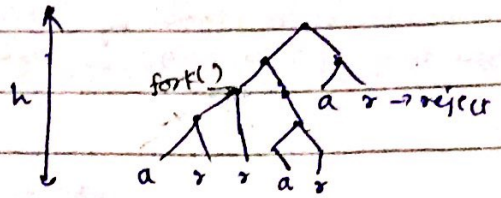
20/2/19

DATE: / /
PAGE NO.:

Non-deterministic TM (M)

→ It works like a tree:



* Since we're talking about decider, height of this will be finite
↓
has to halt (either accept or reject)

→ BFS / DFS : will run exponentially. always

We define M(w) which runs in polynomial time
↓
depends only on height of tree
⇓
have to consider only branch of height h. (not other branches)

↳ We can also see it as:

fork():
1 TM is starting 2 more TMs.

So, multiple TM are running parallelly
& finally, every TM has to end with 'a' or 'r'.

M(w):

∃ a branch whose leaf is a → accept → NTIME(n^i) $i \geq 0$ (many options) $\in$ NP

∀ branch leaf r → reject → Co-NP

here, h = height of branch

[if L(M) $\in$ NP
  L(M) $\in$ Co-NP]

Ex. HAMPATH = { <G, s, t> | There is a Hamiltonian Path from s to t in G }

We have to design the non-deterministic TM for this.

Mid-Term, Quiz: Till Kolomgrov,

Godel No: NPTEL Anuradha Singh

Mathematical Logic: lec-42

DATE: / /

PAGE NO.:

- If a path from s to t': we'll try to guess that path.

NTM {

1. guess $p_1 p_2 \dots p_m$ is a permutation of vertices in G. ($p_i \in V[G]$)
2. if they have covered all vertices v in V → accept
   if not → reject
3. No repeated vertices
4. $p_1 = s$, $p_m = t$
5. Check if p_i is connected to p_{i+1} i.e., $(p_i, p_{i+1}) \in E((G))$

}

guess: can use randomness if we introduce sample space S:

S = {all permutations of V}

- We can now redefine def^n of NP

NP = M(w, c) → (guess)

c → Certificate: with its help, we can find whether w belongs to language or not

- In above example,

$w \in$ HAMPATH

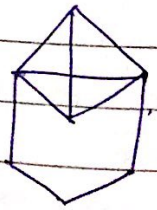
c = guess → it will check if $w \in$ HAMPATH: accept else: reject

|c| : must run in polynomial time

so, $\phi \cdot c = O(p(|w|))$

Ex. Clique = {(G, k) : G has a clique of size k or subgraph of K_k}

Ex.:

G = 

(G, 4) = True (we have subgraph of 4 vertices which is connected.)

(G, 1) = always true

(G, 2) = True (if there exists an edge)

DATE : / /
PAGE NO. :

Here, we will use 2-input NTM (earlier : used 1-input ...)
↓
call Verifier

Verifier (w, c) {
with help of c, check if: w ∈ L (Verifier)
}

- We have to find if verifier is equivalent to NTM.
If it is true, then we can use verifier instead of NTM.

NTM M(w) ≡ V (w, c)

Proof : NTM M(w) ⟶ V(w, c)

let L(M) : language accepted by M(w)
all language accepted by L(M) should be accepted by V.

We know, whatever guess we're doing in M, is passed in c
so, M(w) ⟶ V(w, c)

M(w) {
guess a string which allows to accept w
V (w, c);
}

V(w, c) ⟶ M(w)
↓
- Pass certificate as guess & run M on c and w
↓
only input : w

V(w, c) {
check w ∈ L(Verifier)
Run M on c and w
}

DATE : / /
PAGE NO. :

Ex. ϕ = { set of all satisfiable logical formulas }

NTM {
guess a satisfying assignment
check
}

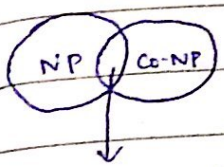
→ Polynomial Time
⇓
NP

$\phi \in$ NP

Ex $\bar{\phi}$ = { unsatisfiable logical formula } : can't do from 1 guess ⇓ have to reject all cases

$\bar{\phi} \in$ Co-NP

B.



NP $\stackrel{?}{=}$ Co-NP : Conjecture

$L \in NP \cap Co\text{-}NP$

Prime = { x is Prime } $\in P$: Before this proof, it was proved that Prime $\in$ NP ∩ Co-NP

(1) (2)

if it checks all the possible factors → it definitely $\in$ Co-NP.

for NP : have to find a certificate c

(P, C) → Pratt certificate : using this, we can find in polynomial time

↳ Format Little Theorem

Still, Time complexity of (1) is > (2)

→ Format little Theorem :

if gcd (a, n) = 1

then $a^{n-1} \pmod n = 1$ (or nice versa)

08/03/19

DATE : / /
PAGE NO. :

M(w) — ∃ a branch on simulation tree which ends at accept

∀ branches M ends in reject state → reject

↑
Co-NP

$$P = \bigcup_{i \geq 0} TIME(n^i)$$

$$NP = \bigcup_{i \geq 0} NTIME(n^i)$$

Co-NP := L in Co-NP ⇔ $\bar{L}$ in NP

NTIME(n^i) — define in terms of one input NDTM
" " " " two " "

$$EXP = \bigcup_{i \geq 0} TIME(2^{n^i})$$ ← size of input

$$NEXP = \bigcup_{i \geq 0} NTIME(2^{n^i})$$

4) Path = { (G, s, t) | ∃ a path from s to t in G }
↓
graph

↳ belongs to class P since linear time O(V+E)
↓
linear

$$|C| = O(|w|^i)$$

DATE : / /
PAGE NO. :

V(w, c) → accept / Reject
↳ certificate

Time complexity of V ⇒ $O(|w|^k)$

⇒ This language can be accepted in linear time by V.

HAMPATH

w will be G, S, t

C will be $p_1, p_2, \ldots p_n$

HAMPATH = { (G, S, t)

eg: Factor = { (x, y) | ∃ a factor of x less than y }

↳ this language belongs to NP.

Given a verifier such that V(<x, y>, c) provide c as a no. less than y.

eg Prime = { n | n is a prime no. }

Pratt Certificate = c

V(n, c)

given a no. check whether it is a prime

↳ NP.

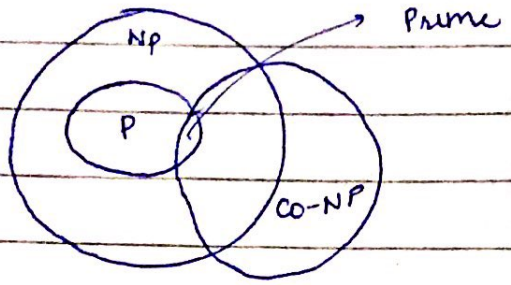
Prime ∈ NP by giving Pratt Certificate.

Co - NP , n is not prime

∴ Prime ∈ NP

* Prime belongs to NP and also co - NP

* AKS Algorithm



Theorem : $P = NP \implies EXP = NEXP$

Assumption : $P = NP$

Pick a language in NEXP, increase the size of input, running time same, then the class will change

$$NEXP = \bigcup_{i \geq 0} NTIME(2^{n^i})$$

↓

This does not change.

But $n \uparrow$

In terms of size of input, this will be linear

$M(n)$

↓

2^{n^i}

DATE : / /
PAGE NO. :

Proof:

$L \in NTIME(2^{n^i})$

$L' = \{ x \cdot 1^{2^{|x|^i}} \mid x \in L \}$

M will take input y : $M(y)$ and y should be of form

$y = z^{2^{|z|^c}}$ → no : reject

→ yes : → remove $1^{2^{|z|^c}}$

→ run M on z upto $2^{|z|^c}$ times

Check whether checking can be done in less than $2^{|z|^c}$.

→ Exponential's exponential can be done in polynomial time.

$n = 10$ $\quad a \cdot a^2 \cdot \ldots \cdot a^{\log_2 n}$

↳ multiply $a^2 \cdot a^8$

converting a into binary

∴ L' will belong to NP

$L \in NEXP$

↓

$L \in NP \Rightarrow L' \in P$

↓

$L \in EXP$

13/03/19

PAGE NO.:

Polynomial Time Reduction

Let f be a polynomial time computable func^n.
↳ Takes a T.M., takes w as i/p, runs f, halts

Consider language A & B:

$w \in A \Leftrightarrow f(w) \in B$ (if above cond^n is true)

denoted by $A \leq_p B$ (Reduce lang A to lang B in polynomial time)

↳ we want this polynomial time red^n because we want to work with P, NP classes.

