \in \Sigma^* | M \text{ accepts } w\}$$

TM either accepts or rejects a string. That is, **TM acts as a language acceptor.**

- A TM is said to accept an input string w if :
  $(q_0, \vdash wB^W, 0) \stackrel{*}{\Rightarrow} (t, y, n), y \in \Gamma^w, n \in N$
- A TM is said to reject an input string w if :
  $(q_0, \vdash wB^W, 0) \stackrel{*}{\Rightarrow} (r, y, n), y \in \Gamma^w, n \in N$

Turing Machines as Language Accepters

- Turing machines can be viewed as accepters in the following sense.
  - String w is written on the tape, with blanks filling out the unused portions.
  - The machine is started in the initial state q_0 with the read-write head positioned on the leftmost symbol of w.
  - If, after a sequence of moves, the Turing machine enters a final state and halts, then w is considered to be accepted.

Definition

- Let $M = (Q, \Sigma, \Gamma, \delta, B, \vdash, q_0, t, r)$ be a Turing machine. Then the language accepted by M is

$$L(M) = \{w \in \Sigma^+ : q_0 w \vdash^* x_1 q_f x_2 \text{ for some } q_f \in t, x_1, x_2 \in \Gamma^*\}.$$

Turing Machine (TM)

Q:Design a TM to accept the language $L = \{0^n 1^n | n \geq 1\}$.

- Initially, the tape of M contains $0^n 1^n$ followed by infinity of blanks.
- Repeatedly, M replaces the leftmost 0 by X, moves right to the leftmost 1, replacing it by Y, moves left to find the rightmost X, then moves one cell right to the leftmost 0 and repeats the cycle.
- If, however, when searching for a 1, M finds a blank instead, , then M halts without accepting.
- If, after changing a 1 to a Y, M finds no more 0's, then M checks that no more 1's remain, accepting if there are none

Turing Machine (TM)

State	Symbol 0	1	X	Y	B
q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, B, R)
q_4	—	—	—	—	—

Figure: Transition table

Turing Machine (TM)

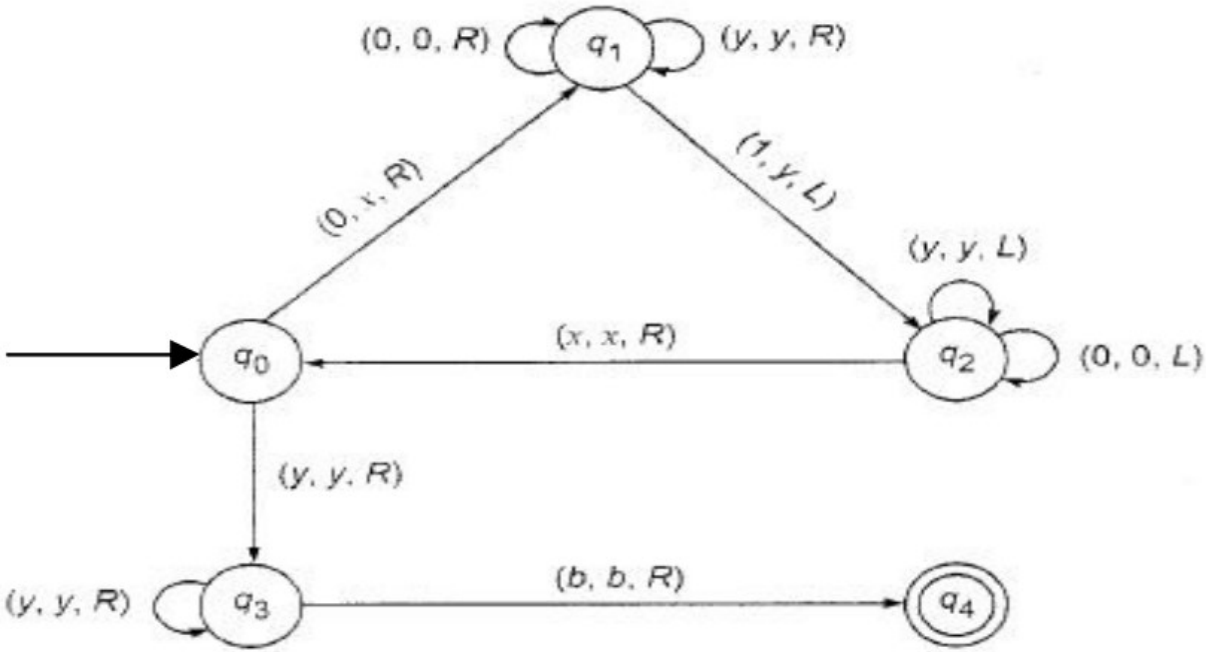


Figure: Transition diagram

Turing Machine (TM)

String Verfication by Turing Machine

$$
\begin{aligned}
q_0 0011 &\vdash X q_1 011 \vdash X0 q_1 11 \vdash X q_2 0Y1 \vdash \\
q_2 X0Y1 &\vdash X q_0 0Y1 \vdash XX q_1 Y1 \vdash XXY q_1 1 \vdash \\
XX q_2 YY &\vdash X q_2 XYY \vdash XX q_0 YY \vdash XXY q_3 Y \vdash \\
XXYY q_3 &\vdash XXYYB q_4
\end{aligned}
$$

Figure: A computation of M

Turing Machine (TM)

TM as transducers

A TM can function as a transducer. That is, a string is given as input to TM, it produces some output.

- Input to the computation is a set of symbols on the tape.
- At the end of computation, whatever remains on the tape is the output.

A TM can be viewed as a transducer for the implementation of function f defined as,$y = f(x)$ for strings $x, y \in \Sigma^*$

- A function, f is said to be computable or Turing computable, if there exists a TM M such that $q_0 x \vdash^* q_f \ f(x)$
- where x is in the domain of f.

Turing Machine (TM)

TM is an abstract model of our modern computer system

- We can see that all common mathematical functions are turing computable.
- This means, basic operations such as addition, subtraction, multiplication, division can be performed on it.
- This means that a TM is an abstract model of our modern computer system.

Turing Machine (TM)

Unary Representation

- In computing functions using a Turing machine (TM) where inputs are represented using unary notation, a non-negative integer n is indeed represented as a sequence of 1s or 0s.
- For example:
  - If n = 2, it is represented as "11" or "00"
  - If n = 4, it is represented as "1111" or "0000"

Turing Machine (TM)

Q:Design a Turing machine that computes the following function,

$$f(m, n) = m + n.$$

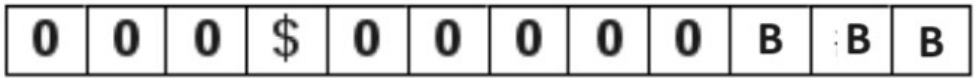
A positive integer on a Turing tape can be represented by an equal number of 0s.For example,

- integer 5 is represented as 00000, and
- integer 8 is represented as 00000000.

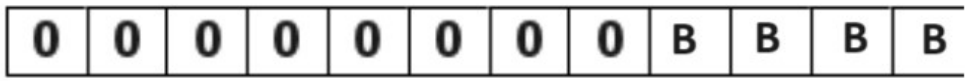
In the tape two integers are separated using the symbol, \$.

Turing Machine (TM)

Let us assume that m=3 and n=5.Then the input tape of the Turing machine will be,



After addition, tape will contain the results of addition as shown below



TM for addition works as follows:

- Head reads the 0s of the first number, m and reaches the separator symbol, $. Symbol, $ is replaced with 0 and head moves towards right
- It continues to move towards right till the second number, n is passed over and B is reached.
  - At B, it turns left and replaces rightmost 0 with B.
  - Now the tape contains the sum of m and n, and TM halts.

Turing Machine (TM)

Conversion of the separator symbol, \$ to 0, helps TM to make the single number on the tape. The conversion of symbol, 0 to B removes the additional 0 that was generated from the conversion of \$ to 0.
Transition table for the TM is shown below:

	Input Symbol		
Current State	0	\$	B
$\longrightarrow q_0$	$(q_0, 0, R)$	$(q_1, 0, R)$	—
q_1	$(q_1, 0, R)$	—	(q_2, B, L)
q_2	(q_f, B, L)	—	—
$*q_f$	—	—	—

Figure: Transition table

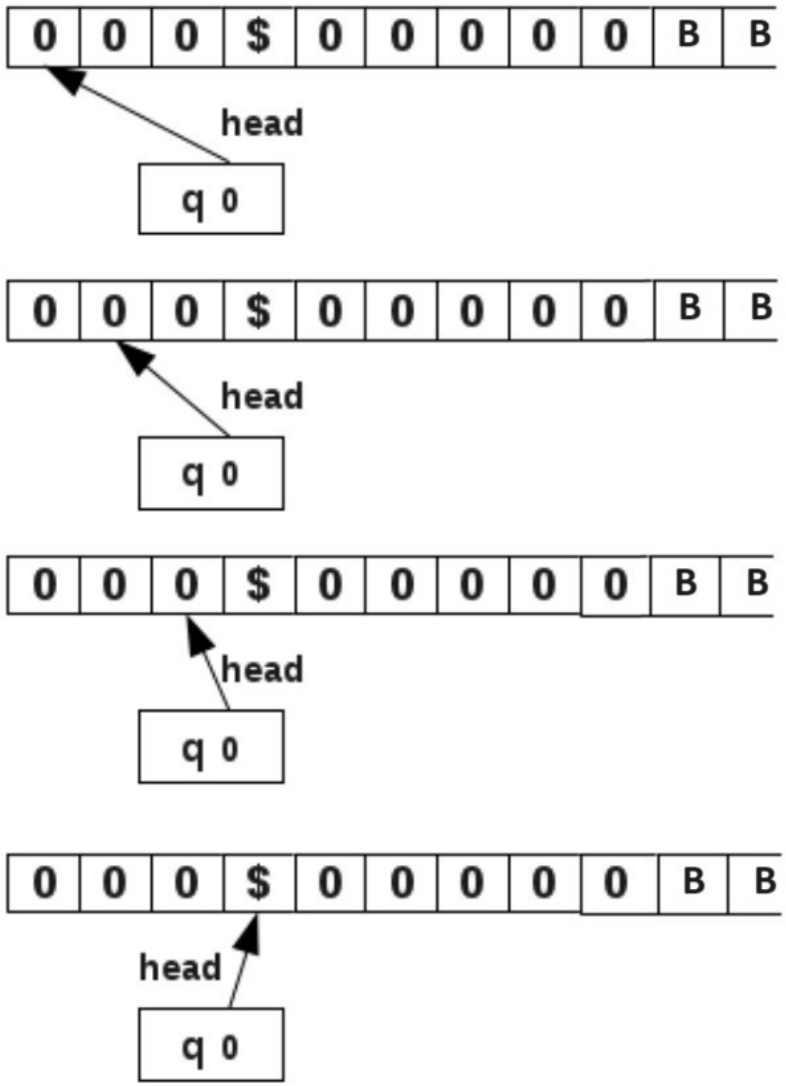
Turing Machine (TM)

