THEORY OF COMPUTATION
UNIT 3

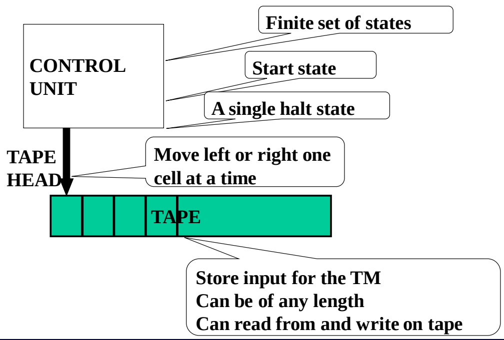
 U3. 1

Learning Objective

- Explain the concepts of Turing Machine and its extension.
- Explain the Concept of decidability and different undecidable problems
- Understand the concept of reducibility
- Explain the concept or recursion and recursion theorem

 U3. 2

Turing Machine



 U3. 3

Turing Machine

- What does a Turing Machine Does
  - Determine if an input x is in a language.
    - ✓That is, answer if the answer of a problem P for the instance x is “yes”.
  - Compute a function
    - ✓Given an input x, what is f(x)?
- How does Turing Machine work
  - For each move, a TM
    - ✓reads the symbol under its tape head
    - ✓According to the *transition function* on the symbol read from the tape and its current state, the TM:
      - write a symbol on the tape
      - move its tape head to the left or right one cell or not
      - changes its state to the *next state*

Turing Machine

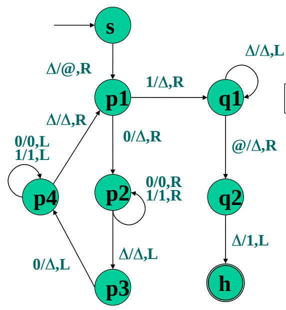
- When does a Turing Machine stop
  - When it gets into the special state called halt state. (halts)
    - ✓The output of the TM is on the tape.
  - When the tape head is on the leftmost cell and is moved to the left. (hangs)
  - When there is no *next state*. (hangs)

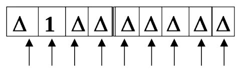
Formal definition of Turing machine

- A Deterministic Turing Machine is a quintuple $(Q, \Sigma, \Gamma, \delta, s)$, where
  - The set of states Q is finite, not containing halt state h,
  - The input alphabet Σ is a finite set of symbols not including the blank symbol Δ,
  - The tape alphabet Γ is a finite set of symbols containing Σ, but not including the blank symbol Δ,
  - The start state s is in Q, and
  - The transition function δ is a partial function from

$$Q \times (\Gamma \cup \{\Delta\}) \rightarrow Q \cup \{h\} \times (\Gamma \cup \{\Delta\}) \times \{L, R, S\}.$$

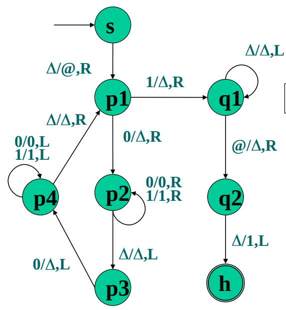
Example of Deterministic Turing machine

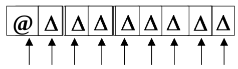




On the input 0001000, the TM halts.

Example of Deterministic Turing machine





On the input 0000000, the TM hangs.

Configuration of Turing Machine

- Let T Q Σ Γ, δ s) be a DTM

  A configuration of T is an element of

  $Q \cup \{h\} \times (\Gamma\cup\{\Delta\})^* \times (\Gamma\cup\{\Delta\}) \times (\Gamma\cup\{\Delta\})^*$
- Can be written as
  - (q,l,a,r) or
  - $(q,l\underline{a}r)$

string to the right of tape head

symbol under tape head

Current state

string to the left of tape head

Turning Machine as a language acceptor

- Definition

  Let $T=(Q, \Sigma, \Gamma, \delta, s)$ be a TM, and $w \in \Sigma^*$.