- If we have EXP (exponential) / NEXP classes ⇒ Restriction on f would be: polynomial time red^n only (not exponential time red^n)
  [We don't want to ↑ hardness of the problem]

↳ Suppose, we define a new complexity class:

$$LT = \bigcup_{i \geq 0} DTIME((\log n)^i)$$

$$NLT = \bigcup_{i \geq 0} NTIME((\log n)^i)$$ → accepted by Non-Det TM which runs in logarithmic time

⇒ $LT, NLT \subseteq P, NP \subseteq EXP/NEXP$. (P class: n time)

↓

smaller than

P class

⇒ $A \leq_p B$ → can have smaller red^n than this in case of LT / NLT classes.

Definitions :-

↳ SAT = $\{ \langle \phi \rangle \mid \phi$ is a satisfiable logical formula $\}$

DATE : / /
PAGE NO. :

$Q = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_3 \wedge x_2)$

$1 = 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \rightarrow$ True

- Q ∈ this set.

- we've already shown that SAT ∈ NP

↳ $\underline{k\text{-SAT}} = \{ \bigwedge_{b=1}^{m} C_b \mid m \in \mathbb{N} \}$

$C_i = \left(\bigvee_{j=1}^{k} a_{ij} \right)$

$k\text{-SAT} = (a_{11} \vee a_{12} \vee a_{13} \ldots \vee a_{1k}) \wedge (a_{21} \vee a_{22} \ldots a_{2k}) \wedge \ldots$

- 3-SAT ∈ NP

Ex. $(x_1 \vee x_2 \vee x_1) \wedge (x_2 \vee x_3 \vee x_1)$ (should be 3; OR → 3, AND → any no. of times)

Ex. $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_2 \vee \bar{x}_3)$

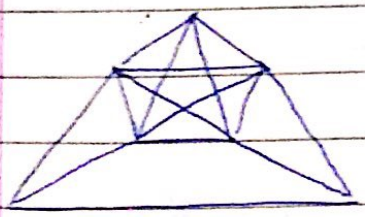
Read proof

- 2-SAT ∈ NP as well as P.

↳ $\underline{\text{clique}} = \{ \langle G, k \rangle \mid G \text{ has a clique of size } k \}$

↓

G has a sub-graph of K_k → complete graph of size k.



(G,1) ✓ (G,3) ✓ (G,5) ✓

(G,2) ✓ (G,4) ✓ (G,6) ✗

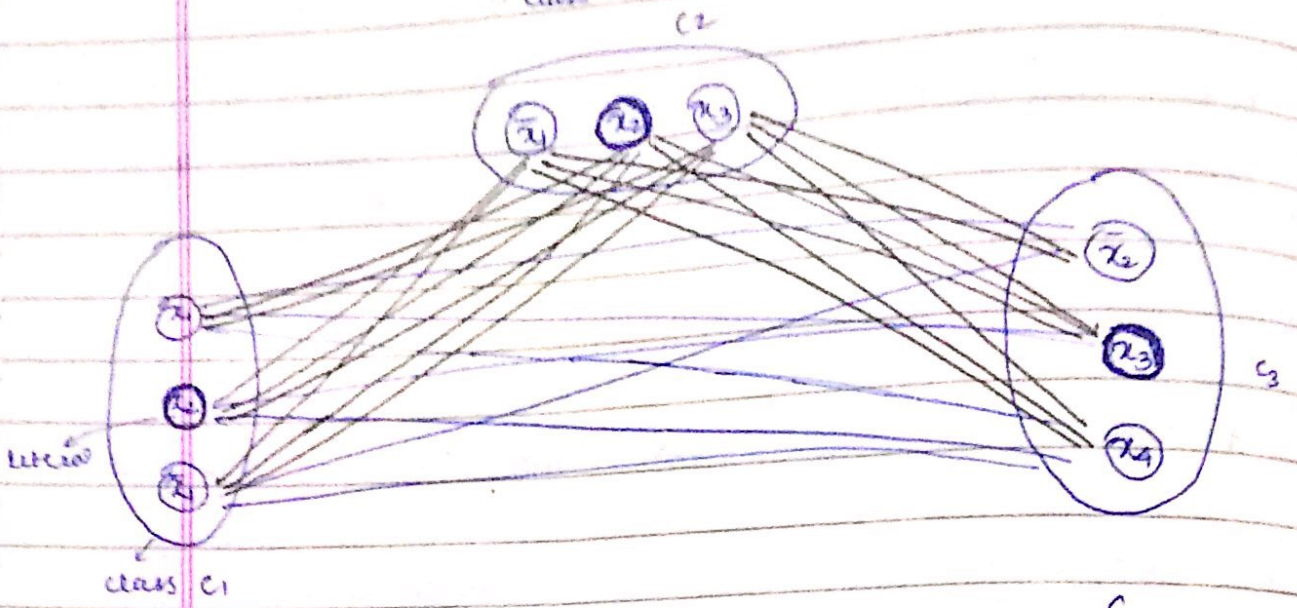
digital logic , graph theory

Ex 1) Reduce 3-SAT into clique

3-SAT $\leq_p$ Clique

literals

$\phi = \underbrace{(x_1 \vee x_2 \vee \bar{x}_1)}_{\text{class}} \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4)$



→ Connection strategy:

1. x and $\bar{x}$ won't be connected (Only 1 of them will be true)
2. Within a class, there will be no connection

- To prove :
  1.) ϕ is satisfiable
  2.) G has 3 clique

(1) If ϕ is satisfiable : 1 literal is atleast true in all 3 clauses choose vertex acc.

C_1: suppose $x_2 = 1$ → choose x_2 in C1

C_2: $x_2 = 1$ → choose x_2 in C_2 & similarly for C_3

3-SAT ∈ NP-complete

DATE : / /
PAGE NO. :

So, these 3 nodes must be connected in a triad

" ϕ is satisfiable (atleast 1 in each class is true)
(choose only True assigned variables)

2) G has 3 clique → get sub-graph of size 3

have 3 classes : each node in a class is connected to other in other classes → 3-clique

If we've k-SAT → k-clique

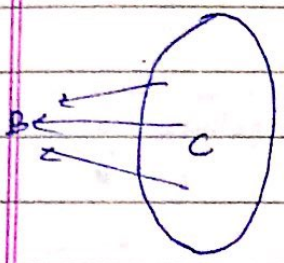
2) Clique $\leq_p$ 3-SAT : Reduce clique to 3-SAT

Can choose a set of vertices acc. to 2 cond's given. It will satisfy ϕ.

→ C-Complete : C = P / NP / EXP / NEXP...

1. $\forall A \in C \quad A \leq_p B$ ⇒ B is C-Hard (1. Only)

2. $B \in C$. ⇒ B is C-complete (1. & 2.)



If all problems of B are reduced to C: B = NP Hard

↓ & if $B \in C$: B is NP complete

15-03-19

DATE : / /
PAGE NO. :

→ A language is NP-Complete if it satisfies the following cond^{n}s:

1. $L \in NP$
2. $\forall L' \in NP, \; L' \leq_p L$

Properties :

→ If L is NP-Complete & $L \in P$, then $P = NP$ [$P \subset NP$]

↓

If I can solve L in Polynomial time, I " " all other NP in " "
as all can be reduced to L.

* → L is NP-Complete and $L \leq_p L''$ and $L'' \in NP$, then L'' is NP-Complete

$L \in NP, \; \forall L' \in NP, \; L' \leq_p L \leq_p L'' \; \& \; L'' \in NP$

$\rightarrow \forall L' \in NP, \; L' \leq_p L''$

Ex: SAT is NP-Complete. Prove. (similar to PCP) → NPC

Soln: We already know : $SAT \in NP$.

We've to prove : $\forall L' \in NP, \; L' \leq_p SAT$

↳ instead of dominos we'll make boolean expression here.

