8/6/23 (TOC)

CLASSTIME Pg. No.
Date / /

Unit 3 & Unit 4

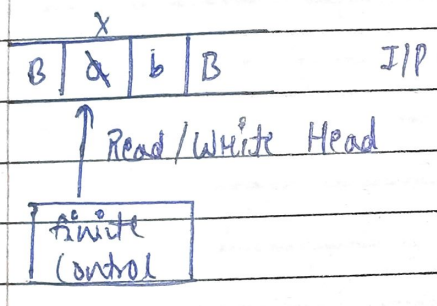
* Unit-3

- **Turing Machine**

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

- Q → States
- Γ → Symbols to be written in tape
- B → Blank State

$\Sigma \subseteq \Gamma$



$\delta = Q \times \Gamma \longrightarrow Q \times \Gamma \times (L, R)$ — Transition

$(q_0, a) \longrightarrow (q_1, \#, R)$

- In TM, we can Read / Write in I/P - O/P tape & in both left & right directions

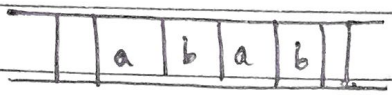
DTM (Deterministic) ≡ (NonDeterministic) NTM
(equal in Power)

- DTM: for an input, we can move either in left or in right dirn.
- NTM: for an input, we can move in both dirns.

- LBA (Linear Bounded Automata)

FA < PDA < LBA < TM

↳ TM + Input Size Tape



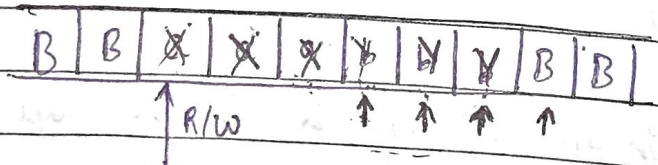
~~M = (Q, I~~ .

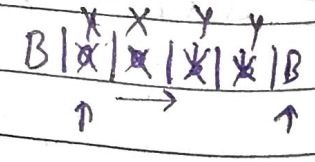
Standard Examples

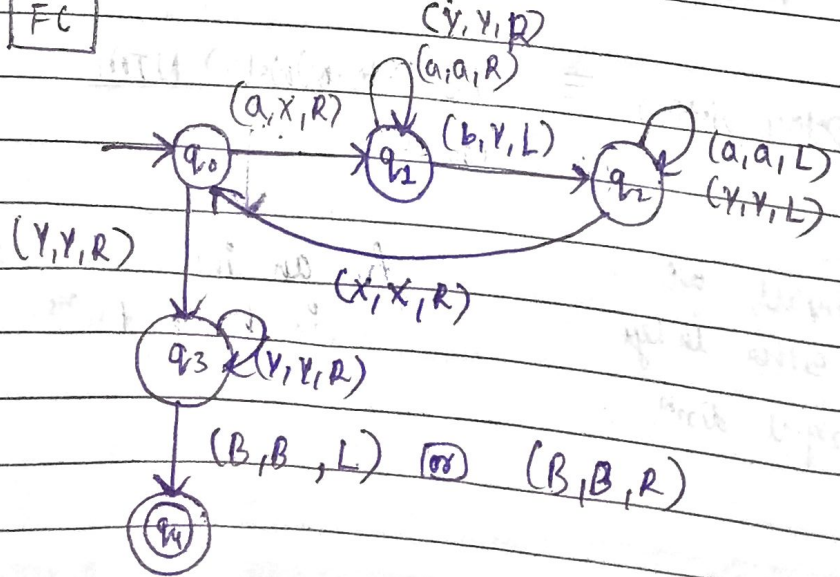
1) $L = \{ a^n b^n c^n : n \geq 1 \}$
2) $L = \{ a^n : n \text{ is prime} \}$
3) $L = \{ a^n : n \text{ is non prime} \}$
4) $L = \{ a^{n!} , n \geq 0 \}$
5) $L = \{ ww : w \in (a,b)^* \}$
6) $L = \{ www^R : w \in (a,b)^* \}$
7) $L = \{ a^n : n = m^2 , m \geq 1 \}$

