Other Models of Turing Machine

* **Equivalence of Two Automata**

- Two automata are equivalent if they accept same language.
- If for every language automata M_1 in C_1, there is automata in C_2 such that

$$L(M_1) = L(M_2)$$

then we say C_2 is as powerful as C_1. If the converse also holds and for every M_2 in C_2 there is an M_1 in C_1 such that $L(M_1) = L(M_2)$, we say that C_1 and C_2 are equivalent.

Turing Machine with stay option

→ In stay R/W head will stay in same position

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$

Proof

TM with stay option ≡ standard Turing Machine

stay option TM is extension of standard TM, so it can simulates standard TM moves by ignoring stay as option

Stay TM to std TM

stay TM

$$\delta(q, a) \vdash (q_j, b, L/R)$$

Std TM

$$\hat{\delta}(\hat{q}, a) \vdash (\hat{q}_j, b, L/R)$$

Stay TM

$\delta(q_i, a) \vdash (q_j, b, S)$

Normal TM

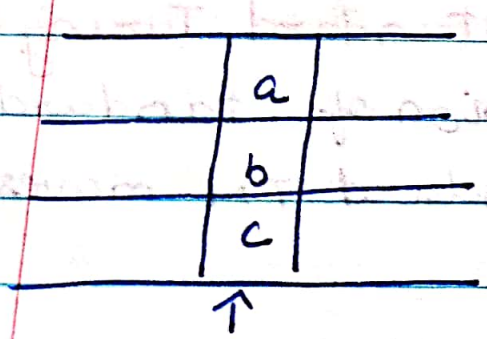
$\hat{\delta}(q_i, a) \vdash (\hat{q}_{js}, b, R)$

$\hat{\delta}(\hat{q}_{js}, c) \vdash (q_j, c, L) \quad \forall\ c \in \Gamma$

Stay option implementation - to implement stay option in standard TM just move to right by replacing symbol then move left for all symbol.

Multiple Track TM

When we divide each cell of tape into three parts called tracks, each containing one member of triplet



→ in multi-track TM we read all symbol as one

→ input size & number of track.

Turing Machine with semi-Infinite Tape

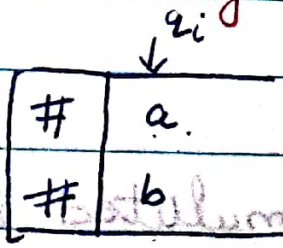
→ In this TM is unbounded only in one direction
→ This TM is identical to standard TM except that no left move is permitted when the read/write head is at boundary.
→ This restriction doesnot effect the power of machine

Equivalence of semi-infinite and standard Turing Machine

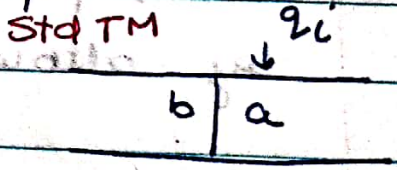
M - standard TM
$\hat{M}$ - TM with semi-infinite tape

Simulating machine $\hat{M}$ has a tape with two tracks.
→ on the upper we keep information to right of some reference point of M's tape

semi-infinite TM

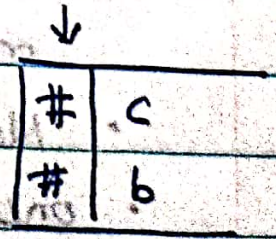


Std TM



$(q_i, a) \vdash (q_j, c, L)$

simulting machine



$\hat{\delta}(\hat{q}_i, (a,b)) \vdash (\hat{q}_j, (c,b), L)$

$\hat{\delta}(\hat{q}_i, (\#,\#)) \vdash (\hat{q}_j, (\#,\#), R)$

State P_i is used to signify move for symbol of lower track.

Turing Machine with more complex storage

1. Multitape TM - This TM has multi-tape and each with its own independtly controlled read write head.

δ: $Q \times \Gamma^n \longrightarrow Q \times \Gamma^n \times \{L, R\}^n$

for eg.