The Transition table is describes as follows:

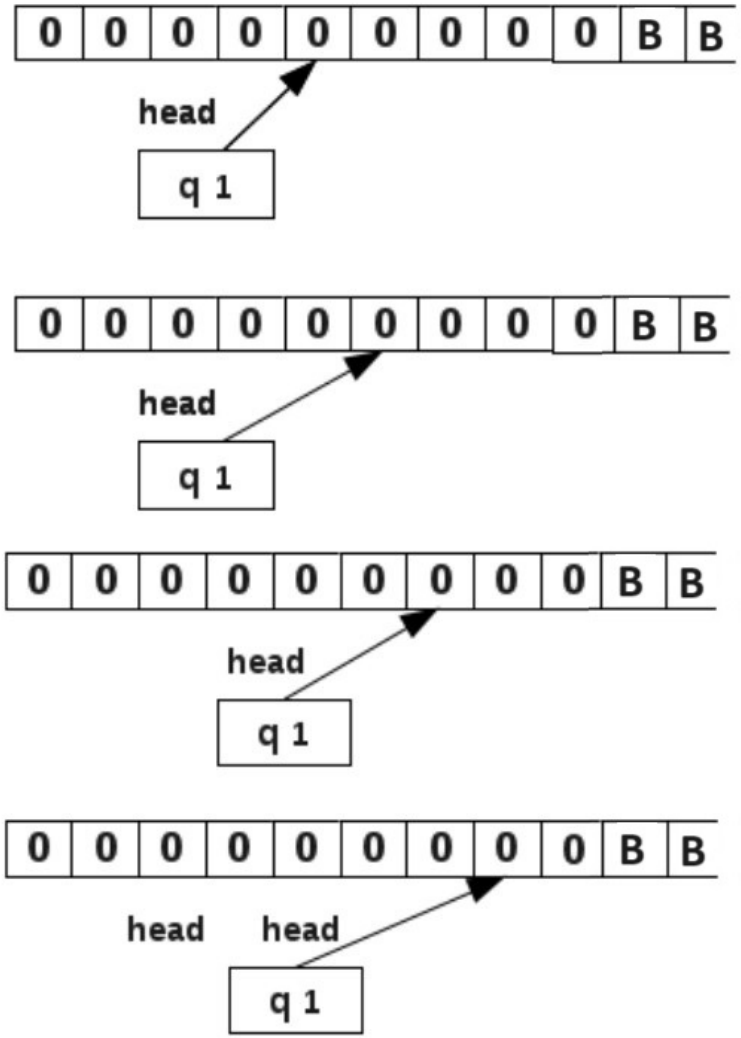
1. Initially, TM is in state q_0. Head reads the first symbol, 0 and moves towards right without changing the state.
2. Head continues to move towards right till the separator symbol,$ is reached. It replaces $ with 0, changes state to q1, and continues to move towards right.
3. Head continues to move towards right and passes over the second number to reach B.
4. On reaching the B, head changes state to q_2 and turns left.
5. In state q_2, it replaces symbol 0 with B, and changes state to q_f .
6. In the final state, TM halts to indicate the completion of the process.

Turing Machine (TM)

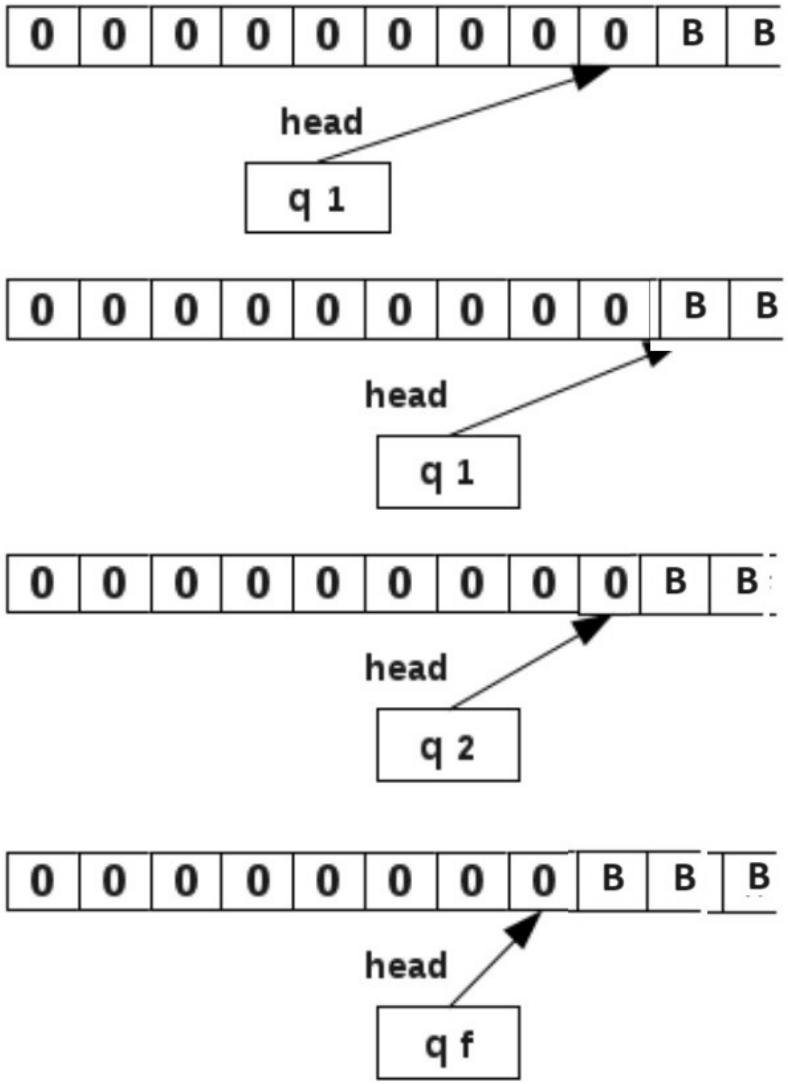
Addition of 3 and 5 is shown below



Turing Machine (TM)



Turing Machine (TM)



Robustness of the standard TM model

Robustness of the standard TM model

- A computational model is robust if the class of languages it accepts does not change under variants.
- The robustness of Turing Machines is by far greater than the robustness of DFAs and PDAs.
- We introduce several variants on Turing machines and show that all these variants have equal computational power.

Robustness of the standard TM model

Types of Turing Machines

- Two-way infinite tape
- Multitape Turing machines
- Multi-track Turing Machine
- Nondeterministic Turing machines
- Multidimensional Turing machines
- Off-line Turing machines
- Universal Turing Machines

Two-way infinite tape

Two-way infinite tape

- The TM we discussed so far had a tape that was finite on the left end and infinite on the right end.
- Two way infinite TM is an extension to this such that tape is infinite at both ends. This is shown below:

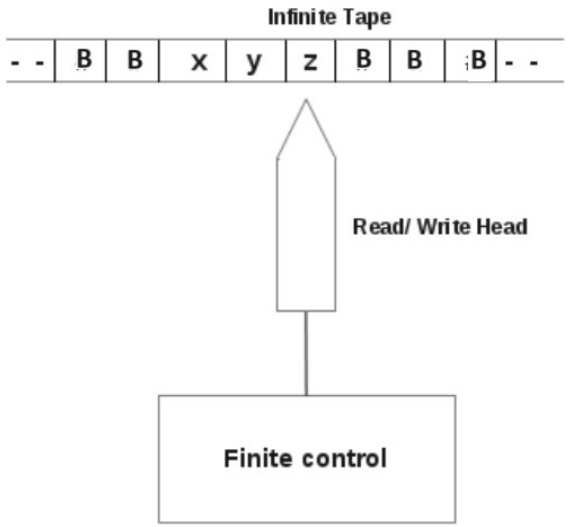


Figure: Two-way infinite tape

- Thus on both sides of the tape, there is an infinite sequence of blank symbols(B).
- This model does not provide any additional computational capability.