Take a T.M. (NDTM) 'M' st:

M(w) convert → SAT formula (in polynomial time, M must run in $O(n^k)$)

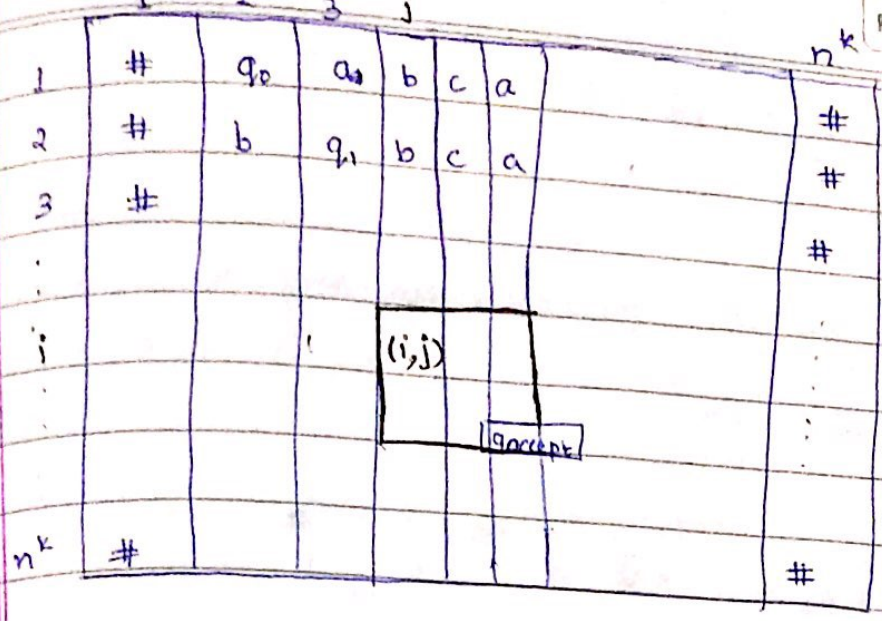
runs on w & exec^n will convert to SAT formula.

NDTM M on i/p w run $O(n^k)$ time & halt

$M \leq_p SAT$

→ In the 2-D Matrix : 1st & last col contains #.

1st row : contains 1st ~~representation of input~~ config^n (q0w) → abca



→ taken $(n^k \times n^k)$ because NDTM runs in $O(n^k)$

↳ suppose, we've

$\delta(q_0, a) \rightarrow (q_1, b, R)$

→ If we get q_{accept} / q_{reject} somewhere · we're done & give o/p accordingly

↳ This arrangement we will try to model into SAT

↳ How to convert ?

Possible value at a cell : $Q \cup \Gamma \cup \{\#\}$

But we've to put only 1 symbol at a particular position.
(or only 1 symbol) ↓ using ∧ / ∨ ..

a	b	c	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

→ we've 3 input & want to on exactly 1.

a \ bc	00	01	11	10
0	0	1	0	1
1	1	0	0	0

$= \bar{a}\bar{b}c + \bar{a}$

$= \bar{a}\,\bar{b}c + a\,\bar{b}\bar{c} + \bar{a}bc$

$0 = (\bar{a}\bar{b}\bar{c}) \vee (bc) \vee (ac) \vee (ab)$

$0 \rightarrow (a \vee b \vee c) \wedge (\bar{a} \vee b \vee c) \wedge (a \vee \bar{c}) \wedge (a \vee b)$

Sipser : $(a \vee b \vee c) \wedge (\bar{a} \wedge \bar{b}) \wedge (\bar{a} \wedge \bar{c}) \wedge (\bar{b} \wedge \bar{c})$

(min 0 in k-map & complement it)

→ This formula we can generalize for any no of variables

$a_1, a_2, \ldots a_{|c|}$

$$\bigvee_{i=1}^{|c|} (a_1 \wedge a_2 \wedge \ldots \wedge \bar{a}_i \wedge \ldots a_{|c|})$$

↓ take 1 complement at each time.

→ We'll have to do it for all the cells.

$x_{i,j,s}$ = 1 / 0 ($x_{i,j,s}$: boolean variable)

↓ ↳ -nt

+nt

s is present at position (i,j) (or not)

$C = \Gamma \cup Q \cup \{\#\}$

$$\phi_3 = \bigvee_{s=1}^{|c|} (x_{i,j,a_1} \wedge x_{i,j,a_2} \ldots \wedge \bar{x}_{i,j,a_s} \ldots \wedge x_{i,j,a_{|c|}})$$

$1 \le i, j \le n^k$ ⇓

for all cells. This shows we've exactly 1 symbol is +nt at each cell (i,j)

* We've to ensure there is no illegal move, when do we reach reject / accept state.

→ for the 1st row, let $w = w_1 w_2 \ldots w_n$

$$\phi_s = x_{0,0,q_0} \wedge x_{0,1,w_1} \wedge x_{0,2,w_2} \ldots \wedge x_{0,n,w_n} \wedge x_{0,n+1,\sqcup} \ldots \wedge x_{0,n^k-1,\sqcup}$$

(blank)

ϕ_1 $\triangleq$ each cell of $n^k \times n^k$ grid contains exactly 1 symbol from set C

ϕ_{accept} $\triangleq$ T a cell in $n^k \times n^k$ grid which contains q_{accept}

DATE : / /
PAGE NO. :

ϕ_{accept} . $\phi_{legal\ move}$

final answer: all ϕ_i must be true

$\phi = \phi_1 \wedge \phi_c \wedge \phi_{accept} \wedge \phi_{legal\ moves}$

ϕ_{accept} : if q_{accept} has appeared anywhere in the grid

$$\phi_{accept} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{accept}}$$

10.03.19

$\phi_{legalmoves} = ?$

We'll take a 2×3 grid & find legal moves for this grid. in figure

a	q_1	q_2
a B	b	c

Reject

$\delta(q_1, a) \rightarrow (q_2, b, L)$

	q_1	a	b
:	q_2	B C	b

→ legal wrt above transition

a	b	b
b	b	b

→ legal window

$\delta(q_1, a) \rightarrow (q_2, b, R)$

q_1	a	b
b	q_2	b

→ legal wdw

DATE : / /
PAGE NO. :

4. For each transition fun^n, there is a finite no of legal windows ($\ll |c|^6$)

generalise :

a_1	a_2	a_3
a_4	a_5	a_6

$$\phi_{\text{legal moves } 2\times 3} = \bigvee_{\substack{a_1,a_2,\ldots,a_6 \\ \text{such that} \\ \text{it is legal}}} \left(x_{i,j,a_1} \wedge x_{i,j+1,a_2} \cdots \wedge x_{i+1,j+2,a_6} \right)$$

For whole grid :

$$\phi_{\text{legal moves}} = \bigwedge_{1 \le i,j \le n^k} \left(\bigvee_{\substack{a_1,a_2,\ldots,a_6 \\ \text{such that} \\ \text{it is legal}}} \left(x_{i,j,a_1} \wedge x_{i,j+1,a_2} \cdots \wedge x_{i+1,j+2,a_6} \right) \right)$$

↓

taken 'AND' because each 3 move should be legal.

$$\boxed{\phi = \phi_1 + \phi_{\text{start}} + \phi_{\text{legal moves}} + \phi_{\text{accept}}}$$

- We must show that this conversion happens in ~~of~~ polynomial time.

$$\phi = \phi_1 + \phi_{\text{start}} + \phi_{\text{legal moves}} + \phi_{\text{accept}}$$

$O(n^{2k})$ — $O(n^{2k})$ — $O(n^k)$ — $O(n^{2k})$ — $O(n^{2k})$

$O(n^{2k})$ → polynomial time

$6 \times |c|^6 \times O(n^{2k})$ → at max

OR

AND

For T.M. : $|c|^6$ will be const.

DATE : / /
PAGE NO. :

Ex: 3-SAT is NP complete. Prove.

① Reduce SAT into 3-SAT SAT $\leq_p$ 3-SAT $\in$ NP

↳ whatever formula we get in M $\leq_p$ SAT, we'll try to reduce it to 3-SAT

3-SAT has : ~~OR of ANDs~~ AND of ORs

in req^n process, we got : ~~AND of ORs~~ OR of ANDs

↓ have to be converted into

AND of ~~ORs~~ ORs (use digital system concept)

↳ Another problem : → ①

we might get $(a_1 \vee a_2 \vee a_3 \vee \dots \vee a_4) \wedge (a_5 \vee a_6 \vee \dots \vee a_t)$: we want only 3 variables

$\Leftrightarrow$ $(a_1 \vee a_2 \vee z) \wedge (\bar{z} \vee a_3 \vee a_4)$

↓

can choose z s.t. when ① is true, this exp^n is true.

$(a_1 \vee a_2 \vee a_3 \vee a_4 \vee a_5)$

↓ 3-SAT

~~$(a_1 \vee a_2 \vee a_3 \vee z) \wedge (\bar{z} \vee \dots)$~~

$(a_1 \vee a_2 \vee z) \wedge (\bar{z} \vee a_3 \vee a_4 \vee a_5)$

↓

$(\bar{z} \vee a_3 \vee y) \wedge (\bar{y} \vee a_4 \vee a_5)$

↓

Can generalize this

∴ 3-SAT is also NP-complete.