$\delta(q_0, a, e) \vdash (q_1, x, y, L, R)$

a → tape symbol of 1st tape

e → " 2nd tape

Equivalence of multi-tape and standard TM

→ Multitape TM can implement standard TM by allowing only one tape to be used

→ Single to Multi TM

- Multi tape TM can be simulated by multi track standard turing machine
- No. of track = 2 × no. of tape
- one track keep the symbol for tape and other track keep information of read/write head

Non-Deterministic Turing Machine

→ It adds nothing to power of turing machine

$\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$

Ex $\delta(q_0, a) \vdash \{(q_1, b, L), (q_2, c, R)\}$

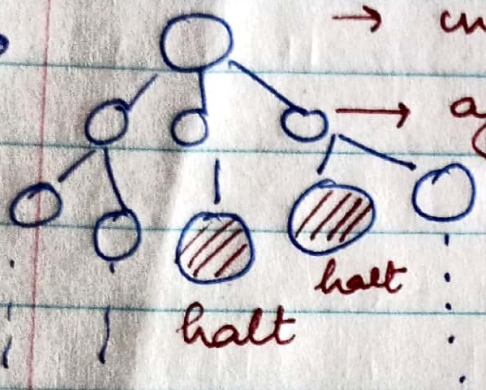
→ Nondeterministic automata are viewed as accepte

→ NTM is said to accept w if there is any possibl sequence of moves such that

$q_0, w \vdash x_1 q_f x_2$ and $q_f \in F$

→ w is reject if NTM move to non-finial state infinite loop.

Equivalence of NTM and DTM → NTM can be simulated by DTM using Multi-tape TM



→ For each different move the input is copied to different tape and moved accordingly.

→ The width of tree depends on the branching factor i.e. number of options available on each move

→ A Non-Deterministic TM M is said to accept a language L if for all w ∈ L at least one of the possible configuration accept w

Universal Turing machine

Universal TM Mu is an automata that given as input the description of TM M and a string w can simulate computation of M on w.

Assume

$Q = \{q_1, q_2 \ldots q_n\}$ where q_1 is intial and q_2 is finial

$\Gamma = \{a_1, a_2 \ldots a_m\}$ a_1 represent blank.

Encoding

q_1 as 1 a_1 as 1 0 is used as separator

q_2 as 11 a_2 as 11 between 1's

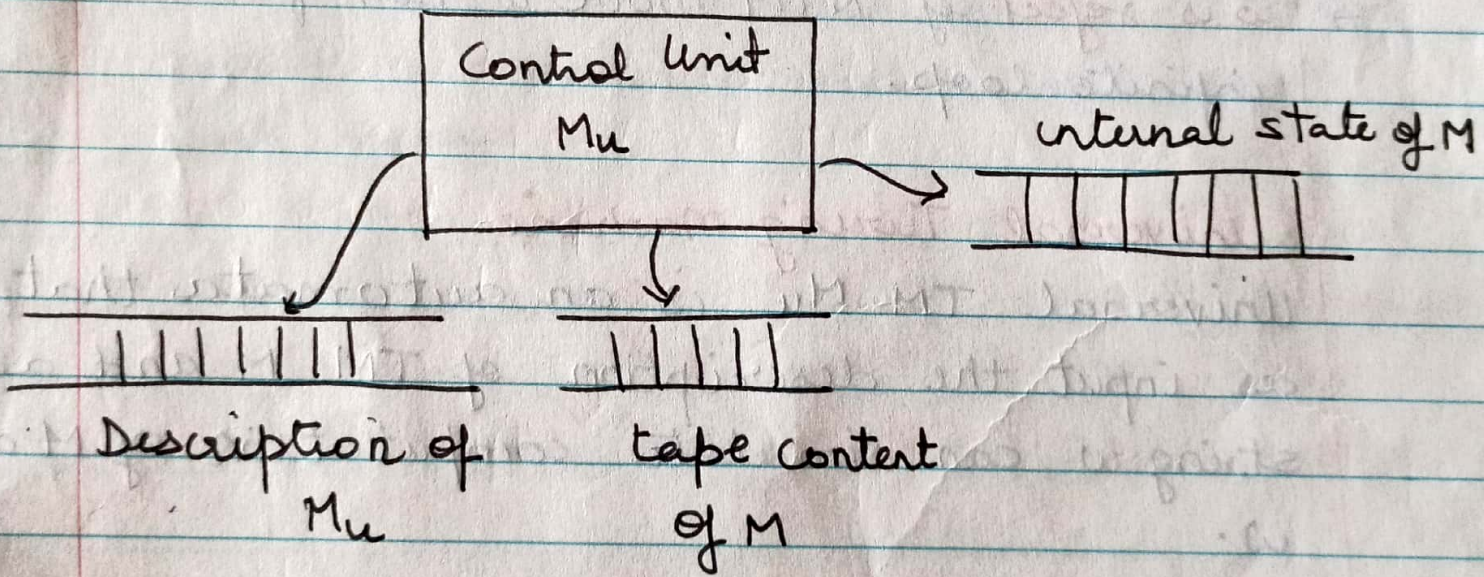
L → 1 R → 11

Ex.