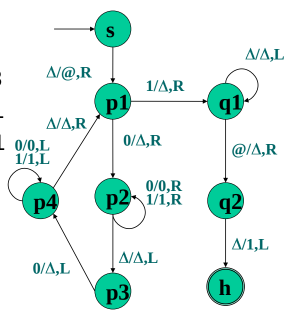
  T accepts w if $(s, \varepsilon, \Delta, w) \mid\!-_T^* (h, \varepsilon, \Delta, 1)$.

  The language accepted by a TM T, denoted by $L(T)$, is the set of strings accepted by T.

Example of TM as a Language acceptor

$L(T)=\{0^n10^n \mid n \geq 0\}$

- T halts on 0^n10^n
- T hangs on $0^{n+1}10^n$ at p3
- T hangs on 0^n10^{n+1} at q1
- T hangs on $0^n\ 1^2\ 0^n$ at q1



Turing Machine computing a function

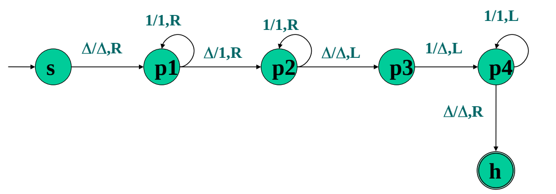
- Definition

  Let $T=(Q, \Sigma, \Gamma, \delta, s)$ be a TM, and f be a function f Σ^* to Γ^*. T computes f if, for any string w in Σ^*,

  $(s, \varepsilon, \Delta, w) \mid\!-_T^* (h, \varepsilon, \Delta, f(w))$.

Example of Turing Machine computing a function

Computation of sum of two integer



 U3. 13

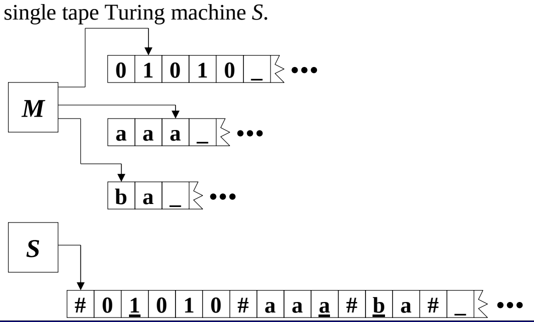
Mutitape Turing Machine

- In this extension, the machine can have several tapes, each with its own read/write head. We focus on the case where there are two tapes.
- In one step, the machine reads the symbols scanned by all heads and then, depending on those symbols and its current state, each head will move or write and the control unit will enter a new state.
- Note that the actions of the heads are independent of each other: in one step one head might move left and another might move right or write an "a".

 U3. 14

Mutitape Turing Machine

- Every multitape Turing machine *M* has an equivalent single tape Turing machine *S*.



 U3. 15

Multiple Head Turing Machine

- This is a single-tape Turing machine with *K* read/write heads.
- The heads are numbered 1 through K.
- The move of the TM depends on the state and on the symbol scanned by each head.
- In one move the heads may move independently left, right, or remain stationary.
- The move function of the two head can be defined as
- δ(state, symbol under Head1,Symbol under Head2) = (New state, (S_1, M_1), (S_2, M_2)) [Assumed only two head]
- Every multi-head Turing machine M_1 has equivalent single head Turing machine M_2. (How?)

Multi-dimensional Turing Machine

- This type of Turing machine consists of *k*-dimensional array of cells infinite in all 2*k* directions, for some fixed *k*.
- Initially the input is along one axis, and the head is at the left end of the input.

B	B	B	B	B	B	B
B	B	a	b	a	B	B
B	B	b	a	a	B	B
B	B	a	a	b	B	B
B	B	B	B	B	B	B

- Transition function : δ (q, a) = {p, A, L/R/U/D}

Multi-dimensional Turing Machine

- Every Multi-dimensional tape Turing machine has equivalent single tape Turing machine

B	B	B	B	B	B	B
B	B	a	b	a	B	B
B	B	b	a	a	B	B
B	B	a	a	b	B	B
B	B	B	B	B	B	B

- The "B" represents Blank symbol. The equivalent single tape represented by following.

B	a	b	a	B	b	a	a	B	a	a	b	B	B	B	B

- Each row is separated by blank space.