★ Designing of TM

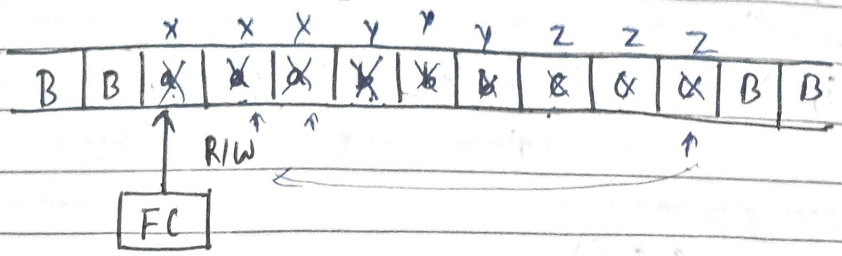
1) $\{ a^n b^n \mid n \geq 1 \}$

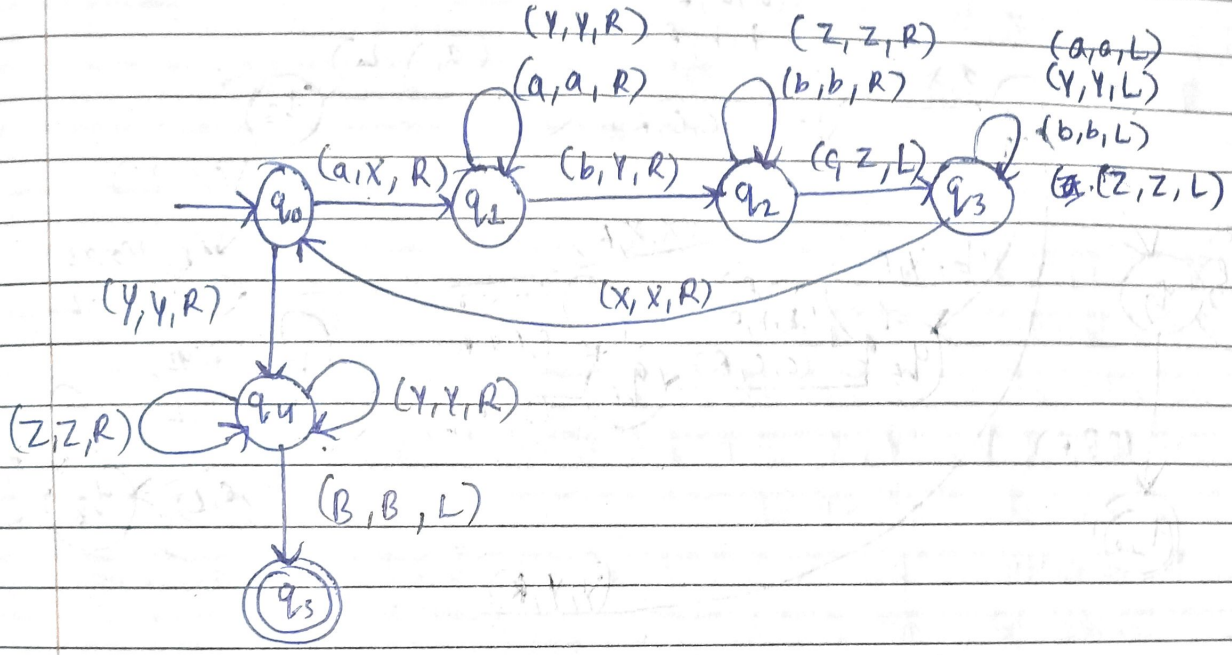




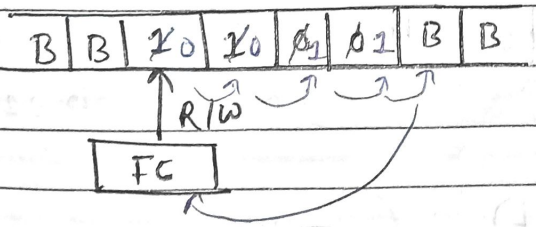


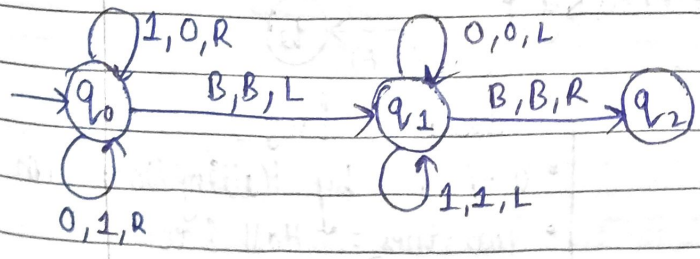
2) $\{ a^n b^n c^n \mid n \geq 1 \}$





3) Turing Machine for 1's complement

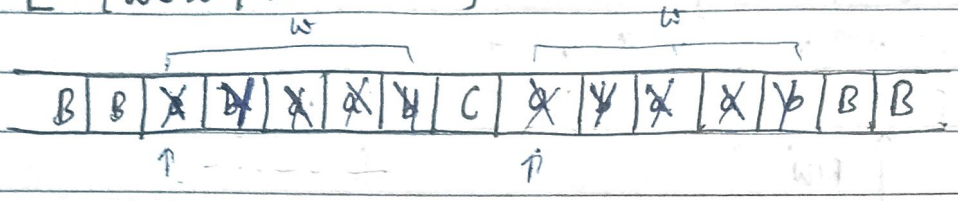


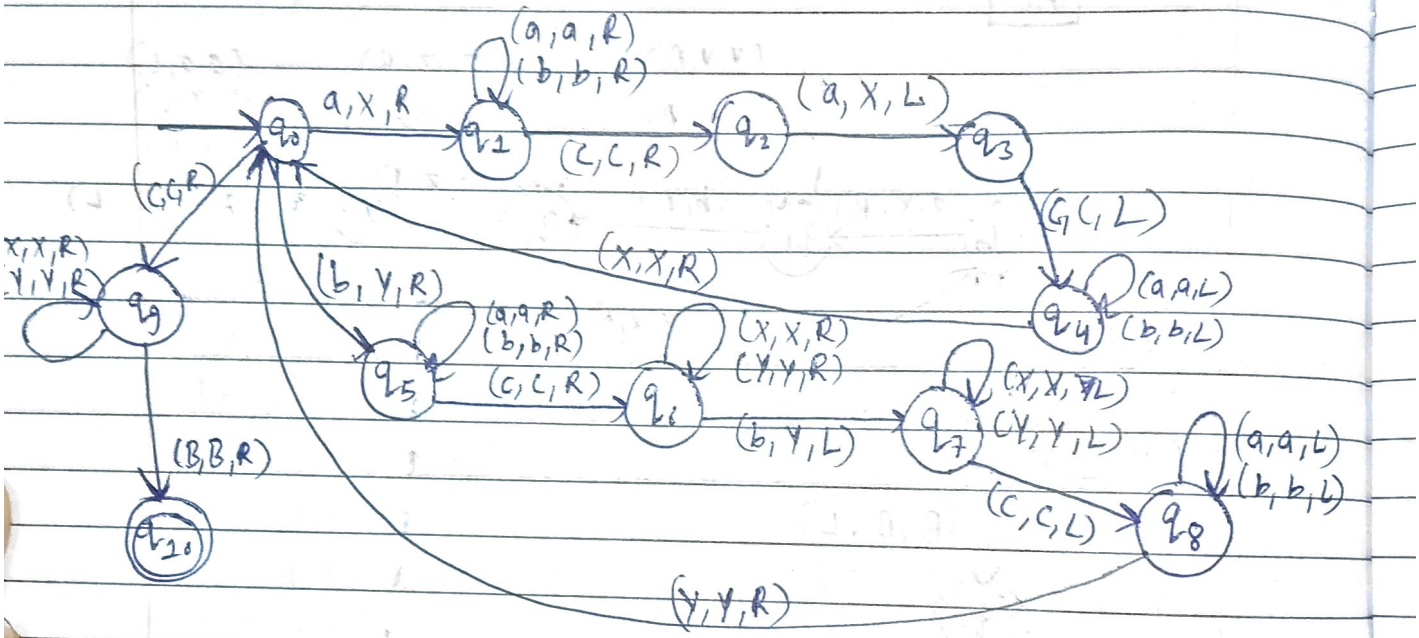


State	0	1	B
q_0	1 R q_0	0 R q_0	B L q_1
q_1	0 L q_1	1 L q_2	B, R, q_2
q_2	—	—	—

q B Y
↓ ↓ ↓
change dirn new state

4) $L = \{wcw \mid w \in (a,b)^*\}$



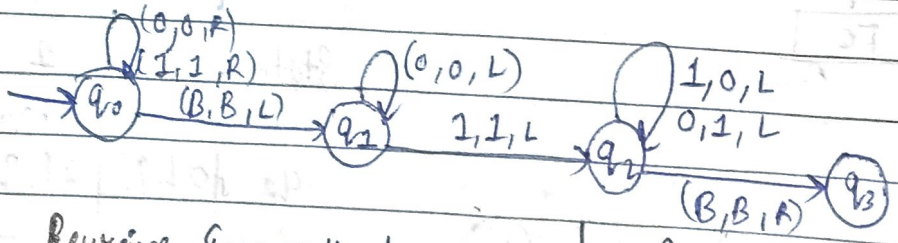


5) TM for 2's Complement

B	B	1	0	1	0	1	0	0	0	B	B

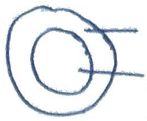
↑

2's C
↳ 0 1 0 1 1 0 0 0

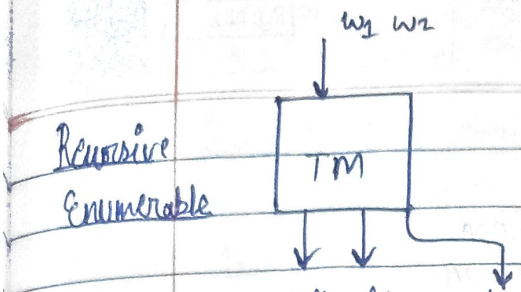


★ **Recursive Enumerable lang.**
- accepted by TM.