3-SAT $\leq_P$ 2-SAT

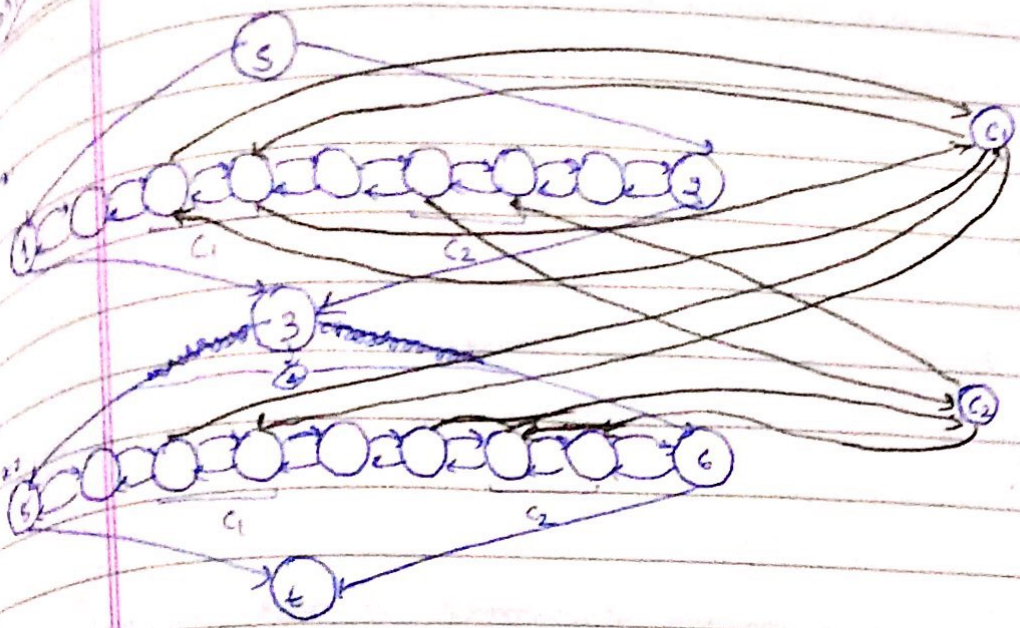
& 2-SAT $\in$ P $\rightarrow$ 3-SAT ??

an instance of SAT problem. Doesn't mean it is P. Only an instance is polynomial time solvable.

DATE : / /

PAGE NO. :

27/0



$$\phi = (x_1 \vee \bar{x}_1 \vee x_2) \wedge (x_1 \vee x_1 \vee x_2)$$
$$C_1 \qquad\qquad C_2$$

→ Create diamond structures for only variables, not their complements (not all literals)

↓

we are already managing $\bar{x}_1$ in x_1 only.

↙ $x_1 = T \qquad x_2 = F \Rightarrow \phi = T$ → satisfiable

want to get HAMPATH for this assignment

⇓

Start from s. ∴ $x_1 = T$: Take left move, go to 1 & move right, cover class C_1, ~~cover class C_2~~, move to 2, then 3, 4

∴ $x_2 = F$: ~~St~~Take right move, go to 6 & move left

↓

if left: won't be able to cover all the vertices.

↳ we can cover a class from any 1 variable only.

(Here, we've covered C_1 from x_1 only & not x_2)

(" " C_2 " x_2 " " " x_1)

DATE : / /
PAGE NO. :

→ From a satisfiable assignment ($\phi = T$), we can always get a HAMPATH.

• HAM $\leq_p$ 3-SAT Now, we've to prove this.

1st we've to rule out certain cases:

→ Can't go & come from 2 diff. diamond structures for a class.

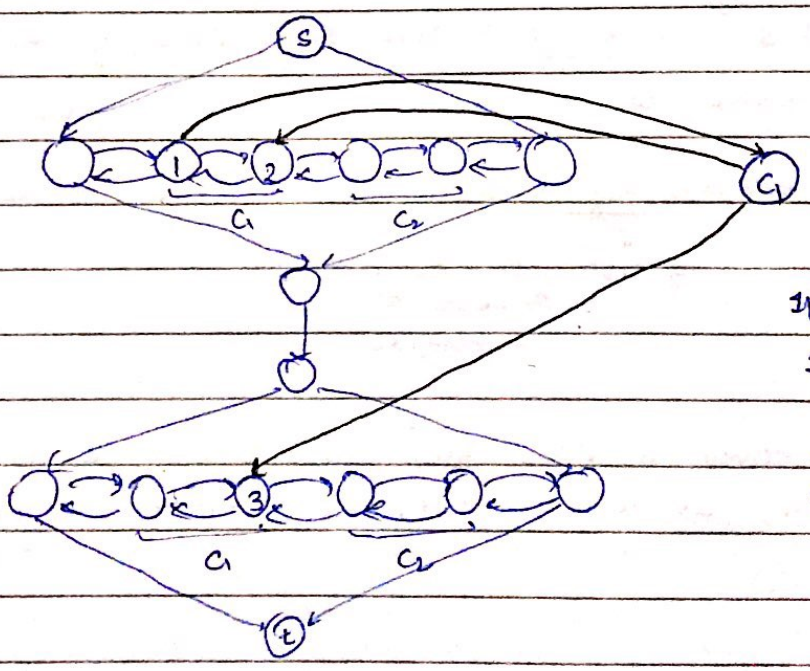
(same proof as called) ~~Suppose you've path as we discussed in last example.~~
Suppose from x_1 to $\bar{x}_1$, C_1 & return back to x_2

It means if you make x_1 variable True. ⇒ class C_1 will be true.

For st : HAMPATH → will have to cover each class node.

* It'll work even if you remove the boundary separator nodes.

↳ Separator nodes :→ 2 outgoing, 2 incoming edges
suppose we don't have any separator node.



If move from 1 to C_1 to 3
⇓
can't cover 2

DATE : / /
PAGE NO. :

14/02/19

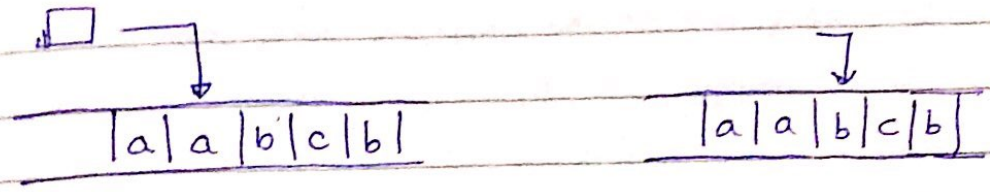
Section 1.7
Arora Barak

Thm: If a TM M halts on i/p x in time $O(T)$, then UTM U o/p $M(x)$ in $O(T \log T)$ time

Proof: UTM U is multitape

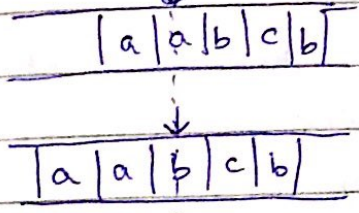
But we'll do it for just single tape $O(kT) \equiv O(T)$ only.

M :



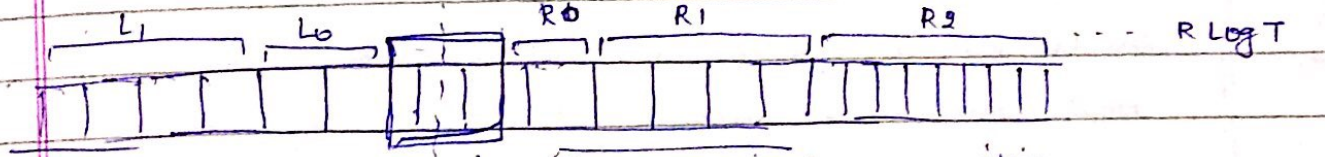
How to simulate this move in UTM U ?

U :



will shift whole tape ~~alphate~~ quantity to the left

By this, we'll maintain this position (unlike in TM M)



Suppose this is the position we're maintaining

divide this right side into different sized - zones

→ We'll add a new dummy symbol '#' now to fill pos^n's where letters are - nt.

→ We've to maintain some invariants while shifting.

R0 : 2^{0+1}

R1 : 2^{1+1}

R2 : 2^{2+1}

$2(2^i-1)$

DATE : / /
PAGE NO. :

Invariants

not in any zone

① Have to make middle box const. by shifting tape qty.

② Each zone must either empty, half full or full.