$\delta(q_1, a_2) \vdash (q_2, a_3, L)$

10110110111010...

→ A universal turing machine Mu has an input alphabet that alphabet {0,1} and structure of multitape Machine



→ Tape 1 is consulted to see what M would do then tape 2 & 3 will be modified to reflect result on move.

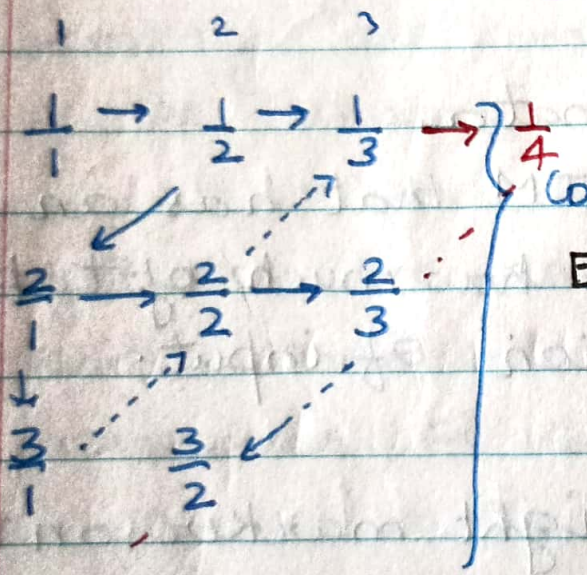
Basics from set theory

Countable Sets – A set is said to be countable if all its elements are put one to one correspondance with +ive integers

Ex set of all even integers

Uncountable set – if such correspondance cannot exist

Is quotient set of form P/q countable/uncountable



Countable

Enumeration procedure

1/1, 1/2, 2/1, 3/1, 2/2, 1/3, 1/4, 2/3, 3/2, 4/1, ...

Theorm set of turing machine, although infinite is countable

Proof. We can encode TM using 0 and 1

enumeration procedure

1. Generate next string $\{0,1\}^+$ in proper order
2. Check generated string to see if it defines TM, if yes write it on tape, if not ignore the string
3. Step to step 1

Recursive and Recursivdy enumerable language

→ A language is said to be R.E if there exists a TM that accepts it

$$q_0 w \vdash_M x_1 q_f x_2$$

→ A language L is said to be recursive if there exists a TM M that accept L and halt on every w in Σ^+

Recursive language

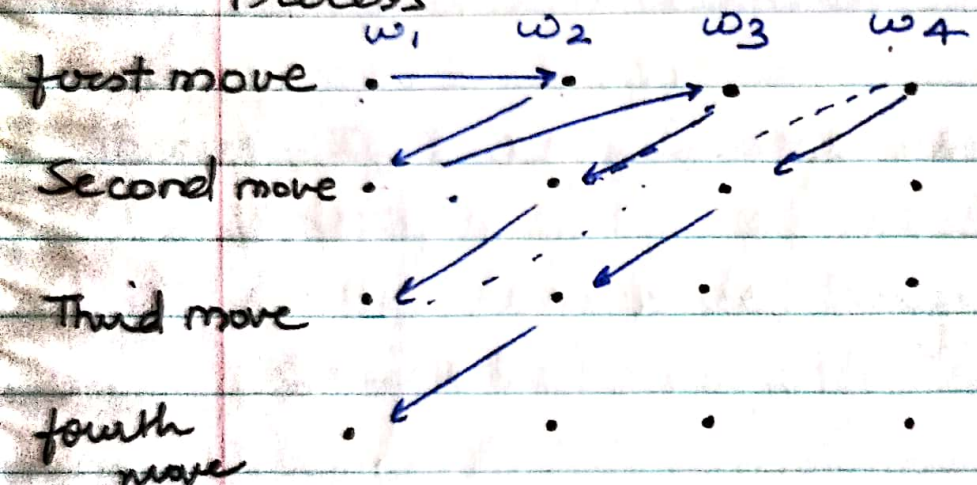
→ If L is recursive then there exists a enumeration process.