Multitape Turing Machines

- A multitape TM consists of multiple tapes. Each tape has a separate head

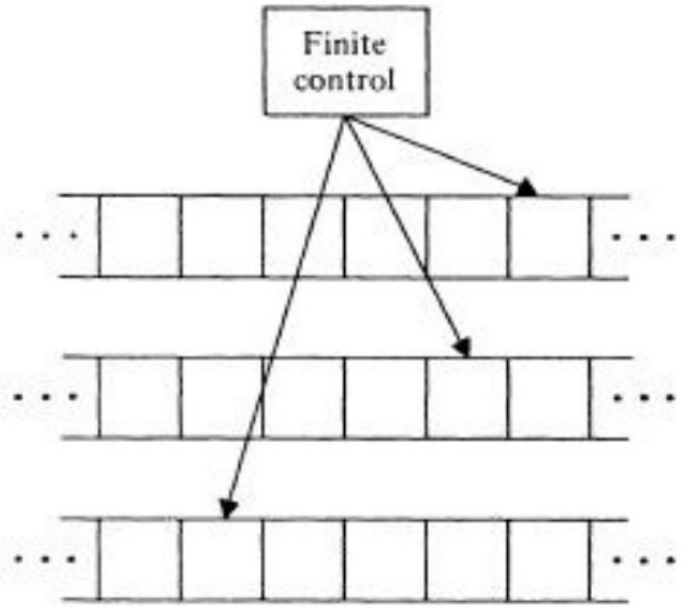


Figure: Multitape Turing Machines

Multitape Turing Machines

- There are n tapes, each divided into cells. The first tape holds the input string. Initially, all the other tapes hold the blank symbol, B
- A head has three possible options:
  - to move towards left (L),
  - to move towards right (R), or
  - to remain stationary (N).
- All the heads are connected to a finite control. Finite control is in a state at an instant.
  1. Initially, finite control is in state q_0.
  2. Head in the lowermost tape points to the cell containing leftmost symbol of the input string.
  3. All the cells in the upper tapes contain B symbol

Multitape Turing Machines

- When a transition occurs,
  1. Finite control may change its state.
  2. Head reads the symbol from the current cell and writes a symbol on it.
  3. Each head can move towards left (L), right (R) or stay stationary (N)

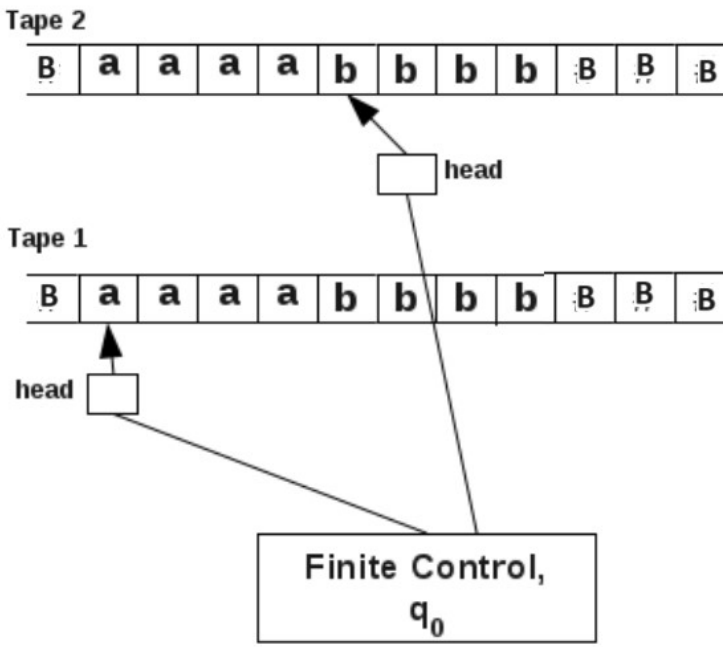
An example transition function for a 4 tape TM is given below:

- $\delta(q_1, [a_1, a_2, a_3, a_4]) = (q_2, [b_1, b_2, b_3, b_4], [L, R, R, N])$
- Here the finite control is in state q_1, It reads the symbol a_1 from the uppermost tape, a_2 from the next uppermost tape and so on.
- After reading, finite control changes its state to q_2, and replaces the symbol a_1 by b_1, a_2 by b_2, a_3 by b_3 , a_4 by b_4.
- After this head in the upper most tape turns left. Head in the 2nd tape turns right. Head in the 3rd tape turns right. Headin the 4th tape remains stationary.

Multitape Turing Machines

Example: Consider the language $L = \{a^n b^n \big| n \geq 1\}$.

- In a normal TM, head has to move back and forth to match each pair of symbols a and b. On a multitape TM, no such movements are needed.
- This is done by making a copy of the input string in another tape.
- Let the input string be aaaabbbb.
- Let the input srtring be in tape 1. Make a copy of this string in tape 2



Multitape Turing Machines

- The head of tape 1 is positioned on the first a of the input string. Head of tape 2 is positioned on the first b of the input string.
- Now the heads advance on both tapes simultaneously towards right and the string is accepted if there are equal number of a's and b's in the string.
- This will happen if the head on tape 1 encounters the first b and the head on tape 2 encounters the first B(blank symbol) simultaneously.

*Every multitape Turing Machines has an equivalent single tape Turing machine.

Multi-track Turing Machine

- Multi-track Turing machines, a specific type of Multi-tape Turing machine, contain multiple tracks but just one tape head reads and writes on all tracks.
- Here, a single tape head reads n symbols from n tracks at one step.
- It accepts recursively enumerable languages like a normal single-track single-tape Turing Machine accepts.

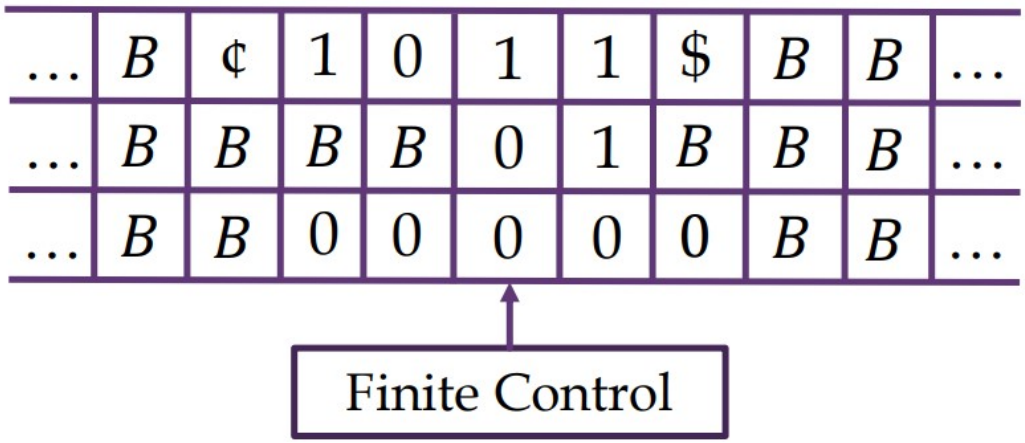


Figure: Multi-track Turing Machines

Multitape Turing Machines

Theorem:
Every language accepted by a k-tape TM M1 is also accepted by a single tape TM M2

Proof by construction

- Let L be accepted by M1 ,a TM with k-tapes. We can construct M2, a one-tape TM with 2k tracks, two tracks for each of M1's tapes.
- One track records the contents of the corresponding tape of M1
- and the other is blank, except for a marker in the cell that holds the symbol scanned by the corresponding head of M1.
- The finite control of M2 stores the state of M1 ,along with a count of the number of head markers to the right of M2's tape head.H

Multitape Turing Machines

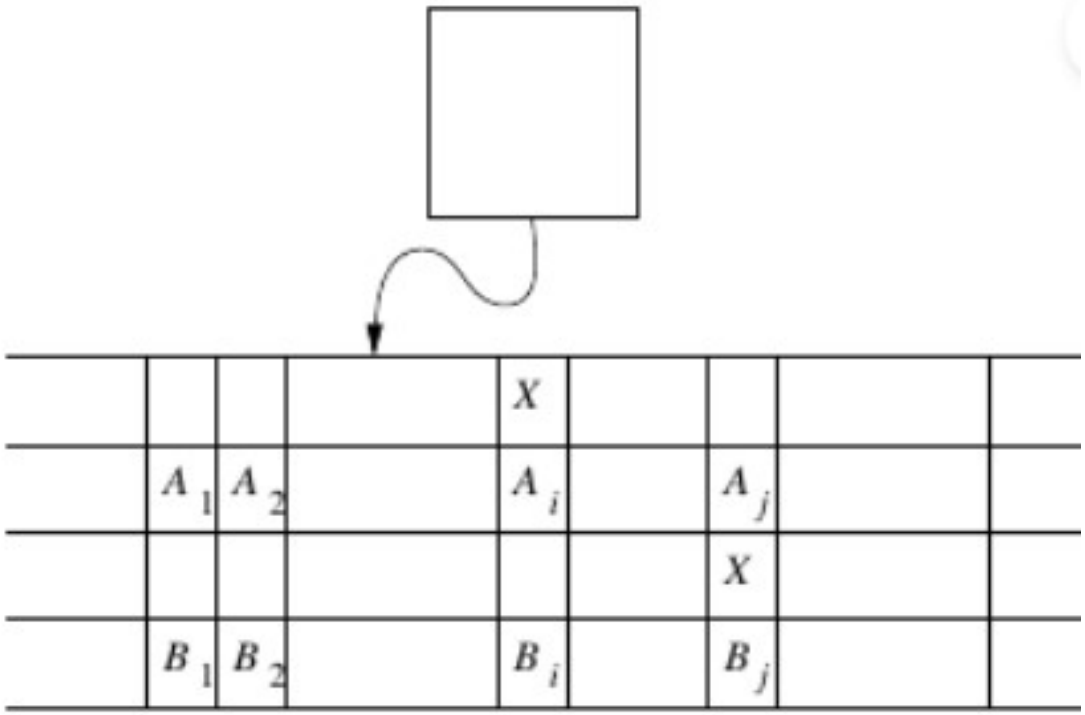


Figure: Simulation of multi tapes by one

Multitape Turing Machines

To simulate one move of the k-tape TM:

- Each move of M1 is simulated by a sweep from left to right and then from right to left by the tape head of M2.
- Initially, M2's head is at the leftmost cell containing a head marker.
- To simulate a move of M1, M2 sweeps right, visiting each of the celis with head markers andrecording the symbol scanned by each head of M1.
- When M2 crosses a head marker, it mustupdate the count of head markers to its right.
- When no more head markers are to the right, M2 has seen the symbols scanned by each of M1's heads, so M2 has enough information todetermine the move of M1.
- Now M2 makes a pass left, until it reaches the leftmost headmarker.