③ Total no of symbols in R_i. $L_i \cup R_i = 2^{i+1}$ (want to maintain this)

→ L_i : empty → R_i must be full

L_i : half full ⇒ R_i " " half full

L_i : ~~empty~~ full → R_i " " empty

→ How to shift?

Ⓐ Left shift :

Right

1. Find i^{th} i s.t. R_i is not empty (No. of symbols possible in R_i = 2^i or 2^{i+1} (empty positions)

2.

We'll shift symbols of R_i into $R_{i-1}, R_{i-2}, \ldots R_0$

R_i to R_{i-1} : are empty fully.

& we have atleast 2^i symbols in R_i.

2^i

We'll put 1 symbol in middle box

Now remaining symbols : $2^i - 1$

Now, we'll ~~(not)~~ keep $R_0, R_1, \ldots R_{i-1}$ all half full.

Total : $2 + 2^2 + \ldots + 2^i = 2(2^i - 1)$

& half full : $2^i - 1$ ⟸

R_i : empty

2^{i+1}

$2^i + 2^i$ ⇒ 2^i symbols will ~~move~~ shift left

& R_i will contain 2^i symbols

Ⓑ ~~Shift right~~

Left

L_i is not full. ($L_i \cup R_i = 2^{i+1}$) Since R_i is not empty, L_i is not full

Since $R_{i-1}, R_{i-2}, \ldots R_0$: all empty

⇒ $L_{i-1}, L_{i-2}, \ldots L_0$: all full

DATE : / /
PAGE NO. :

→ No. of symbols in $L_{i-1}, L_{i-2}, \dots L_0$: $2(2^i - 1)$

" " L_i = 2^i (half full) or 0 (empty)

vaccancy in L_i : 2^i

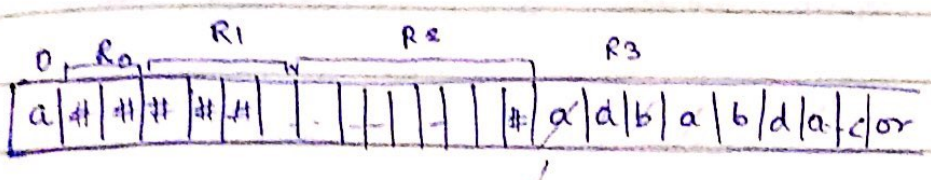
We'll shift half of $L_{i-1}, L_{i-2}, \dots L_0$ each to L_i

$\Rightarrow 2^i - 1$

Also, element at 0th posin (middle box) will also be shifted -1

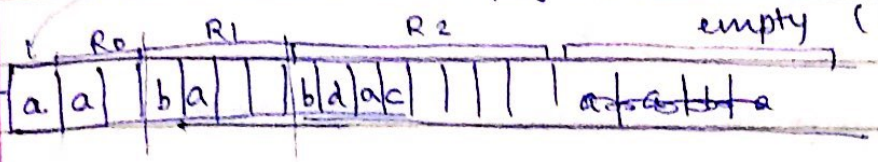
So, L_i will be either full or half full.

Ex



Left shift :

R_3 : 1st non-empty zone



Similarly, we'll do for left side also.

single tape *

a b c a — have to shift whole string → Time needed = $O(|w|^2)$

If have multi tape

a b c a — just copy paste → $O(|w|) + O(|w|)$ = $O(|w|)$

→ Task of shifting : $O(|R_0| + |R_1| + \dots + |R_i|)$

$O(2^{i+1})$

If we have left shift & R_3 : non-empty

↓ after left shift oper^n

all R_0, R_1, R_2 : non-empty

→ if we make next ~~shift~~, $2^i - 1$ shifts it will be ~~for~~ R_0. within range $i \in [0, 3)$

We're only shifting R_3 to L_3, not R_3 to $R_{\log T}$ or L_3 to $L_{\log T}$ ⇒ reducing time

→ it'll require atleast $2^i - 1$ symbols to make all empty ($R_{i-1}, R_{i-2}, \ldots R_0$)

R_i

TM M : 0 ... R_i ... T

left shift i upto $2^i - 1$ shifts, we're in range L_i to R_i

~~* if right shift comes, i will shift right~~

$$\leq \left(\frac{T}{2^i - 1}\right)$$

↳ atmost how much time it has gone to pos^n R_i

Cost of each shift = $O(2^i)$

→ Time complexity $\sum_{i=0}^{\log T} \left(\left(\frac{T}{2^i - 1}\right) O(2^i) \right)$

DATE : / /
PAGE NO. :

Time Constructible Function

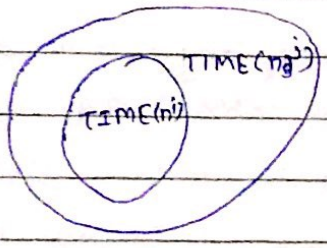
f is a time constructive func^n if
$\exists$ T.M. $M(1^n)$ ⟶ writes $1^{f(n)}$ in o/p tape in $O(f(n))$ time

→ Till now, whatever func^{n}s we've used are time constructible
$f(n) = n^2,\ 2^n,\ n\log_2 n$ $\quad [f(n) = \Theta n + \dots]$

Ex $f(n)$ = const.
$f(n) = 5$ (let) ⟶ not time constructible because TM M has to atleast read i/p 1^n (length → n). so, it can never be done in $O(5)$

* $\boxed{f(n) \geq n}$

Deterministic time hierarchy theorem :
TM M which halts in $O(n^i)$ time
① $TIME(n^i)$ ② $TIME(n^j)$
if $j > i$: then :



∃ language not decided by ①, but decided by ②

Proof : Will have to use a language not accepted (halted) by any TM inside smaller circle.
In case of halting : we did diagonalization
$M_1, M_2, M_3, \dots$ → which halts in $O(n^i)$ time

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	...
M_1			
M_2			
⋮			

→ complement these entries

Time complexity of U may be $> O(n^i)$ ∀

DATE : / /
PAGE NO. :

Take M(~~⟨M⟩~~) UTM U (M(⟨M⟩))
↳ take Universal TM to check if M accepts ⟨M⟩ or not.

o/p

opposite of M

→

Theorem: $TIME(f(n)) \subsetneq TIME(g(n))$

$o(g(n)) = f(n) \log f(n)$ → $\forall n > n_0$ — ①

small o

let $f(n) = n$

$o(g(n)) = n \underbrace{\log n}_{< n} = n \cdot n^{\epsilon}$, $0 < \epsilon < 1$

→ $TIME(n) \subsetneq TIME(n \cdot n^{\epsilon})$ ✓

→

D(w)

1. Put g(n) in tape ; g(n) is time constructible func^n
2. Check if $w = \langle M \rangle 10^{n_0}$
3. Run following steps $g(n)$ times. (Decrement $g(n)$ -- in 1st tape
4. UTM U (M, ⟨M⟩) o/p opposite → ~~o not given~~ give o/p as : (complement o/p of U)
   if it completes within g(n) time
   otherwise → give any o/p

diagonalization

D(w) : runs within g(n) time → TIME (g(n))

Also ④ → ⑧ by complementing, we're avoiding all TM which halts in O(f(n)) time.

→ We can compute time constructible func^n using other TM

D(w) : set difference of TIME(g(n)) - TIME(f(n))

DATE : / /
PAGE NO. :

$g(f(n))$

→ If a TM M does task of D(w) → pass M in U. in ④, then, o/p will be opp. of M

→ have to ensure D(w) halts (have otherwise course also) (within g(n) time)

UTM : c. $(f(n) \log f(n))$

We've to ensure

$$c \cdot (f(n) \log f(n)) < c' g(n) \quad — ②$$

Using ①, we'll ↑ size of i/p ($n \overset{\text{s.t.}}{>} n_0$)

UTM (M , $\langle M \rangle 10^{n_0}$)

If $n > n_0$, then ③ will always be correct.