- Three States : - Halt & Accept
  - Halt & reject
  - Never Halt
- closed under all ~~accept~~ except → set diff., complement

Recursive Lang.
- accepted by Halting/Total TM
- Two states : Halt & accept
  - Halt & reject
- Closed under all except Homomorphism & Substitution



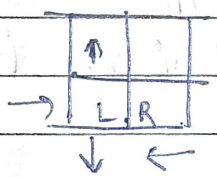
CLASSTIME Pg. No.
Date / /



(TM → halts means it is either accepting or rejecting the string)
↳ doesn't halts (∞ loop) → Undecidable whether to accept or not

☆ Turing Machine With Modifications

1) TM with Stay Option
2) TM with Semi infinite loop
3) Offline TM (Read / write — tape)
4) Multidimensional TM



5) NDTM
6) UTM (Universal) (3 tapes in TM)
7) Multitape TM
8) Multi head TM
9) TM with 3 states
10) Turing TM
11) Non erasing TM
12) Always writing TM

Same Power on the basis of accepting languages

1) TM without writing Capability → FA
2) TM with input size Tape → LBA
3) TM with Tape used as Stack → NDTM → NPDA
DTM → DPDA

4) TM with finite tape → FA

☆ TM can work as a transducer also
↳ converting I/P to O/P

☆ Church Turing Thesis (1936)
- A function on natural no. is computable by an algo if & only if it is computed by TM
- Anything that can be done by current digital computers can also be done by TM
- Currently, there is no problem which can be solved by digital comp. but not by TM.
- No mathematical model is more powerful than TM.

☆ Decidable Languages
- A Language is decidable if it is a recursive lang.
- All decidable languages are recursive lang. & vice-versa

☆ Partially decidable Lang
- ↳ A lang. is partially decidable if it is a recursively enumerable lang.