Enumeration process for Recursive language

- Suppose M is TM that determine membership is in recursive language L.
- Construct another TM say $\hat{M}$ that generate all string in Σ^+ in proper order
- All these string become input to M

Enumeration process for R.E language

→ Above method cannot be used for R.E as if for any w that stuck in infinite loop then we shall not be able to check other



Theorem If a language L and its complement $\bar{L}$ are both R.E then L is recursive.

Proof. If L and $\bar{L}$ are both R.E. then there exists TM M and $\hat{M}$ that serve as enumeration process for L and $\bar{L}$

→ if w is generated, then w belongs to either L or $\bar{L}$ then it accepted by M or $\bar{M}$

→ so membership for both L and $\bar{L}$ can be decided by ~~halt~~ halting TM, therefore both are recursive

Theorem If L and $\bar{L}$ are recursive then both are R.E

Proof Any recursive language is R.E so proof complete.

Theorem Recursive language are proper subset of R.E. language

or

There exist R.E. that is not recursive.

Proof. let L be R.E but $\bar{L}$ is not R.E.

→ so acceptance is defined by halting TM

but not rejection cannot be defined by halting.

→ If L and T are R.E then only one can be defined by halting TM else L is recursive language.

Theorm Let S be an infinite countable set. Then its power set is not countable

Proof. Let $S = \{s_1, s_2, s_3 \ldots\}$ then any element t of 2^S can be represented by a sequence of 0's and 1's with 1 at i^{th} pos iff $s_i \in t$

Eg. set $\{s_2, s_3, s_6\}$ is represented by 01100100...
$\{s_1, s_3\}$ by $\{1000100\ldots\}$

→ Suppose 2^S is countable
→ then element can be written in some $t_1, t_2 \ldots$
→ if 2^S is countable then its element should be written in some order as

t_1 (1) 0 0 0 ...
t_2 0 (1) 0 0 ...
t_3 0 0 (1) 0 ...
.
.
.

→ Now take Main diagonal
→ complement it
→ new diagonal is same 0,1 representation and it should belong to 2^S
→ But it cannot be represented as it differ from one (diagonal) pos
→ so it contradicts our logical impase that 2^S is countable.

Above argument is called Diagonalization

Theorm For any non empty Σ there exist langu
that are not R.E.

Proof → A language is subset of Σ^*.
→ Every such subset is language, there
set of all language is 2^{Σ^*}.

① As Σ^* is infinite, so 2^{Σ^*} is not count
(proof previous theorem)

② All TM can be enumerated so R.
is countable.

By ① and ② we can say that th
must be language that is not R.

Closure Properties of recursive language

1. Both L and $\bar{L}$ are recursive.
   For $\bar{L}$ make non final as final state

2. Neither L nor $\bar{L}$ is R.E.

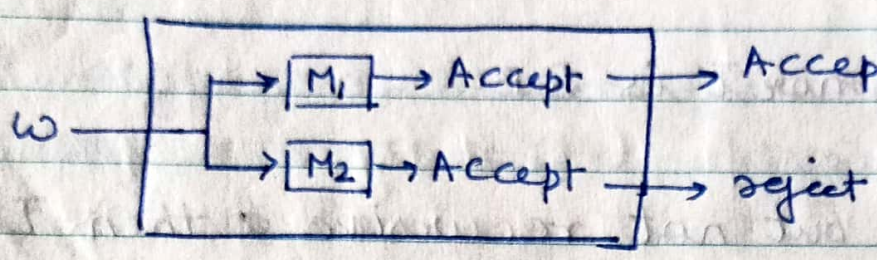
2. L is R.E but not recursive then $\bar{L}$ is not R.E.
   as final state is not defined so we can't construct accepting state for $\bar{L}$ in TM therefore $\bar{L}$ is not r.e.

3. $\bar{L}$ is R.E but not recursive and L is not R.E
   Same as above.

4. Set of R.E language are closed under intersection
   Let T_1 and T_2 be turing machine that accept r.e, language L_1 and L_2
   → Build a TM T_3 that first run T_1 on an input and if T_1 accept T_3 runs T_2 on that same input.
   → If T_2 accept than T_3 accept
   → if either T_1 or T_2 rejects then T_3 will reject
   So T_3 accept $L_1 \cap L_2$

If both a language L and its complement are RE, then L is recursive.