⇒ Every simulation of TM D will be finished in $O(g(n))$ time

03/04/19

→ Today, we'll prove ND version of hierarchy theorem

Non-Deterministic time hierarchy theorem

Let g(n) be a time constructible func^n

$$f(n+1) \log f(n+1) = O(g(n))$$

Have to find a language L:

$$L \in NTIME(n^i) - NTIME(n^j) \qquad j > i$$

Problem : $L \in P \Rightarrow \bar{L} \in P$

But $L \in NP \Rightarrow \bar{L} \in NP$ Not true always

Prove: $NTIME(f(n)^{n+1}) \subsetneq NTIME(g(n))$

DATE : / /
PAGE NO. :

→ If i/p is itself exponentially large, & if we follow all branches (also done "), then it can be done polynomial time. But time isn't going to change. (Like we did in the case of NEXP proofs)

simulation halt

⟹

$P = NP \Rightarrow EXP = NEXP$

NDTM M accepts $L \in NEXP$

$$L' = \{ \langle x, 1^{2^{|x|^c}} \rangle ; x \in L \}$$

{
Check $y = x \mid 1^{2^{|x|^c}}$
Run $M(x)$
}

size of i/p $= O(2^{|x|^c})$

Running time of this $= O(2^{|x|^c})$

size of i/p is linear wrt running time

$L' \in NP \Rightarrow L' \in P \rightarrow L \in EXP$ ⇐ Overall polynomial time complexity

↓ measure on basis of size of i/p

⟹

for each TM, we're assigning a particular interval

NDTM : $M_k \quad (l_k, u_k]$ s.t. $u_k = 2^{l_k}$

↳ should be disjoint continuous at most
↳ next interval : atleast $(u_{k+1}, 2^{u_{k+1}}]$

$D(1^n)$ {

1. find k s.t. $n \in (l_k, u_k]$
2. if $n < u_k$, then simulate a Non deterministic UTM which takes i/p M_k & 1^{n+1}. & O/p accordingly upto $g(n)$ steps

Simulate TM M_k on i/p 1^{n+1}

To ensure it happens in $O(g(n))$ time.

3. if $n = u_k$, Deterministically run M_k on 1^{l_k+1} & O/p opposite

doing some diagonalization, that's why have complemented O/p → won't affect result as in Deterministic TM

This will run in polynomial time as size of i/p is exponential

if $L \in$ Class, then $\bar{L} \in$ Class.

DATE : / /
PAGE NO. :

→ Have to argue that all $L \in NTIME(f(n)) \notin D$

proof by contradiction ↳ Suppose $\exists$ a TM. M s.t.

$L(M) = L(D)$

using 3. ↓ and M runs in $O(f(n))$ time

doing complement here only ↓ So, for this TM M also, we'll assign a particular interval (l_k, u_k)

$L(M) = L(D)$

$\Rightarrow M(l_k+1) = D(l_k+1)$

$M(l_k+2) = D(l_k+2)$

$\vdots$ — ①

$M(u_k) = D(u_k)$

$M(u_k+1) = D(u_k+1)$

→ We're trying to compare behavior b/w M & D over given interval.

In step 2 : simulating M_k on next i/p → n+1

We want to prove : $M(l_k+1) = M(l_k+2) = \cdots = M(u_k)$

M_k : we can replace with M (can find an interval)

$\Rightarrow D(l_k+1) = M(l_k+2)$

$D(n) \rightarrow M(n+1)$

Similarly, $D(l_k+2) = M(l_k+3)$

$\vdots$ — ②

$D(u_k-1) = M(u_k)$

From ① & ②, we get

$D(u_k) = M(l_k+1)$

But in step 3, $D(u_k)$ = complement of $M(l_k+1)$

⇓

By contradiction, $\exists$ no such TM M.

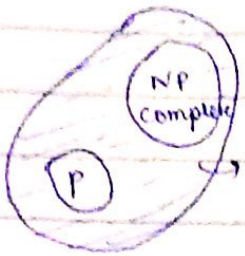
DATE : / /
PAGE NO. :

5/4/19

Ladner

Th^m : Assume $P \neq NP$ Then $\exists$ a language $L \in NP - P$ s.t. $L \notin NP\text{-complete}$

have to select a language $\in NP - P$ & $L \notin NP\text{-complete}$.



↳ we're talking about this region

M_i : enumeration all polynomial time DTM

f_i : " " " reduction func^n (mapping $\dots \leq_p \dots$)

✓ can enumerate because f_i is calculated by a T.M.

Define lang A s.t.

1. $A \neq L(M_i) \quad \forall M_i$
2. & $x \in NP\text{-C} \iff f_i(x) \in A \quad \forall f_i$ — x : NP-complete problem.
3. $A \in NP$ ↳ can choose any NP-C language (SAT)

$$A = \{ x \in SAT \mid f(|x|) \text{ is even} \}$$

($x \in SAT$: NP; $f(|x|)$ is even : ↳ If we get this also in polynomial time) $\rightarrow A \in NP$.

2 → $\forall\ x \in SAT \Rightarrow f_i(x) \notin A \quad \vee \quad x \notin SAT \Rightarrow f_i(x) \in A$

- Proving $f(|x|)$ runs in polynomial time (by induction)

1. $A \neq L(M_i)$ $f(0) = 1$ $f(1) = 2$

$f(n) = 2i$ check all x, $|x| < \log n$ s.t.

(a) $M_i(x)$ accept $x \notin SAT$ or (V) $f(|x|)$ is odd

+ run x on M_i ; + if accepts, we've to reject this for A

(b) $M_i(x)$ reject $x \in SAT$ V $f(|x|)$ is even

↳ If true : $f(n+1) = f(n) + 1$

else $f(n+1) = f(n)$

→ We're checking each x upto $\log n$ for each DTM M_i,

$|x| < \log n$ — x: binary no. — upper limit

∴ value of x : $2^x = 2^{\log n} = O(n)$ ✓

For even stage:

(a) & (b) : ensure that x is not accepted by a polynomial time DTM

↳ We're ↑ our n upto till we get (a) & (b) true

↓ We're finding largest n till which (a) & (b) conditions are not met.

2. Requirement $f(n) = 2i+1$ check all x, $|x| < \log n$ s.t.

(a) $x \in SAT$ then $f_i(x) \notin A$

(b) $x \notin SAT$ then $f_i(x) \in A$.

↳ If true : $f(n+1) = f(n) + 1$

else : $f(n+1) = f(n)$

At odd stage : trying to avoid all x for which there is no reduction from SAT. to A

Algo. runs like : $M_1, f_1, M_2, f_2, \ldots$

Algo. $M_i(x)$ → is P. / $x \in$ or not SAT → is NP → each step is either in P or in NP

∴ A is polynomial time solvable.

→ Can it happen that always $f(n+1) = f(n)$

if $f(n_0)$: even → always odd after $n > n_0$?

DATE : / /
PAGE NO. :

08/04/19

Let $f(n_0)$ = even & ~~after~~ $\forall\ n > n_0$, $f(n)$ = even

→ we are going to else part always.

Req 1.

→ ∄ a M_j which accepts A ⇒ P.

→ ∄ $L(M_j) = A$ after n_0.
↳ infinite

$\forall$

before n_0 : finite length → can hard code it → A is accepted in polynomial time DTM.
→ f is calculated " "
→ SAT is also " "
" " P = NP
↓
This is not true
(SAT ∉ P)

→ This case won't happen.

similarly for odd case.

→ $f(n_0)$ = odd & $\forall\ n > n_0$, $f(n)$ = odd

$f(n+1) = f(n)$

going to else part always

Req 2.

→ ∄ red^n from SAT to A after n_0

for x for which $f(|x|)$ = odd : it won't ∈ A
(def^n of A)

So, ∄ finitely many elements in A

⇒ A is solvable in P time. A ∈ P.

& we have red^n from NP-C problem to A.

SAT ∈ P → P = NP ✗

Req P : TM runs : $|x|^i = (\log n)^{f(n)} \leq n$
↑
max.

⇓

when f(n) will cross this limit, we'll apply it ~~to~~ in algo.

Graph isomerphic → not known ∈ NP-C or not.
∈ NP ✓

DATE : / /
PAGE NO. :

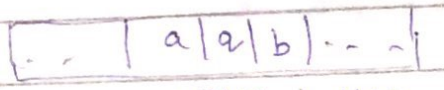
∴ f(n) won't be const (always odd or always even)
→ A ∉ P & A ∉ NP-C (last proof)

08/04/19

SPACE (f(n)) = { L | language L is accepted by DTM in O(f(n)) space }

NSPACE (f(n)) = " " " NDTM

→ When space is fixed, it becomes Linear Bound Automata.



→ configuration

No. of possible confg's when we have a DTM which halts:

$< 2^{O(f(n))}$

If we consider each confgⁿ as a vertex in the graph,

If we can get C_j from C_i using some δ of T.M. → connected
else → not

Configuration graph.

$C_0 = q_0 w$ $\quad C_{accept} = w' q_{accept} w''$

decidable language → If we get path from C_0 to C_{accept} → T.M. accepts L
else → rejects

Ex. SAT = { ϕ : Satisfiable logical Formula } → not decidable in poly. time, but decidable in poly. space

variables : $x_1 x_2 \ldots x_m$; SPACE(n^i) i>0
O(m) space

have to do in O(m) space. How?
→ Have to generate particular assignment of variables.