☆ Undecidable Lang.
- It may sometimes be Partially decidable but not decidable
- If a lang. is not even partially decidable, then there exists no TM for that lang.

Transition function

1) NTM $\quad Q \times \Sigma \to P\{\Gamma \times (L,R) \times Q\}$
↳ Proper Set.

2) DTM
$Q \times \Sigma \to \Gamma \times (R|L) \times Q$

☆ **Decidability Problems**

→ A problem is decidable if we can construct a Turing Machine which will halt in finite amount of time for every I/p & give ans. as 'yes' or 'no'.

→ It has an algorithm to determine the answer for a given I/p.

eg) i) equivalence of RL
2) whether 2 RL are finite or not?

Undecidability Prob.

→ If there is no Turing Machine which will always halt in finite amount of time to give answer as 'yes' or 'no'

→ It has no algo.

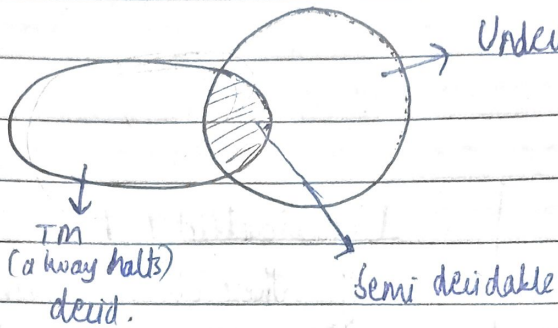
eg) i) Ambiguous of CFL ?
2) Equal or not (CFL)

→ Examples in the Decid./Under. table

		Reg.	DCFL	CFL	CSL	REC	RE.
1)*	Membership Problem $w \in \Sigma^*$	✓	✓	✓	✓	✓	X
2)*	Infiniteness Problem L = infinite or finite.	✓	✓	✓	X	X	X
3)*	Emptiness Prob. $L = \phi$	✓	✓	✓	X	X	X
4)*	Equality Prob. $L_1 = L_2$	✓	✓	X	X	X	X
5)	L is Ambiguous?	✓	✓	X	X	X	X
6)	Completeness $L = \Sigma^*$	✓	✓	X	X	X	X
7)	$L_1 \cap L_2 = \phi$	✓	X	X	X	X	X
8)	if $L_1 \subseteq L_2$ Subset Prob.	✓	X	X	X	X	X

★ Semi-decidable

- Always halt if ans. is 'Yes'
- May or may not halt if ans is 'No'



★ Undecidable Problems (for which algo. doesn't exist)
(recursively unsolvable)

1) Halting Problem

Can we design a machine which if given a program can find out or decide if that program will always halt or not halt on a particular I/P?

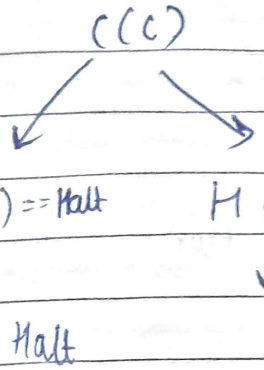
- Let us assume that we can design such machine:

H (P , I)
→ Halt
→ Not Halt.

- This allows us to write a program.

C (X)
if { H (X, X) == Halt }
Loop forever;
else
Halt Return;

Now, if we run 'C' on itself:



This shows that our assumption was wrong & hence we cannot design such a machine.

2) Post Correspondence Problem (PCP)

→ The PCP helps in describing the undecidability of the string

The PCP consists of 2 lists of strings that are of equal length over the I/p. The 2 lists are:

$A = w_1, w_2, w_3, --, w_n$ and $B = x_1, x_2, x_3, --, x_n$

then, there exist a non empty set of integers i_1, i_2, i_3 such that

$$w_1, w_2, w_3, -- , w_n = x_2, x_2, x_3, --, x_n$$

To solve PCP, we will try all combinations of i_1, i_2, i_3 ---- to find $w_i = x_i$. Then, we can say that PCP has a soln. Otherwise it does not have a soln.