Proof



Let $L = L(M_1)$ and $\bar{L} = L(M_2)$ both M_1 and M_2 are simulated in parallel by a TM M. We can make M a two tape TM and then convert it one tape TM.

→ one tape of M simualtes the behaviour M_1
→ second tape of M " M_2
→ if $w \in L$ then M_1 accept so eventually M accept and halt.
→ if $w \notin L$ so M_2 will accept, M halt without accepting
→ Then M halts and $L(M) = L$, we conclude that L is recursive.

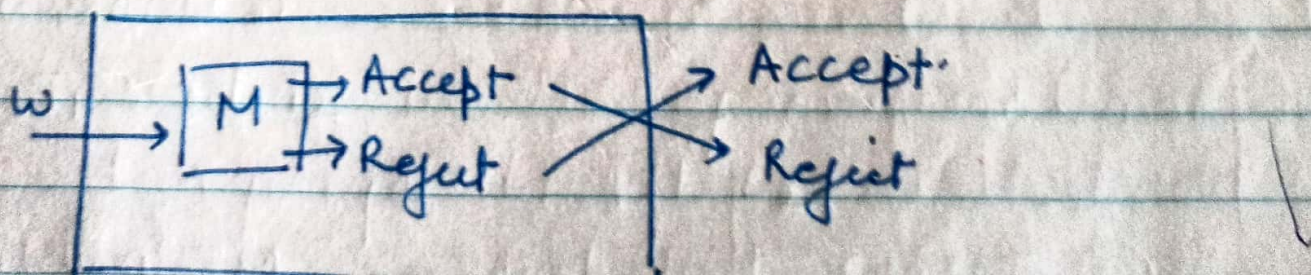
Theorem If L is recursive language so is $\bar{L}$

Let $L = L(M)$ for some TM M that always halt. we construct TM $\bar{M}$ such that $\bar{L} = \bar{L}(\bar{M})$ as follows

* The accepting state of M are made non accepting state of $\bar{M}$ with no transition i.e in these states $\bar{M}$ will halt without accepting

* $\bar{M}$ has new accepting state r, there are no from transition from r

* For each combination of a non accepting state and a tape symbol of M such that M has no transition (i.e. M halt without accepting) add a transition to accepting state r.



Decidability & Computability

Decidability - when result of computation is simple 'yes' or 'no', in this case we say a problem as decidable or undecidable.

Computability - A function is said to be computable only if there is TM that can compute it.

Important points

→ When we state decidability / undecidabilit we must always know what the doma is because this may affect the conclus

→ Problem may be decidable on some doma but not on another

→ A single instance of problem is always decidable.

① Turing machine halting problem

→ undecidable

→ because it is very difficult to find is on which TM can enter in infinite loop and TM is not halting.

② $L(G) = \phi$ is undecidable

→ Let G be unrestricted grammar. Then the problem of determining whether $L(G) = \phi$ is undecidable.

→ as we have to test all string (from infinite) domain if nothing is accepted than language is empty.

3. M is any turing machine. Then the question of whether or not L(M) is finite is undecidable

Proof. It is undecidable as its possible that a string may go in loop and then come to termination

Post Correspondence Problem

→ It is used to relate problem with existing halting problem

PCP can be stated as

→ Given any sequence of ∧ string on some alphabet Σ.

$$A = w_1 w_2 \ldots w_n$$

and

$$B = v_1 v_2 \ldots v_m$$

we say that there exist a PCP solution for pair (A, B) if there is non empty sequence of integer $i, j \ldots k$ such that

$$w_1 w_2 \ldots w_k = v_1 v_2 \ldots v_k$$

Ex. $\Sigma = \{0, 1\}$

A: $w_1 = 11$ $w_2 = 100$ $w_3 = 111$

B: $v_1 = 111$ $v_2 = 001$ $v_3 = 11$

if we take

$w_1 w_2 w_3 = v_1 v_2 v_3$ so there exist a PC solution

2. $\Sigma = \{0,1\}$

$w_1 = 00$ $\quad w_2 = 001$ $\quad w_3 = 1000$

$v_1 = 0$ $\quad v_2 = 11$ $\quad v_3 = 011$

no PC solution

→ This is a lengthy process. We go for modified PCP where $A \equiv B$ iff $w_1 w_2 \cdot w_j \cdot \cdot w_n = v_1 v_2 \cdot \cdot v_n$

→ Modified PCP and PCP is undecidable.