Multitape Turing Machines

- The count of markers to the right enables M2 to tell when it has gone for enough.
- As M2 passes each head marker on the leftward pass, it updates the tape symbol of M1 "scanned" by that head marker, and it moves the head marker one symbol left or right to simulate themove of M1.
- Finally, M2 changes the state of M1 recorded in M2's control to complete thesimulation of one move of M1.
- If the new state of M1 is accepting, then M2 accepts.

Nondeterministic Turing machines

- A nondeterministic Turing machine is a device with a finite control and a single, one-way infinite tape.
- For a given state and tape symbol scanned by the tape head, the machine has a finite number of choices for the next move.
- Each choice consists of a new state, a tape symbol to print, and a direction of head motion.
- Note that the nondeterministic TM is not permitted to make a move in which the next state is selected from one choice, and the symbol printed and/or direction of head motion are selected from other choices.
- The nondeterministic TM accepts its input if any sequence of choices of moves leads to an accepting state.

Nondeterministic Turing machines

Formal definition- Nondeterministic Turing Machine

Nondeterministic Turing Machine M is denoted by the 9-tuple:
$M = (Q, \Sigma, \Gamma, \delta, B, \vdash, q_0, t, r)$

- Q is set of all states
- Σ is a finite set (the input alphabet)
- Γ is set of all tape alphabets $\Sigma \subset \Gamma$
- δ is set of transitions $(Q \times \Gamma) \rightarrow 2^{(Q \times \Gamma \times \{R,L\})}$
- B is blank symbol $(B \in \Gamma)$
- $\vdash$ is left end marker $(\vdash \in \Gamma)$
- q_0 is the initial state $(q_0 \in Q)$
- t is accept state $(t \in Q)$
- r is reject state $(r \in Q)$
- $t, r \in Q$ are sink states

Nondeterministic Turing machines

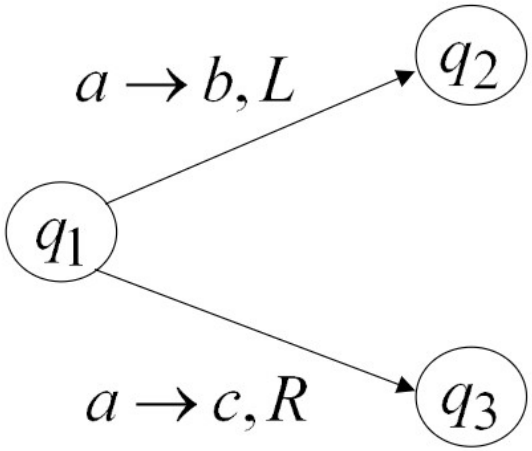


Figure: Nondeterministic Turing machine

Nondeterministic Turing machines

Every nondeterministic Turing Machine has an equivalent deterministic Turing machine

Theorem

- If L is accepted by a single-tape NDTM M_1, then L is accepted by some DTM M_2.
  - Let d be the maximum number of nondeterministic choices M_1 can make at any given move.
  - M_2 will systematically try all nondeterministic possibilities.
  - M_2 will have three tapes.
- M_2 uses 3 tapes:
  - Tape 1 holds the input,It's reused many times.
  - Tape 2 contains a sequence of digits from 1 to d, generated systematically. Each sequence dictates a sequence of choices. eg:
    - [1],[2], [3], ... , [d] ,
    - [1,1] , [1, 2] , [2, 1] , ... , [d, d] ,
    - [1,1,1] , [1,1,2], [1, 2, 1] , ... , [d, d, d] ,
  - Tape 3 contains a scratch copy of the input.

Nondeterministic Turing machines

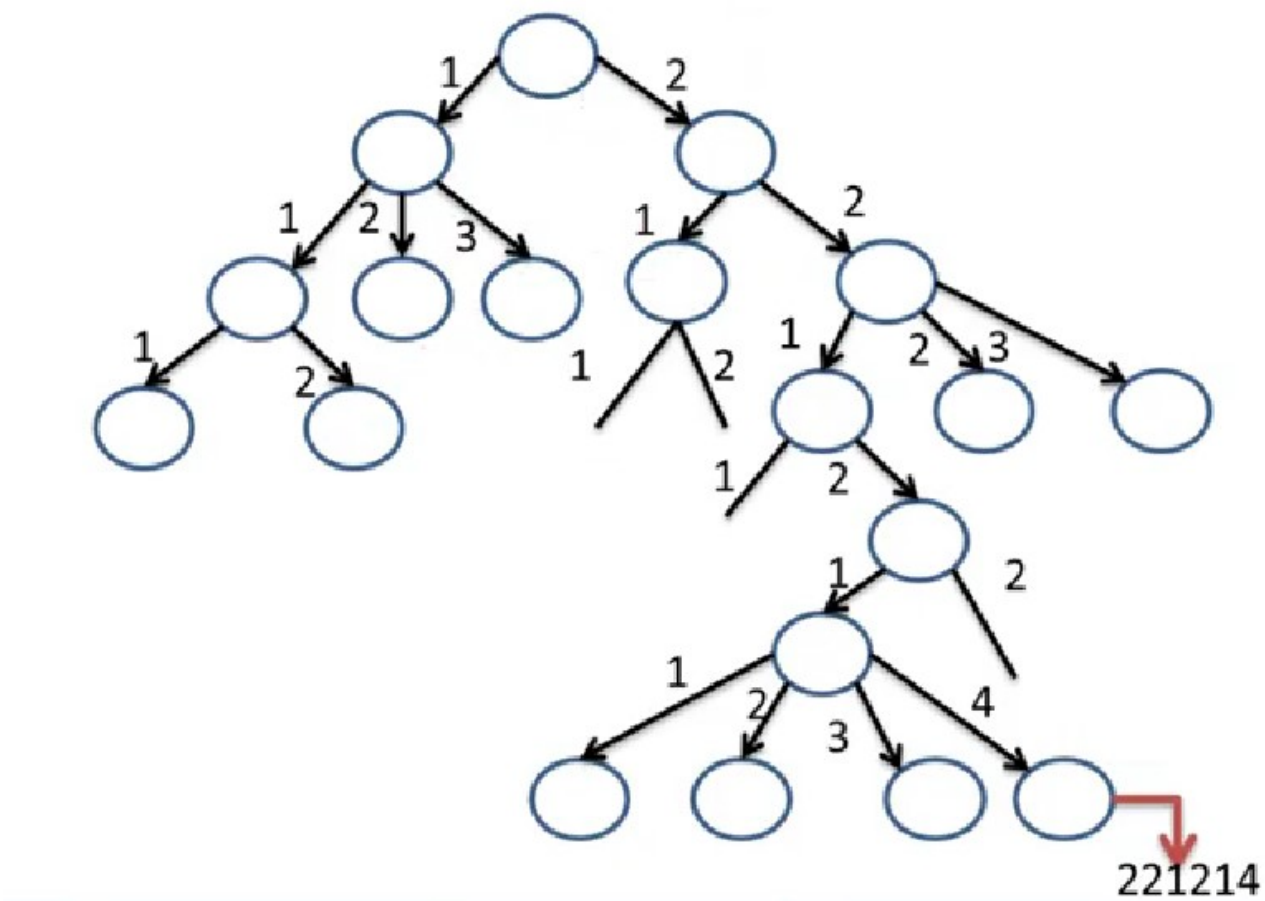


Figure: The computational tree of a non-deterministic Turing machine

Nondeterministic Turing machines

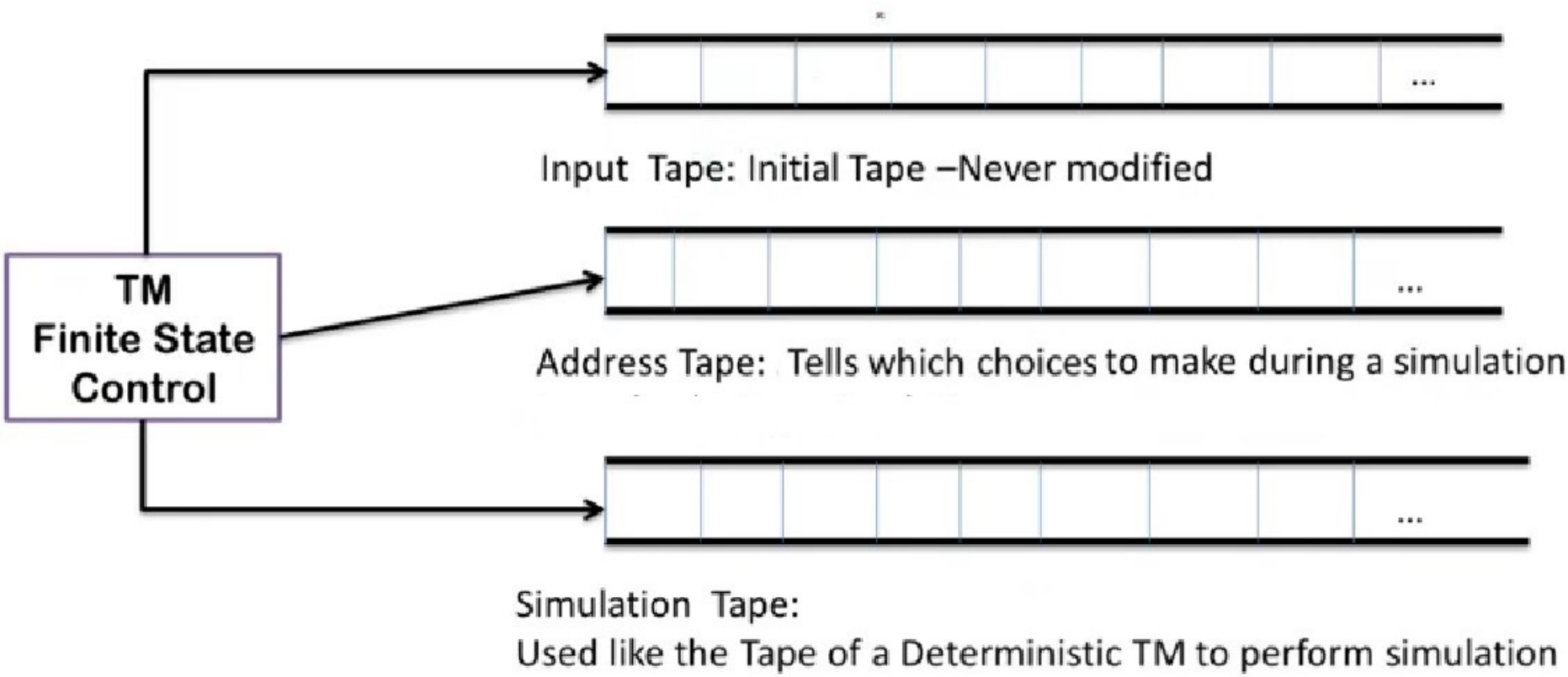


Figure: Deterministic TM simulating Nondeterministic Turing machine

Nondeterministic Turing machines

How M_2 works :

1. Generate the next sequence on tape 2
2. Copy tape 1 (input) to tape 3 (scratch copy)
3. Simulate M_1 on tape 3, making choices according to the sequence on tape 2.
4. If M_1 accepts, then accept;
5. Otherwise, go to step 1.