Non-deterministic Turing Machine

- An NTM starts working and stops working in the same way as a DTM.
- Each move of an NTM can be nondeterministic.
- Each move in NTM reads the symbol under its tape head
- According to the *transition relation* on the symbol read from the tape and its current state, the TM choose one move non-deterministically to:
  - write a symbol on the tape
  - move its tape head to the left or right one cell or not
  - changes its state to the *next state*

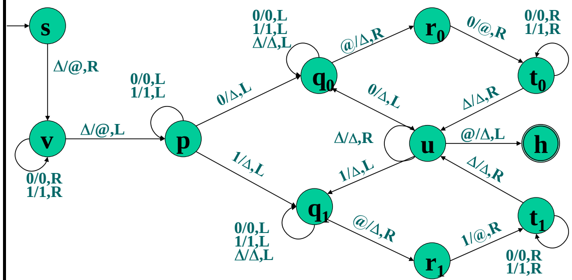
Definition of Non-deterministic Turing Machine

- NTM a quintuple $(Q, \Sigma, \Gamma, \delta, s)$,

  where
  - the set of states Q is finite, and does not contain halt state h,
  - the input alphabet Σ is a finite set of symbols, not including the blank symbol Δ,
  - the tape alphabet Γ is a finite set of symbols containing Σ, but not including the blank symbol Δ,
  - the start state s is in Q, and
  - the transition f^n^ $\delta: Q\times(\Gamma\cup\{\Delta\})\rightarrow 2^{Q\cup\{h\}\times(\Gamma\cup\{\Delta\})\times\{L,R,S\}}$.

Example of Non-deterministic Turing Machine

- Let $L=\{ww \mid w\in\{0,1\}^*\}$



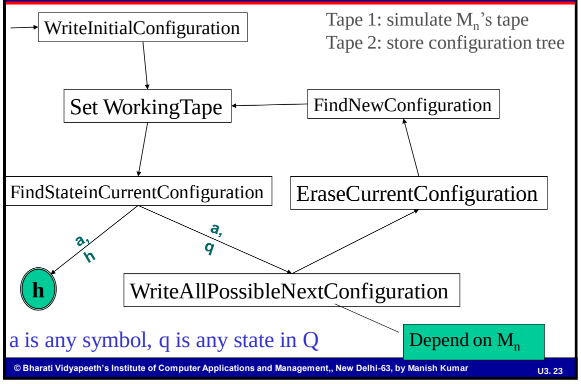
Equivalence of NTM to DTM

- **Theorem:** For any NTM M_n, there exists a DTM M_d such that:
  - if M_n halts on input α with output β, then M_d halts on input α with output β, and
  - if M_n does not halt on input α, then M_d does not halt on input α.

Proof:

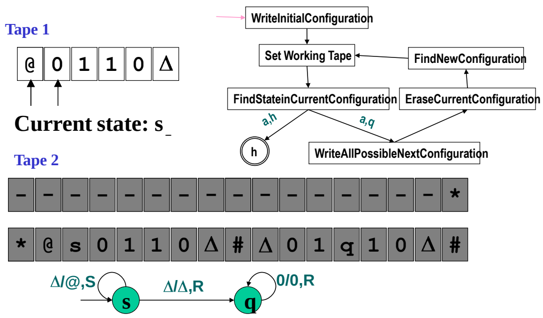
- Let $M_n = (Q, \Sigma, \Gamma, \delta, s)$ be an NTM.
- We construct a 2-tape TM M_d from M_n as follows:

Construct a DTM equivalent to NTM

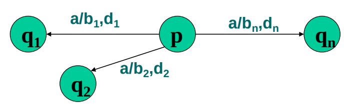


a is any symbol, q is any state in Q

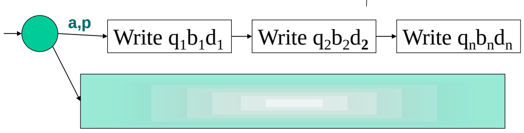
Working of equivalent DTM



Write All possible Configuration



For each $(p,a,q_i,b_i,d_i) \in \delta$, $1 \le i \le n$

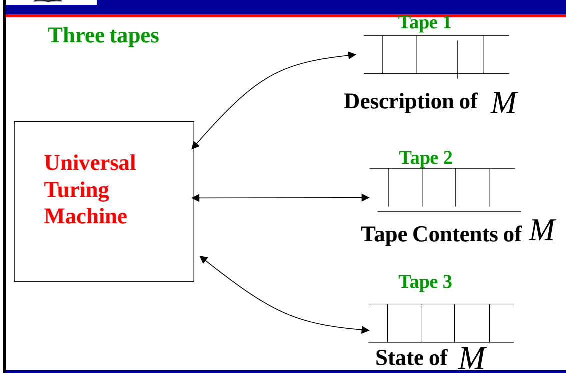


Universal Turing Machine

- Turing Machines are "hardwired" means they execute only one program.
- But, Real Computers are re-programmable.
- So, the solution is Universal Turing machine which has the capability of reprogrammable and simulation of any other Turing machine.
- Universal Turing machine can simulate any Turing machine "M" on string "w".
- Therefore, it accepts a description of Turing machine "M" and the input string "w".

Universal Turing Machine

Three tapes



Universal Turing Machine

- The description of Turing machine is encoded by the binary string.
- The encoding of a Turing machine involves following four types of encoding
  - Alphabet encoding
  - State encoding
  - Head movement encoding
  - Transition encoding
  - Turing Machine encoding

Universal Turing Machine

Alphabet Encoding

Symbols:	a	b	c	d	$\cdots$
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
Encoding:	1	11	111	1111	

Universal Turing Machine

State Encoding

States:	q_1	q_2	q_3	q_4	$\cdots$
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
Encoding:	1	11	111	1111	

Universal Turing Machine

Head Move Encoding

Move: L R

Encoding: 1 11

Transition Encoding

Transition: $\delta(q_1, a) = (q_2, b, L)$

Encoding: 1 0 1 0 11 0 11 0 1 0 1

separator

 U3. 31

Universal Turing Machine

Turing Machine Encoding

Transitions:

$\delta(q_1, a) = (q_2, b, L)$ $\delta(q_2, b) = (q_3, c, R)$

Encoding:

1 0 1 0 11 0 11 0 1 00 11 0 11 0 111 0 111 0 11

separator

 U3. 32

Church-Turing Thesis

- Church-Turing Thesis: ***Whenever there is a effective method (algorithm) for obtaining the values of a mathematical function, the function can be computable by a TM.***
- ***An effective computable method is one in which there is finite no. of steps to find the solution i.e. a mechanical process to derive the result also known as algorithm.***
- Hence Church-Turing thesis defines "algorithm" .
- Strong Church-Turing thesis : ***A Turing machine can do anything that a computer can do.***

 U3. 33

Church-Turing Thesis

- It is a thesis not theorem because it has no mathematical proof.
- Arguments to accept the Church -Turing thesis.
  - No one has been able to suggest a counter example to disprove.
  - Some Turing machines can also perform anything that can be performed by digital computer.
  - There are several alternative model of computation like Random Access Machine, Post Machine etc. But no one is more powerful than Turing machine.
- The universal Turing machine gives the idea about the reprogrammable machine and it was used in the invent of modern computer.

Decidability

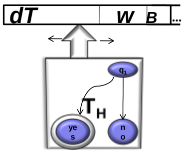
- A language is "*recursive*" if there exists a Turing machine that accepts every string of language and rejects every string over the same alphabet that is not in the language
- A language is "recursively enumerable" if there exists a Turing machine that accepts every string of the language, and does not accept strings that are not in the language.
- Strings which are not in the language may be rejected or may cause the Turing machine to go into an infinite loop.
- Every Recursive language is also recursively enumerable. But it is not clear if every recursively enumerable language is also recursive.

Decidability

- A problem whose language is recursive is said to be decidable means there exists a Turing machine that accepts an instance of problem and determine whether the answer is "yes" or "no".
- The best known undecidable problem is the Halting Problem of TM.
- Halting Problem : "*Given an arbitrary Turing Machine T as input and equally arbitrary input w, decide whether T halts on w.*"
- Basically UTM that takes a TM, *T* as its input, and simulates the *T* running on input *w*, and returns or decides whether or not *T* halts on *w*.