$X_1 \quad X_2 \quad X_3$

a) $M = (ab, bab, bbaaa)$ have a PC soln. ?

$N = (\underset{Y_1}{a}, \underset{Y_2}{ba}, \underset{Y_3}{bab})$

for $X_2X_1X_3 \; \Delta \; Y_2Y_1Y_3$

$X_2X_1X_3 = babbabbbaaa$
$Y_2Y_1Y_3 = babbab$
] not equal.

There is no soln $|X_2X_1X_3 \neq Y_2Y_1Y_3|$ ∴, The PCP is undecidable

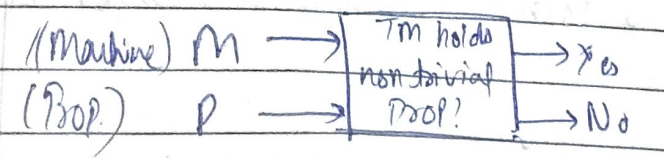
☆ Rice Theorem

- Trivial Property - Property that holds for every machine or holds for none is trivial. Rice's theorem doesnot apply to such a Prop. eg) Lang. of machines $\subset \Sigma^*$
- Non-trivial Prop. - Prop. that is neither true nor false for every computable function. eg) lang. of machine $= \Sigma^*$

Statement

↳ Any Non trivial Property about the language recognised by a Turing Machine is undecidable.

- There is no machine that can always decide whether the lang. of a given Turing Machine holds a particular non trivial Prop.



(Proof → have to ask whether in syllabus or not)

→ If P is a non-trivial Prop, & lang. holding the prop, L_P is recognised by TM M, then: $L_P = \{\langle M \rangle \mid L(M) \in P\}$ is undecidable

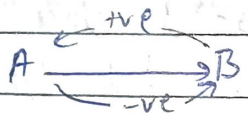
✡ Reducibility

↳ A reduction is a way of converting one problem to another problem, so that the soln to the 2nd prob. can be used to solve the first problem.

eg) finding area of rect. reduces to measuring its width & height

- If A reduces to B, you can use a soln to B to solve A.

$A \longrightarrow B$
$A \leq B$
$A \leq_m B$
] A is reducible to B



- if A is undecidable then, B is also undecidable
- if B is decidable then, A is also decidable.

✡ Universal Turing Machine (from aakash)

Unit-4

Algorithm

- Polynomial Time
  - Linear Search $(O(n))$
  - Bin. " $(O(\log n))$
  - Insertion $(O(n^2))$
- Non-Polynomial Time (exponential Time)
  - Su-Do-Ku (2^n)
  - Scheduling (2^n)
  - Graph colouring (2^n)

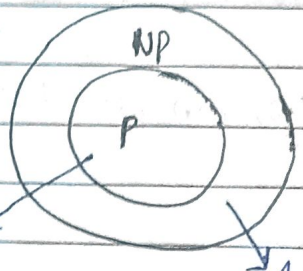
P → NP → Co-NP
Y/N (Yes) (No)

CLASSTIME Pg. No.
Date / /

- **P class Problem**
  - ↳ which can be solved in Polynomial Time eg) Sorting, Searching.
  - ↳ Sol. is easy to find. eg) finding GCD.

- **NP class Problem** (Non Deterministic Polynomial ~~Time~~ class)
  - ↳ which can be solved in Polynomial Time but can be verified in Polynomial time eg) Su-do-ku.
  - ↳ Set of all decision Prob. whose 'Yes' answers is checkable in Polynomial time.

$P \subseteq NP$



Tractable Problems (Solved in theory as well as in Practice)

Intractable Problems (Solved in theory but not in Practice)

→ The soln of NP class are hard to find since they are being solved by a non-determ. machine but the soln are easy to verify.

→ verified by TM in Polynomial time.

eg)- Boolean Satisfiability Prob. (SAT)
- Hamiltonian Path Prob.

- **Co-NP** (complement of NP class)
  - ↳ Set of all decision Problems whose 'No' ans. is checkable in Polynomial-time

If Prob. $X \rightarrow NP$
then X' (complement) $\rightarrow$ Co NP

eg) To check Prime No, Integer factorization.