Universal Turing Machines

- A Turing machine (TM) U is called a universal Turing machine (UTM) if it is able to simulate all other TMs:
- for all TMs M and inputs w, $U(< M, w >) = M(w)$.
- A Universal turing Machine (UTM) takes two arguments
  1. The description of a normal TM
  2. The description of input string w and checks whether w is recognised
- The description of TM and input string w is the encoded string over a finite alphabet set.

Universal Turing Machines

Unary Encoding of the elements in a turing machine

A Turing Machine is a structure of the form:

$M = (Q, \Sigma, \Gamma, s, \delta, \vdash, B, t, r)$, where:

	Unary encoding on $\{1\}$
$Q : \{q_1, q_2, q_3, q_4, q_5\}$	{1,11,111,1111,11111}
$\Sigma : \{a_1, a_2, a_3\}$	{1,11,111}
$\Gamma : \{a_1, a_2, a_3, a_4, a_5\}$	{1,11,111,1111,11111}
$s : q_1$	1
$\delta : (q_1, a_2) \rightarrow (q_3, a_1, R), \ldots$	(1,11) → (111,1,11)
$\vdash : a_4$	1111
$B : a_5$	11111
$t : q_4$	1111
$r : q_5$	11111

Universal Turing Machines

Total coding(Binary Encoding) of the TM is

A Turing Machine is a structure of the form:

$$\mathsf{M} = (Q, \Sigma, \Gamma, s, \delta, \vdash, \mathsf{B}, t, r), \text{ where:}$$

$Q : \{q_1, q_2, q_3, q_4, q_5\}$

$\Sigma : \{a_1, a_2, a_3\}$

$\Gamma : \{a_1, a_2, a_3, a_4, a_5\}$

$s : q_1$

$\vdash : a_4$

$\mathsf{B} : a_5$

$t : q_4$

$r : q_5$

$\delta : (q_1, a_2) \rightarrow (q_3, a_1, R), (q_2, a_3) \rightarrow (q_4, a_5, L)$

$1^5\ 0\ 1^3\ 0\ 1^5\ 0\ 1\ 0\ 1^4\ 0\ 1^5\ 0\ 1^4\ 0\ 1^5\ 0\ 1\ 0\ 1^2\ 0\ 1^3\ 0\ 1\ 0\ 1^2\ 0\ 1^2\ 0\ 1^3\ 0\ 1^4\ 0\ 1^5\ 0\ 1$

Universal Turing Machines

Encoding of an input string

- Consider $\Sigma = \{a, b\}$
- Using unary encoding on the alphabet set $\{1\}$
  - a can be represented as 1 and
  - b can be represented as 11.
- and each successive character representations are separated with a 0
- Let $w = abab$ be the string to be checked on the Turing machine, M. The input to UTM will be
  - $\hat{w} = 101101011$
- UTM uses the binary code of the turing machine, M on string abab and will check if abab is recognised by M. If true UTM, will halt to say yes and if false UTM will stop to say no.

Universal Turing Machines

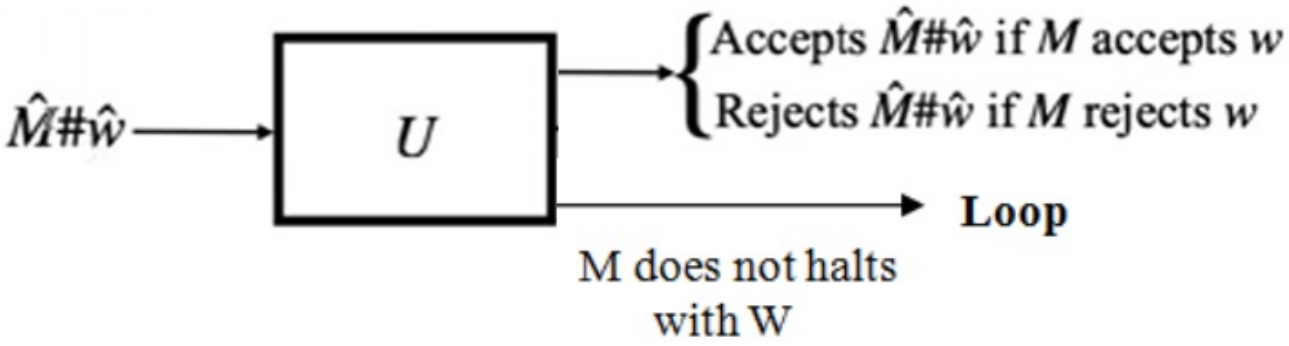


Figure: Universal Turing Machines

$1^5\ 0\ 1^3\ 0\ 1^5\ 0\ 1\ 0\ 1^4\ 0\ 1^5\ 0\ 1^4\ 0\ 1^5\ 0\ 1\ 0\ 1^2\ 0\ 1^3\ 0\ 1\ 0\ 1^2\ 0\ 1^2\ 0\ 1^3\ 0\ 1^4\ 0\ 1^5\ 0\ 1\ \#\ 1\ 0\ 1^2\ 0\ 1\ 0\ 1^2$

Figure: Input to Universal Turing Machines

Universal Turing Machines

- U takes as input the binary encoding $\hat{M}$ of turing Machine M and the binary encoding $\hat{w}$ of an input w to M, seperated by a $\#$ ($\Sigma_U = \{0, 1, \#\}$)
- Given the input string $\hat{M}\#\hat{w}$:
  - Checks whether $\hat{M}$ is a valid encoding of a Turing Machine and $\hat{w}$ is a valid encoding of a string. Rejects if encoding is not valid
  - Run M on w and:
    a. Accepts if M accepts w.
    b. Rejects if M rejects w.
    c. Loops if M loop on w

Universal Turing Machines

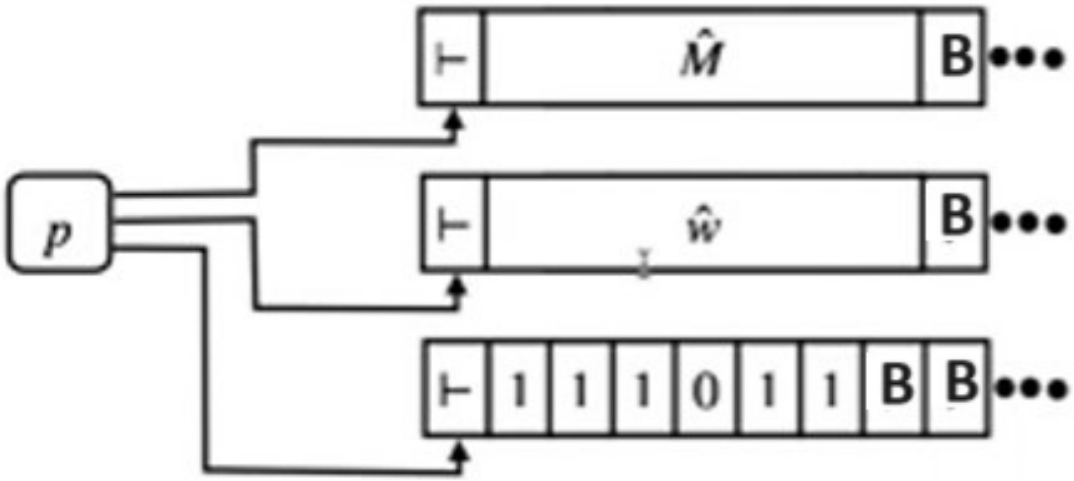


Figure: Structure of a UTM

- Tape 1 storing the encoding $\hat{M}$ of the TM M
- Tape 2 storing the encoding $\hat{w}$ of the i/p w
- Tape 3 storing the Current state of M and the current position of tape head in M

Universal Turing Machines

- U repeatedly performs the following, to simulate a single step of M on w:
  1. Scan tape 3 and find M's current state and the current position of the tape head.
  2. Scan tape 2 and find the next input symbol of M
  3. Scan tape 1 and find the transition to apply in the current configuration of M.
  4. Apply the transition in the current configuration of M and accordingly , update tape2 and 3
  5. If M enters $t_M \big| r_M$ then enter $t_U \big| r_U$

Halting Problem of TM

Halting Problem of TM

- In computability theory, the halting problem is the problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running, or continue to run forever.
- Alan Turing proved in 1936 that a general algorithm to solve the halting problem for all possible program-input pairs cannot exist.
- Halting problem is undecidable.

Halting Problem of TM

Problem

- Given the description of a Turing machine M and an input w, does M, when started in the initial configuration q_0w, perform a computation that eventually halts?
- **Input:** A Turing machine and an input string w.
- **Problem:** Does the Turing machine finish computing of the string w in a finite number of steps? The answer must be either yes or no

Halting Problem of TM

Proof:

- At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself.
- We will call this Turing machine as a Halting machine that produces a 'yes' or 'no' in a finite amount of time.
- If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'.
- The following is the block diagram of a Halting machine.