Halting Problem is Undecidable proof

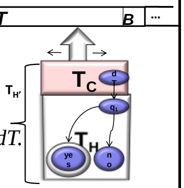
- Formulate a proof, suppose such a machine does exist, call it T_H.
- Let w be input for T.
- Let T be encoded as a description for T_H.
- If T accepts and halts on w, then T_H will give an equivalent result and transfer to the halting "yes" state.
- If T does not halt on w, then T_H will transfer to the halting "no" state.
- If T_H exists, then we can construct another machine $T_{H'}$ by modifying T_H.



Halting Problem is Undecidable proof

Construct a new machine T_H

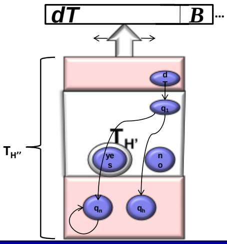
- Add another machine T_c (or some extra code) that makes a copy of dT and hands it to T_H's initial state.
- Alter T_H so that it decides if T halts on dT rather than w.
- T_H's only job is to decide if T halts on dT.
- If $T_{H'}$ exists, then we can construct another machine by modifying $T_{H'}$.



Halting Problem is Undecidable proof

Construct a new machine $T_{H''}$

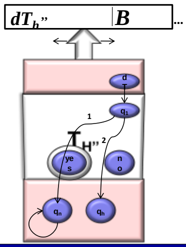
- Alter $T_{H'}$'s two halting transitions so that the *yes* and *no* state are diverted to two new states.
- The *yes* transition goes from q_1 to q_n, once in q_n it will never halt (infinite loop).
- The *no* transition goes from q_1 to q_h a halting state.



Halting Problem is Undecidable proof

The Halting Problem

- If $T_{H''}$ exists, then we can input its own description $dT_{H''}$.
- **Case 1:** If $T_{H''}$ halts on $dT_{H''}$, then $T_{H''}$ does not halt on $dT_{H''}$ because of an endless loop.
- **Case 2:** If $T_{H''}$ does not halt on $dT_{H''}$, then $T_{H''}$ does halt on $dT_{H''}$.
- This ***contradicts*** that T_H ever existed in the first place.
- The Halting Problem is not solvable by any TM hence undecidable.



Reducibility

- A reduction is a way of converging one problem into another problem in such a way that solution to the second problem can be used to solve first problem.
- Suppose someone wants to find the way around new city. He / She can reduce the problem of finding way around city to the problem finding the way on Google map.
- Reduction does not mean to make smaller, it means to transform or convert a problem X to another problem Y that is at least as hard as X. Usually Y is at least as hard as X, and so we express a reduction from X to Y as $X \leq Y$.
- Reducibility can be used to show that a problem is unsolvable.

Reducibility

- Use the idea that " If A is undecidable and reducible to B, then B is undecidable."
- Suppose R decides $HALT_{TM}$. We construct S to decide A_{TM}.
- S = "On input <M, w>
  1. Run R on input <M, w>
  2. If R rejects reject.
  3. If R accepts, simulate M on w until it halts.
  4. If M has accepted, accept; If M has rejected, reject."
- Since A_{TM} is reduced to $HALT_{TM}$, $HALT_{TM}$ is undecidable

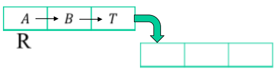
Recursion Theorem

- Statement : Let T be a TM that computes a function
  $t: \Sigma^* \, x \, \Sigma^* \rightarrow \Sigma^*$

  There is TM R that computes a function $r: \Sigma^* \rightarrow \Sigma^*$ where for every w, $r(w) = t(<R>, w)$

  Proof:

  - We construct a TM R in three parts A, B, T where T is given by the statement of theorem.



Recursion Theorem

- In figure, A is TM $T_{<BT>}$ and described by g(<BT>).
- To preserve the input w, we redesign q so that $T_{<BT>}$ write its output following any string preexisting on the tape.
- When A runs the tape contains w<BT>.
- B is a procedure that examines the tape and applies q to its content. The result is <A>.
- Now B combines ABT and obtain its description <ABT> = R
- Finally it encodes together with w and place the resulting string.

Recursion Theorem

- Hence, the recursion Theorem justifies the use of certain recursive definitions.
- However, note that there are some recursive definition that are only satisfied by the completely undefined function.
- It gives the ides that a program can operate on themselves.