• NP Hard Problems are as hard as NP Complete prob.

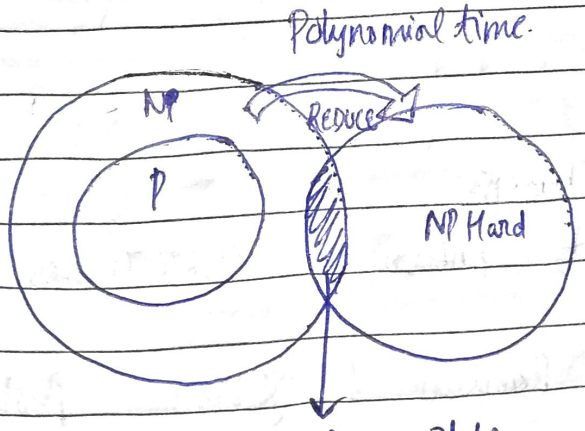
CLASSTIME | Pg. No.
Date / /

NP hard

- A problem is NP hard if every problem in NP can be polynomially reduced to it.
- Not a decision problem.
- Not all NP-hard prob. are np complete.
- It is optimization Problem.
- Eg → Halting Problem, shortest Path algo.

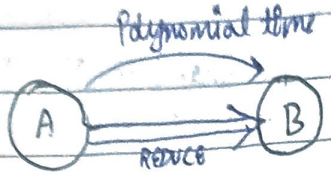
NP Complete

- A problem is NP complete if it is in NP & it is in NP Hard.
- NP Complete Prob. can be solved by a non-deterministic Algorithm / Turing Machine in polynomial Time.
- Decision problem
- All np-complete probs. are np-hard.
- Eg → Euler graphs, boolean formula is satisfiable or not. (SAT), Hamiltonian Path



Polyn. time
A ∝ B
(A is reduced to B)

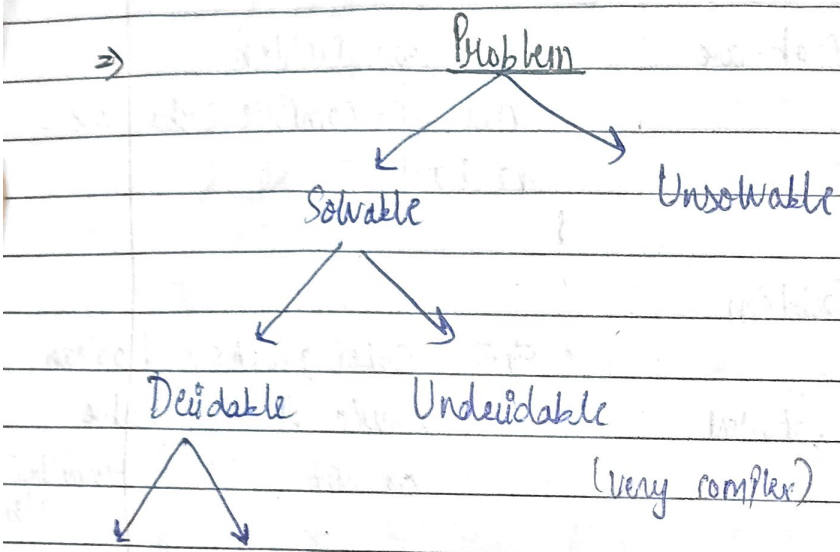
☆ Reduction



$A \propto B$ (A is reducible to B)

Problem 'A' reduces to Problem 'B' iff there is a way to solve 'A' by deterministic Algo. that solve 'B' in Poly. time.

⇒



☆ Cook – Levin Theorem

↳ It is also ~~Kno~~ known as Cook's Theorem.

~~Statement~~- It states that Boolean Satisfiability Problem is NP-complete.

- Any Problem in NP can be reduced in polynomial time by a Nondeterministic Turing Machine to the Boolean Satisfiability Problem.

CLASSTIME Pg. No.
Date / /

- SAT Problem
  - Boolean Satisfiability Problem
  - → This problem determines if there exists a set of Boolean variables which satisfy a Boolean formula.