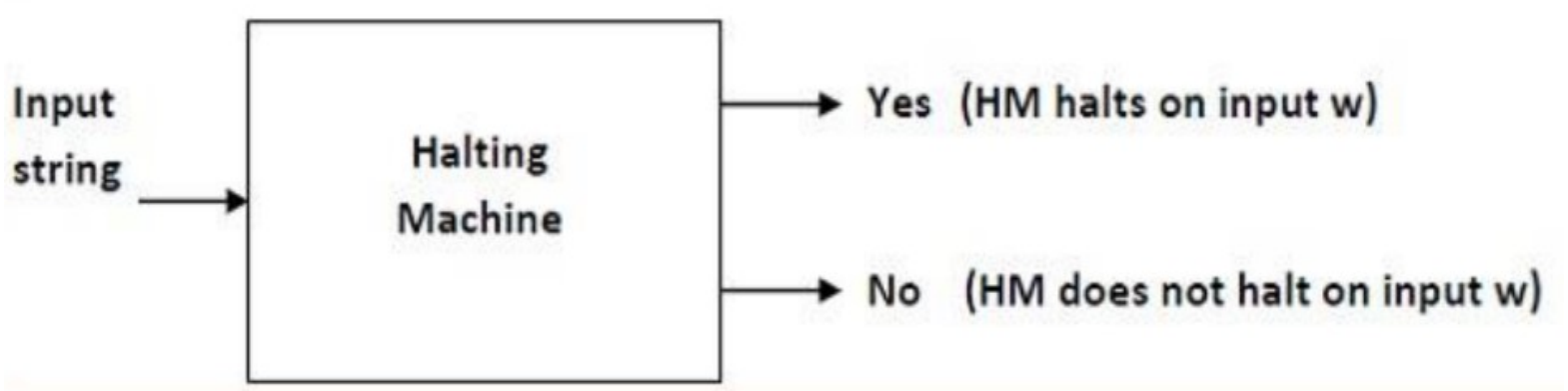


Figure: Halting machine

Halting Problem of TM

Proof cont..

- Now we will design an inverted halting machine (HM)' as
  - If H returns YES, then loop forever.
  - If H returns NO, then halt.
- The following is the block diagram of an 'Inverted halting machine'

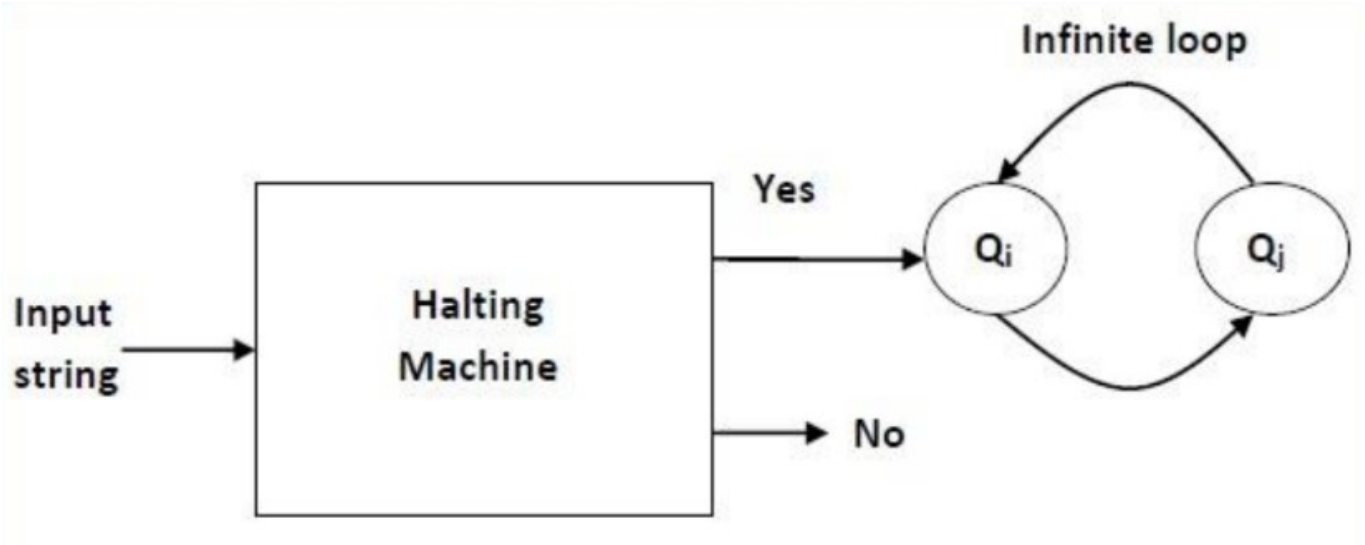


Figure: Inverted halting machine (HM)'

Halting Problem of TM

Proof cont..

- Further, a machine HM_2 which input itself is constructed as follows
  - If HM_2 halts on input, loop forever.
  - Else, halt.
- Here, we have got a contradiction. Hence, the halting problem is undecidable.

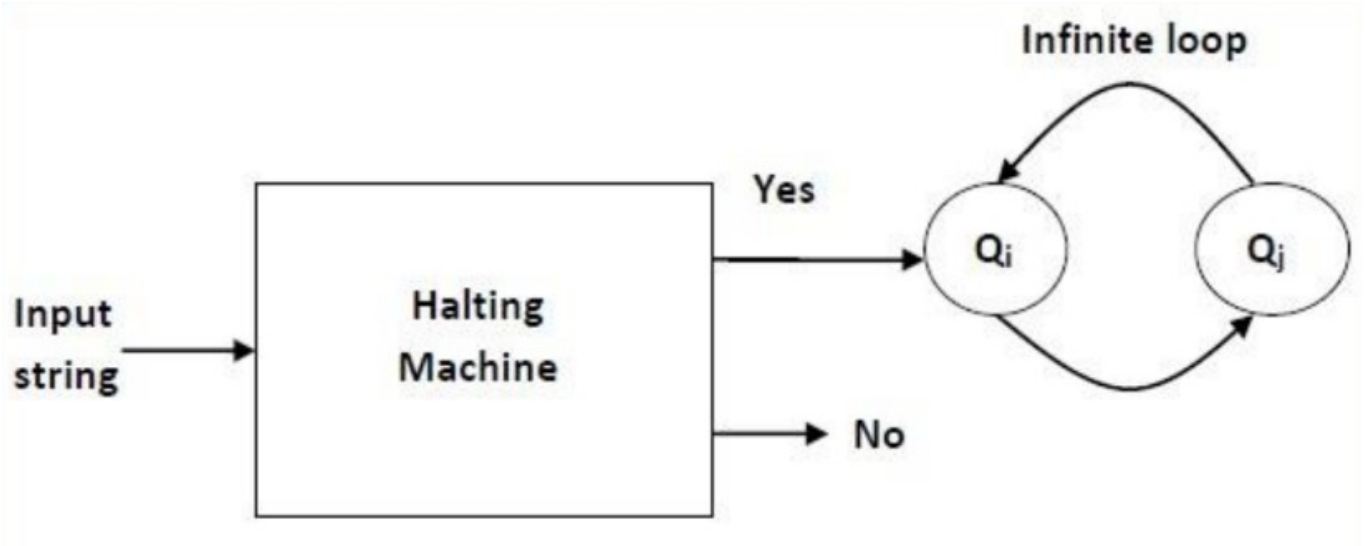


Figure: Inverted halting machine (HM)'

Recursive and Recursive Enumerable Language

Recursive set

- A set is called recursively enumerable if there's a step-by-step method, like using a specific rule or algorithm, that can eventually list out all the numbers in that set.
- However, this method might take a while, and it can't quickly tell you whether a specific number is in the set or not. So, it's a way to gradually list the numbers but not a straightforward way to decide if a particular number belongs to the set.
- The union and intersection of two recursively enumerable sets are also recursively enumerable.

Recursive and Recursive Enumerable Language

Recursive set

- A set S of integers is said to be recursive if there is a total recursive function $f(x)$ such that $f(x) = 1$ for x in S and $f(x) = 0$ for x not in S.
- Any recursive set is also recursively enumerable.
- Finite sets, sets with finite complements, the odd numbers, and the prime numbers are all examples of recursive sets.
- The union and intersection of two recursive sets are themselves recursive, as is the complement of a recursive set.

Recursive and Recursive Enumerable Language

Recursively enumerable languages

- A language is said to be recursively enumerable if there exists a Turing Machine that accepts every string of the language and rejects the string that are not in the language and it may cause TM to enter into an infinite loop.
- Turing machines for recursively enumerable languages may not always halt, and continue to run endlessly for strings that are not a part of that language. Recursively enumerable languages are also called Turing recognizing languages.

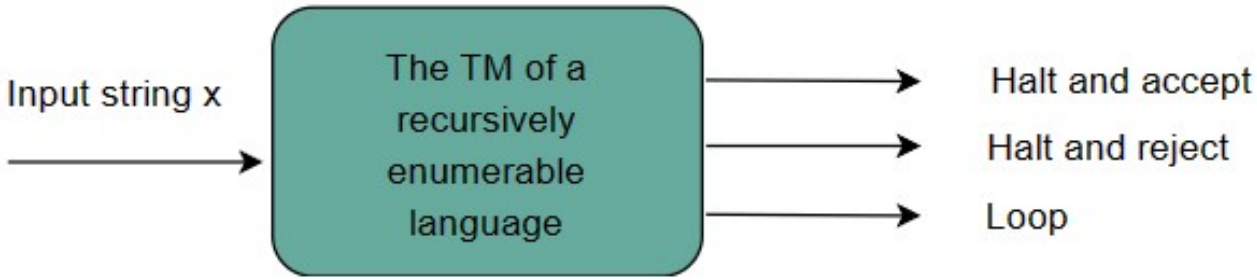


Figure: The output of a TM running over a recursively enumerable language

Recursive and Recursive Enumerable Language

Recursive Language

- A language is said to be recursive if there exists a Turing Machine that accepts every string of the language and rejects the string that are not in the language.
- A TM of a recursive language never goes into a loop and always halts in every case. Recursive languages are hence also called Turing decidable languages.