DATE : / /
PAGE NO. :

Ex. x_1 x_2 x_3

Then for each assignment, we'll have to check whether it is satisfiable logical formula or not

$|\phi| \geq O(m)$ → possible

so, instead of $O(m)$, we'll go with $O(|\phi|)$

ND : 3. PATH = $\{ \langle G, s, t \rangle \mid \exists$ a path from s to t in G $\}$

Ignore size of input (G). (It'll take $O(|V|^2)$)

s ——— t space taken = $O(|V| \log_2 |V|)$

$|V|$ → no. of vertices; $\log_2 |V|$ → space taken by each vertex

↓ can reduce further upto $O(\log_2 |V|)$

~~if we ignore~~

~~whatever path you've guessed~~

$\log_2 |V|$ → to store a vertex

[s] → ~~two~~ guess vertex connected with s.

[x]

→ PSPACE = $\bigcup_{i>0}$ SPACE (n^i)

NPSPACE = $\bigcup_{i>0}$ NSPACE (n^i)

We'll prove that : PSPACE = NPSPACE

4. PATH = $\{ \langle G, s, t \rangle \mid \exists$ a path from s to t in G $\}$

Design a deterministic algo for this which takes $O((\log_2 |V|)^2)$ space.

DATE : / /
PAGE NO. :

Reach (G, s, t, k) → returns 1 when we can reach from s to t within k steps

{

if k = 0 s ≠ t : reject

if k = 1 : t must be adjacent to s
otherwise → reject

else

for all u ∈ V

Reach (G, s, u, k/2) ∧ Reach (G, u, t, k/2) → accept
else → reject

↓

have to store this & next one in stack

⇓

2 × 3 $\log_2 |V|$

let k = |V|

$\log_2 |V|$

~~for graph~~
length of recursion

Total : $O((\log_2 |V|)^2)$

$L \in NSPACE(\log_2 n) \rightarrow L \in SPACE((\log_2 n)^2)$

10/4/19

Theorem Assume that $f(n) \geq n$, then

$NSPACE(f(n)) \subseteq SPACE((f(n))^2)$ — (3)

Proof: We know that : $SPACE(f(n)) \subseteq NSPACE(f(n))$ — (4)

Let A be a language s.t.

$A \in NSPACE(f(n))$, we'll show that $A \in SPACE(f(n)^2)$

↑ ∃ a NDTM N which accepts A in $O(f(n))$ space

Let $w \in A$

$N(w) \rightarrow C_0, C_1, \ldots C_{accept}, \ldots C_{2^{O(f(n))}}$ → ~~some~~ configs

↳ at max, this much configs

DATE : / /
PAGE NO. :

→ Construct a config^n graph on vertices C_0 ——— $C_{2^{O(f(n))}}$

if ∃ path from C_0 to C_{accept} → our work is done. w is accepted by N

Run a DTM M on this graph (M simulates N on w)

If we find a path from C_0 to C_{accept} deterministically → w is accepted by N

Reach (C_i, C_j, t) { → reach from C_i to C_j within t steps

if t = 0 check $C_i == C_j$ → Accept

if t = 1 check whether we reach from C_i to C_j in single step with the help of $\delta \in N$

else ∀ config^{n}s C_m

Reach (C_i, C_m, t/2) ^ Reach (C_m, C_j, t/2) → accept

Otherwise → Reject

}

This func^n we'll run as a sub-routine in our DTM M:

DTM M(w){

Reach (C_0, C_{accept}, $2^{O(f(n))}$)

}

each time → $t = t/2$

space = $\log_2 (2^{O(f(n))}) = O(f(n))$

space taken by this algo :

$|C_i| \leq O(f(n))$

(C_i, C_m, t/2) (C_m, C_j, t/2)

↳① ↳②

$S(t) = S(t/2) + O(f(n))$ $T(t) = 2T(t/2) + \dots$

↓

not $2S(n/2)$ because we can use same space to calculate ① & ②

DATE : / /
PAGE NO. :

$S(t) = O(f(n) + O(f(n)) \quad \log 2^{O(f(n))}$ times

$S(t) = O(f(n)^2)$

→ M accepts w in $O(f(n^2)$ space.

From ③ & ④,

[NPSPACE = PSPACE] ??

if $f(n)$ is polynomial ⇒ $f(n)^2$ is also polynomial
exponential → exponential

from ③, we can conclude
NPSPACE ⊆ PSPACE
from ④, we can conclude A : PSPACE ⊆ NPSPACE
→ PSPACE ≡ NPSPACE

→ PSPACE - complete :

1. $\forall B \in PSPACE \leq_p A$
2. $A \in PSPACE$

→ TQBF = { ϕ | ϕ is a logical formula s.t. all the variables of ϕ are quantified & ϕ is satisfiable }

TQBF ↓ Totally Quantified Boolean formulaes

$(x \vee y) \wedge (x \vee \bar{y})$ → is a SAT but $\notin \phi$ (not quantified)

$\forall x \exists y$ " " " → $\in \phi$ (if satisfiable)

$\forall x \quad (x \vee y) \wedge (\bar{x} \vee z)$ → $\notin \phi$ (y & z are not quantified)

DATE : / /
PAGE NO. :

How to remove quantifier $\forall x$ s.t. satisfiability remains same (if this is true → that formula should also be true)

$x, y \in \{0,1\}$

$$\forall x \exists y \quad (x \vee y) \wedge (x \wedge \bar{y})$$

~~$(x \vee x) \wedge (x \wedge \bar{y}) \vee (y \wedge x) \wedge (y \wedge y$~~

$$\Leftrightarrow \exists y \{(0 \vee y) \wedge (0 \wedge \bar{y})\} \wedge \{(1 \vee y) \wedge (1 \wedge \bar{y})\}$$

Prove: TQBF $\in$ PSPACE

↓

Can have recurrence & for removing quantifiers from ϕ, & use savitch's theorem

$T(\phi)$ {

if $\phi = \exists x \Psi \rightarrow T(\Psi|_{x=0}) \vee T(\Psi|_{x=1})$

if $\phi = \forall x \Psi \rightarrow T(\Psi|_{x=0}) \wedge T(\Psi|_{x=1})$

calculate final value

}

Space Requirement of this algo :→

depth of the Recurrence : No. of variables

$$\boxed{\forall x} \boxed{\exists y} ((x \vee y) \wedge (x \vee \bar{y})) \rightarrow 2$$

$$\boxed{S(|\phi|) = O(n \times |\phi|)}$$ → Polynomial wrt length of formula

↳ no. of variables in ϕ. $n \leq \phi$

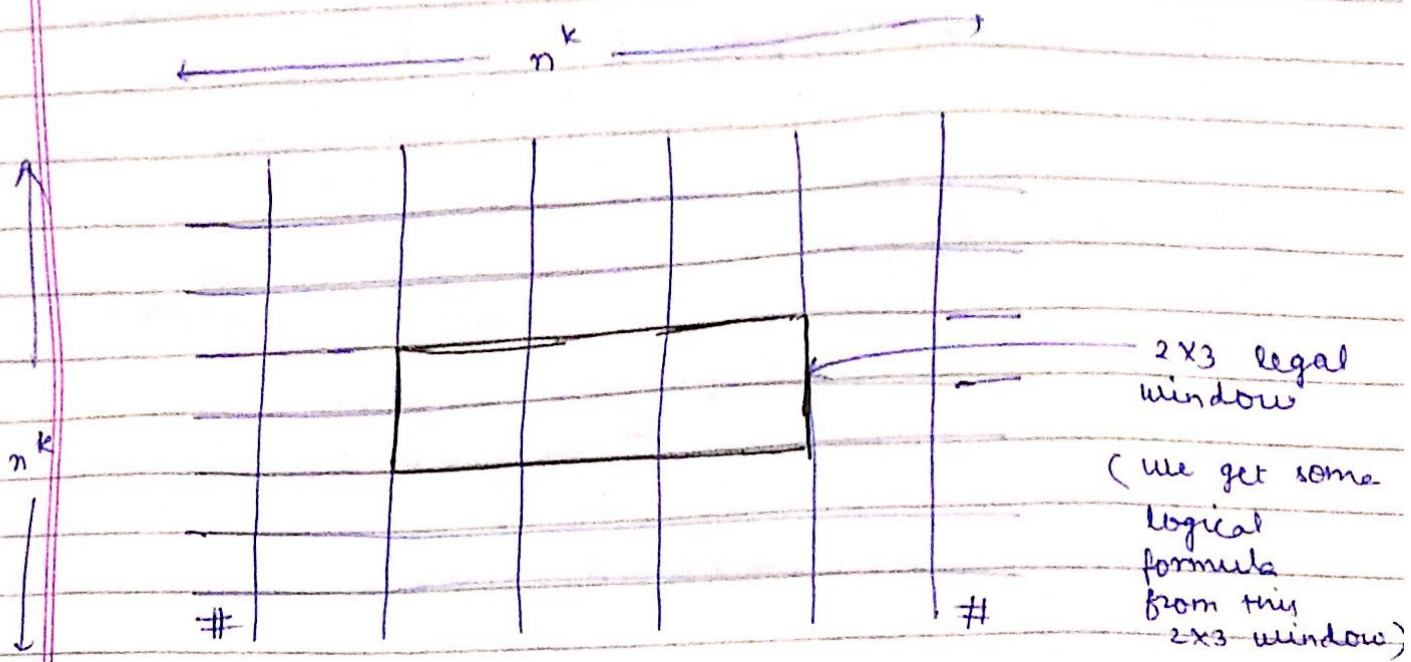
12/4/19

DATE : / /
PAGE NO. :