I/P ⇒ A boolean formula

O/P ⇒ find for which combination of variables, the boolean formula becomes True.

Eg) SAT Problem.

P	Q	P∧Q	~(P∧Q)
T	T	T	F
T	F	F	T
F	T	F	T
F	F	F	T

if not any 'T' then, Not SAT Problem

Boolean formula ~(P∧Q) is satisfiable for

P=F and Q=F

P=F and Q=T

P=T and Q=F

Reduction in NP Complete

- A is reducible to B in polynomial time then B is NP-Hard
- If B is NP, then B is NP-complete.

~~Steps to prove NP-complete~~

☆ Space Complexity

(SPACE) • DSPACE (S(n)) ⇒ { L | L is decidable by a DTM in S(n) space }

• NSPACE (S(n)) ⇒ { L | L is decidable by NTM in S(n) space }

PSPACE (Space Poly(n))	NSPACE
↳ Set of languages decidable by a DTM in Polynomial ~~time~~ Space	↳ Set of languages decidable by a NTM in Polynomial ~~time~~ Space
$\bigcup_{k=1}^{k} DSPACE(n^k)$	$\bigcup_{k=1}^{k} NSPACE(n^k)$

Measuring Tape - TM uses at most S(n) cells in total on it's worktapes.

☆ Savitch's Theorem

↳ This theorem states that any NTM can be simulated by a DTM with at most a quadratic increase in the amount of space required.

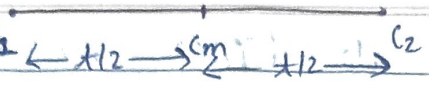
$$NSPACE(S(n)) \subseteq DSPACE(S^2(n))$$

Simulating a space bounded NTM can be accomplished by searching for a path from C_{start} to C_{accept}

NTM: $C_1 \rightarrow C_2$: S(n)

DTM: $C_1 \underbrace{\quad\quad}_{t} C_2$ yieldable ?

$t = 2^{S(n)}$ configuration



$C_1 = C_{start}$

$C_2 = C_{accept}$

C_m = Intermediate State.

canyield (C_1, C_2, t)

if $t = 1$ (stopping statement for recursion)

either $C_1 = C_2$ or C_2 should be reached from C_1 in 1 step

then accept else reject

- canyield (C_1, C_m, $t/2$) [recursive calls]
  canyield (C_m, C_2, $t/2$)
  if both functions accept then
  accept
  else
  reject

$$\frac{t}{2^0} \rightarrow t/\frac{1}{2} \rightarrow t/\frac{1}{2^2} \rightarrow t/\frac{1}{2^3} \cdots\cdots$$

$$t/2^i = 1$$

$$t = 2^i$$

$$\boxed{\log t = i}$$

Now, depth of recursion $\Rightarrow$ $i = \log t$

$t = 2^{S(n)}$

$\log t = \log_2 2^{S(n)} = S(n)$ (depth of recursion)

$$\boxed{S(n) * S(n) = S^2(n)}$$

↓ Parameters ↓ depth of recursion.

Hence, Proved

↳ Bcz of Savitch's Theorem

CLASSTIME | Pg. No. | Date / /

* NSPACE is equivalent to PSPACE bcz a DTM can simulate a NTM without needing much more space (even though it may use much more time)

* Probabilistic Computation.

- A complexity class is a set of problems that can be solved with a quantum TM in polynomial time
- Most commonly used Prob → decision Prob.

The ability to make probabilistic decisions often helps algorithms solve problems more efficiently.

- O/P depends not only on x (input) but also on a random variable(s) & a probability dist^n(s).

↳ (Bounded - error Probabilistic Polynomial Time)

BPP - Class of lang(s) recognized by a Probabilistic TM in Polynomial time with an error probability of 1/3.

BPL - Same as BPP but it includes a restriction that languages must be solvable in logarithmic space.

BPR - (Randomized Probabilistic Polynomial time)

ZPP - (Zero error probabilistic Polynomial time)