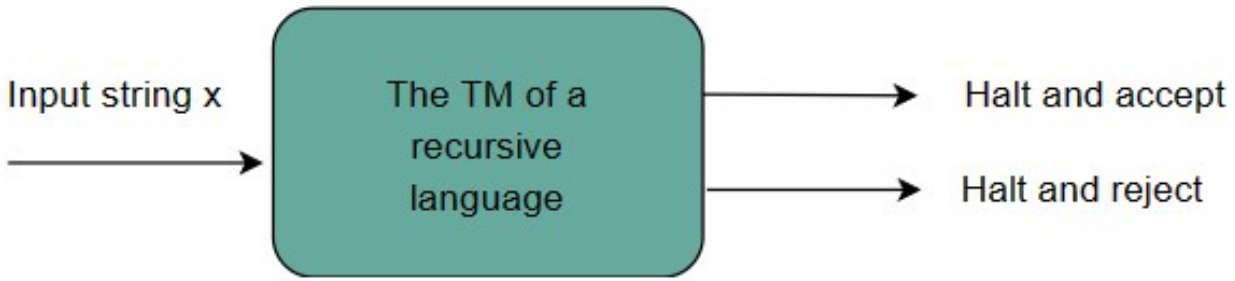


Figure: The output of a TM running over a recursive language

Recursive and Recursive Enumerable Language

- All the Recursive Languages are Recursively Enumerable.
- Every Recursively Enumerable Languages are not Recursive.
- Recursive Languages are subset of Recursively Enumerable Languages.

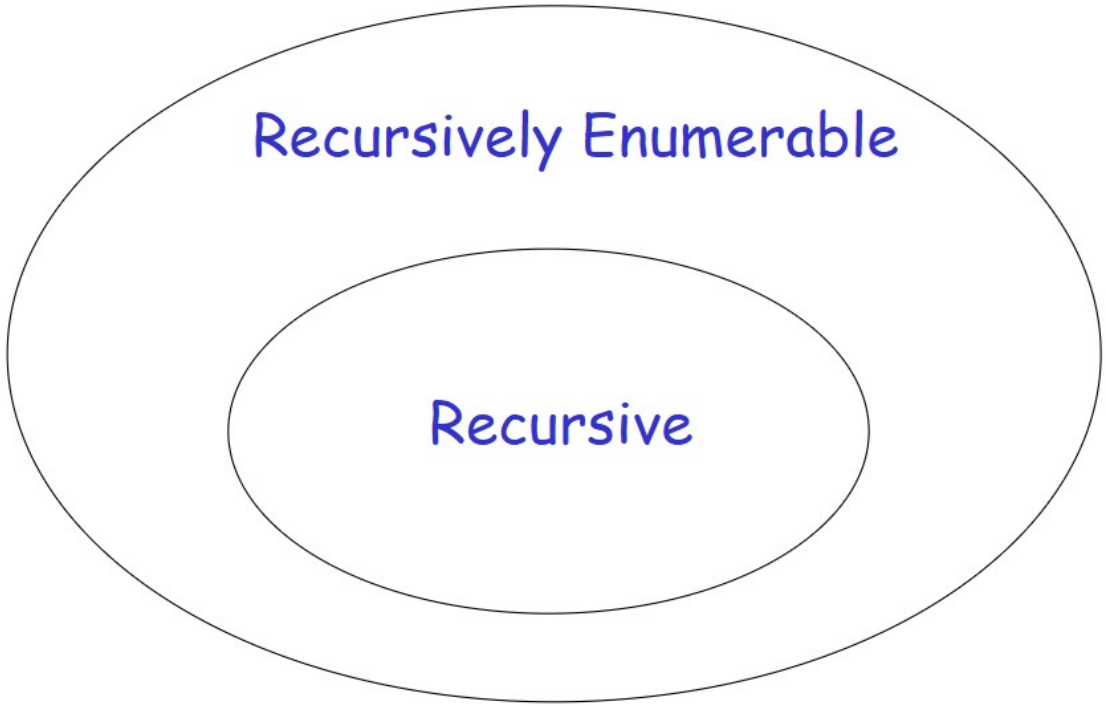


Figure: Recursive and Recursive Enumerable Language

Recursive and Recursive Enumerable Language

Closure properties of recursive languages

- Union
- Intersection
- Complement
- Kleene's closure

Recursive and Recursive Enumerable Language

Union:

- The union of two recursive languages L_1 and L_2 is also a recursive language.
- That is, if the TM accepts any one of the languages, it also accepts $L_1 \cup L_2$

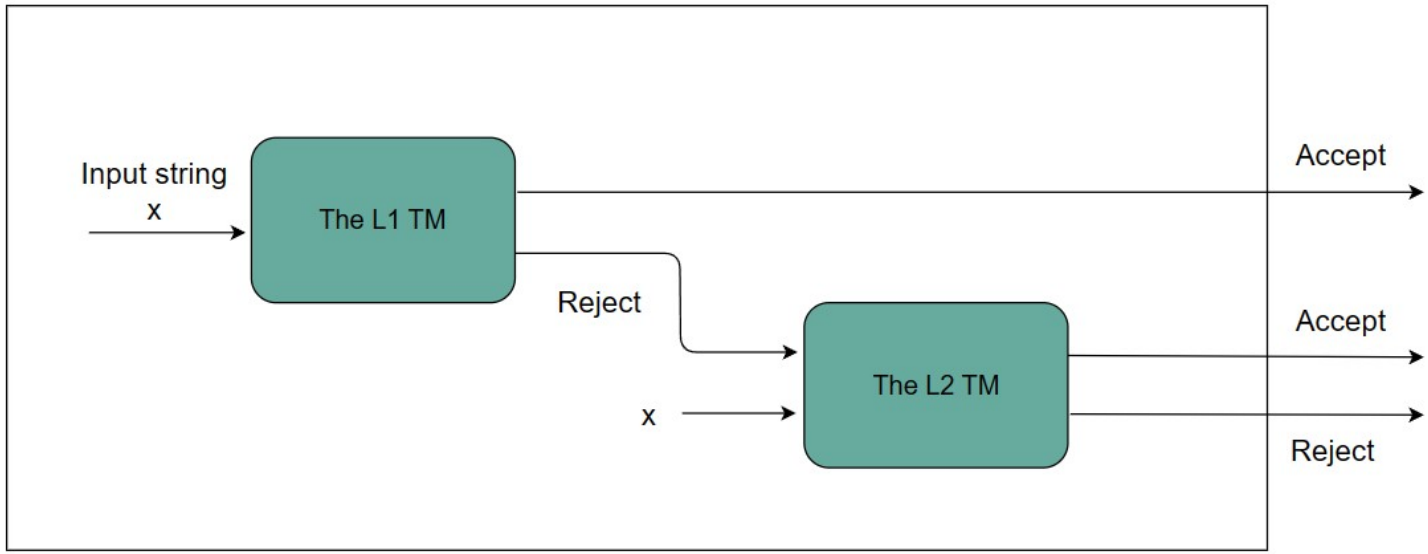


Figure: The union of two recursive languages

Recursive and Recursive Enumerable Language

Intersection:

- The intersection of two recursive languages L_1 and L_2 is also a recursive language.
- That is, if the TM accepts both languages, it also accepts $L_1 \cap L_2$

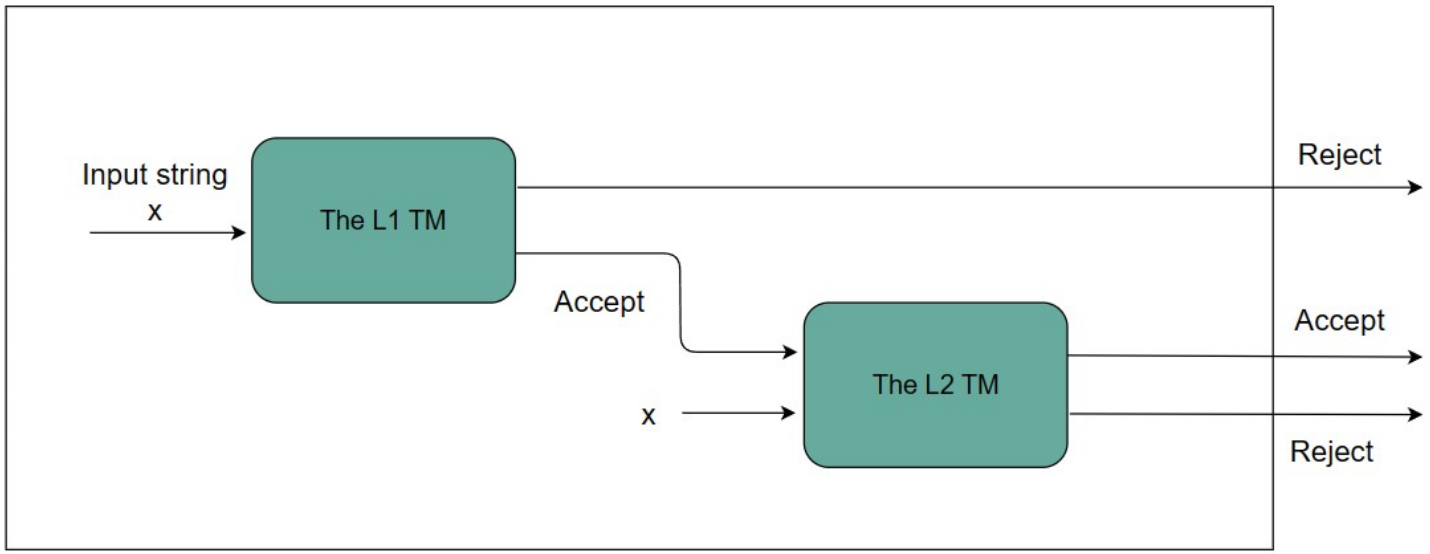


Figure: The intersection of two recursive languages

Recursive and Recursive Enumerable Language

Complement:

- The complement of a recursive language L_1 is also a recursive language.
- That is, if the TM accepts L_1 it rejects it as a final output.

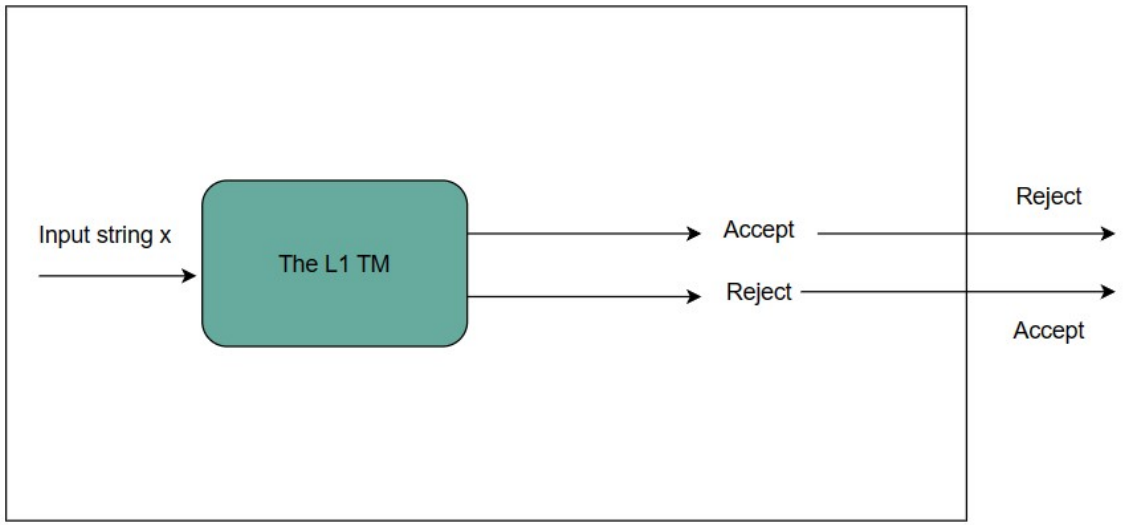


Figure: The complement of a recursive language

Recursive and Recursive Enumerable Language

- If L is recursive, then so is $\bar{L}$.
  - LetL be decided by a deterministic M.
  - Swap the "accept" state and the "reject" state of M.
  - The new machine decides $\bar{L}$

Recursive and Recursive Enumerable Language

Kleene's closure:

- The Kleene's closure of a recursive language L_1 is also recursive
- That is, if $L_1 = \{o^i 1^j\}$ is recursive, then $L_1^* = \{o^i 1^j\}^*$ is also recursive.

Recursive and Recursive Enumerable Language

Closure properties of recursive Enumerable Language

Recursive Enumerable Languages are closed under

- Union
- Intersection
- Concatenation
- Kleene's closure

* The complement of a recursively enumerable language is not necessarily recursively enumerable. There exist recursively enumerable languages L such that $\bar{L}$(complement of L) is not recursively enumerable.

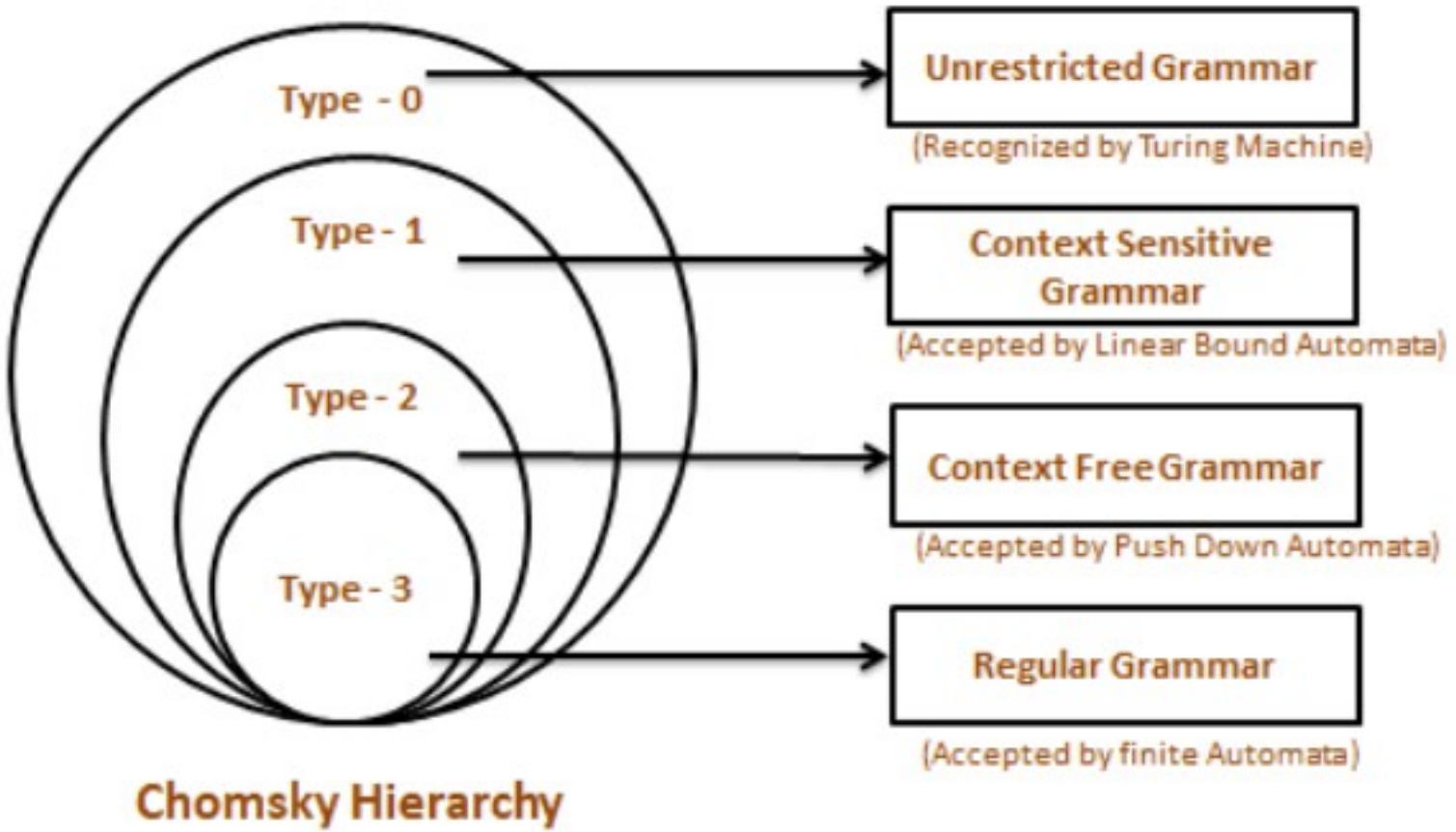
Recursive and Recursive Enumerable Language

Theorem:
If L is recursively enumerated and $\bar{L}$ is recursively enumerable then L is recursive. Proof

- Let M and M' be TMs that accept L and $\bar{L}$, respectively.
- Run M and M' simultaneously.
- For any word x, it must be accepted by one of M or M'
- So, either M or M' will halt and accept
- If M halts and accepts then M' halt and reject
- If M halts and reject then M' halt and accepts

The TM that runs M and M' simultaneously always halts and accepts or rejects so it decides L and L is recursive.

Chomsky classification of formal languages



Chomsky classification of formal languages

Type 0

- All formal grammars fall under type-0 grammars.
- The Turing machine can detect languages with type 0 grammar.
- The Recursively Enumerable languages are another name for these languages.
- Type 0 Language Production in the form of $\alpha \rightarrow \beta$ where
  - α , $\beta \in (V + T)^*$
  - V = Variables
  - T = Terminals.
- In type 0 there must be at least one variable on the Left side of production.
- For example:

$$S \rightarrow ACaB$$

$$Bc \rightarrow acB$$

$$CB \rightarrow DB$$

$$aD \rightarrow Db$$

Chomsky classification of formal languages

Type 1

- It is also known as Context Sensitive Language.
- It is recognized by the Linear Bound Automata.
- According to the following rules, context-sensitive grammar:
  - It should be Type 0
  - Multiple symbols may be present on the left side of the production rules for the context-sensitive grammar.
  - The number of symbols on the left side cannot be more than the number of symbols on the right side.
  - In type 1, Production is in the form of $\alpha A\beta \rightarrow \alpha\gamma\beta$ where A in V, α, $\beta \in (V \cup T)^*$and $\gamma \in (V \cup T)^+$
  - The rule of the form $A \rightarrow \epsilon$ is not allowed unless A is a start symbol.
- For Example:

$$S \rightarrow AB$$

$$AB \rightarrow abc$$

$$B \rightarrow b$$

Chomsky classification of formal languages

Type 2

- It should be Type 1
- It is Context-Free Grammar (CFG)
- It is recognized by a Pushdown automata
- The left-hand side of production can have only one variable and there is no restriction on right side
  - Production is of the form $A \rightarrow \alpha$,
  - where A is a variable and α is a string of symbols from $(V \cup T)^*$.
- For Example:

$$S \rightarrow AB$$
$$A \rightarrow a$$
$$B \rightarrow b$$

Chomsky classification of formal languages

Type 3

- It should be Type 2
- It is a Regular Grammar
- It can be accepted by a finite-state automaton.
- It is the most restricted form of grammar.
- It can be described using regular expressions.
- These languages can be modelled by NFA or DFA.
  - Production is of the form

$$V \rightarrow VT/T \ (left - linear \ regular \ grammar)$$

$$(or)$$

$$V \rightarrow TV/T \ (right - linear \ regular \ grammar)$$

For Example:

$$S \rightarrow aS$$

$$S \rightarrow bS$$

$$S \rightarrow a|b$$