TQBF : PSPACE - Hard

TQBF ∈ PSPACE : we already know

SAT : NP- Complete



→ Each cell contains exactly 1 symbol

logical formula we got from the window :

$\Phi_{(C_i, C_j, 1)} = \forall\, x_{i,j,s}$

↓

moving from C_i to C_j in one step.

$\Phi_{(C_i, C_m, t)} = \exists\, C_u \left[\Phi_{(C_i, C_u, t/2)} \wedge \Phi_{(C_u, C_m, t/2)} \right]$

↑

At each step it's creating two more formula)

$\Rightarrow \Phi_{(C_{start}, C_{accept}, 2^{O(f(n))})}$

$|\Phi_{C_i, C_m, t}| = 2\,|\Phi_{C_i, C_u, t/2}| + O(f(n))$

↑ length of C_u

Manipulation to reduce the exponential size formula

DATE : / /
PAGE NO. :

$$|\Phi_{C_i, C_m, t}| = \exists C_u \; \forall (C_1, C_2) \in \{ (C_i, C_u), (C_u, C_m) \}$$

quantified variables

$$\exists x_1 \; \forall x_2 \; \exists x_i \; [\Phi_{C_1, C_2, t/2}]$$

Eg. $S(t) = |\Phi_{C_i, C_j, t}|$

$S(t) = S(t-1) + O(f(n))$

↑ length of TM n^k

→ $\exists^1 x_1 \; \forall^2 x_2 \; \exists^1 x_3 \; [(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3)]$

players

$\exists$ (Give particular assignment to x) [Win]

$\forall$ If false then it will win

$x_1 = 1$

$x_2 = 1$

$x_3 = 0$

Putting values

$[(1 \vee 1) \wedge (1 \vee 0) \wedge (0 \vee 1)] = 1$ (First Player win)

If $x_1, x_2, x_3 = 1$

$[(1 \vee 1) \wedge (1 \vee 0) \wedge (0 \vee 0)] = 0$ 2nd wins

DATE : / /
PAGE NO. :

→ Player 1 plays in such a way that whatever assignment player 2 will choose it will always win.

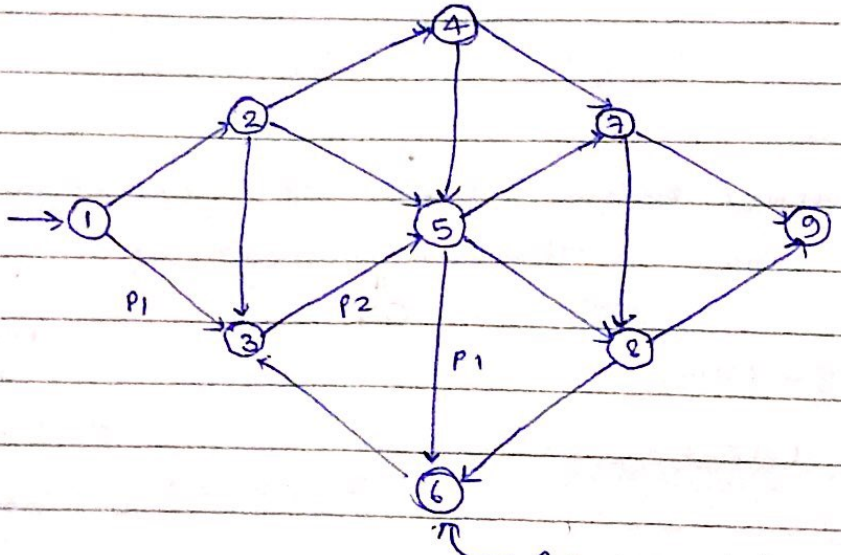
Winning strategy.

→ $x_1 = 1 \quad x_2 = 1 \quad x_3 = \bar{x}_2$

No winning strategy for player 2 for this particular case

Game is P-SPACE complete

Different app" :



P2 looses (reached loosing state)

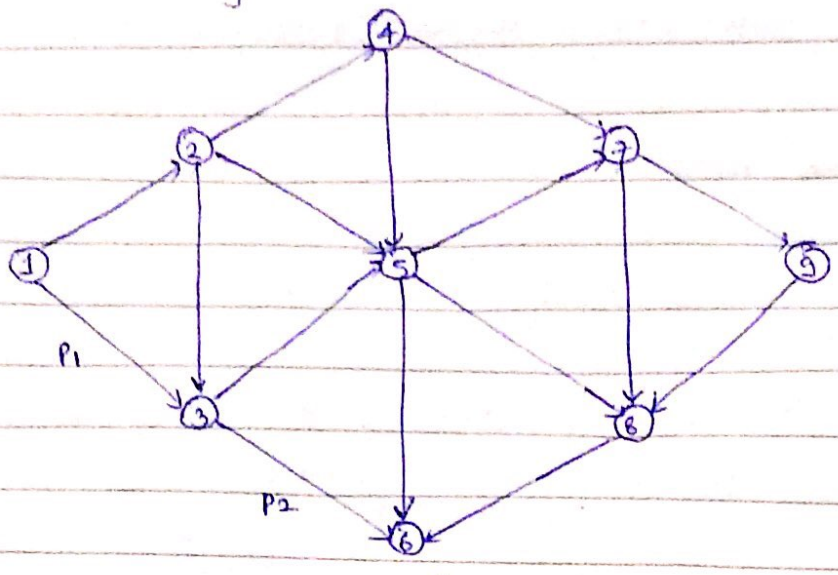
Loosing Criteria :-

① If either player can not move from that vertex

② He has to move to already visited vertex

DATE : / /
PAGE NO. :

Player 2 winning



(P1 looses)

15/4/19

Two player games

1. game on quantified boolean formula → Formula-game (like TQBF) → PSPACE Complete
2. game on a graph → Generalized geography = GG → Formula-game → also PSPACE Complete

→ winning criteria
→ winning strategy of a player

Task: To prove 2. is PSPACE Complete

GG = { < G, b > | Player 1 has winning strategy when he starts at vertex b in graph G }

TO prove GG is PSPACE-Complete

Formula game $\leq_p$ GG

① → Prove it is PSPACE

TM M → runs in polynomial time

DATE : / /
PAGE NO. :

TM M

1. If b has out degree 0 ⇝ (Player 1 looses) → Reject $\forall i$
2. remove b (get G_1 after removing b) & call M (G_1, b_i) where b_i is nbd of b
3. If all accepts → Reject (Player 2 has will move)
4. Otherwise → Accept

From this vertex b_i, player 2 will move ✓

→ depth of this Recursion : at most $|V|$

Space Complexity = $O(|V|)$ ↓ wrt size of input, it is polynomial

Size of input : $O(|V| + |E|)$

⇝ GG is PSPACE

② → Prove GG is PSPACE Hard

↓ using

formula game $\leq_p$ GG

For a given formula, we convert it into graph in polynomial space.

e.g. Suppose, the formula is :

$$\exists x_1 \forall x_2 \exists x_3 \left[(x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3) \right]$$

1. Assume that our formula starts & ends with '$\exists$' & '$\exists$' & '$\forall$' exist alternatively in between
2. If some formula is not in this format, we can convert it.

Ex : $\exists x_1 \ \forall x_2 \forall x_3 \forall x_4 \ [\ \ldots\ ]$

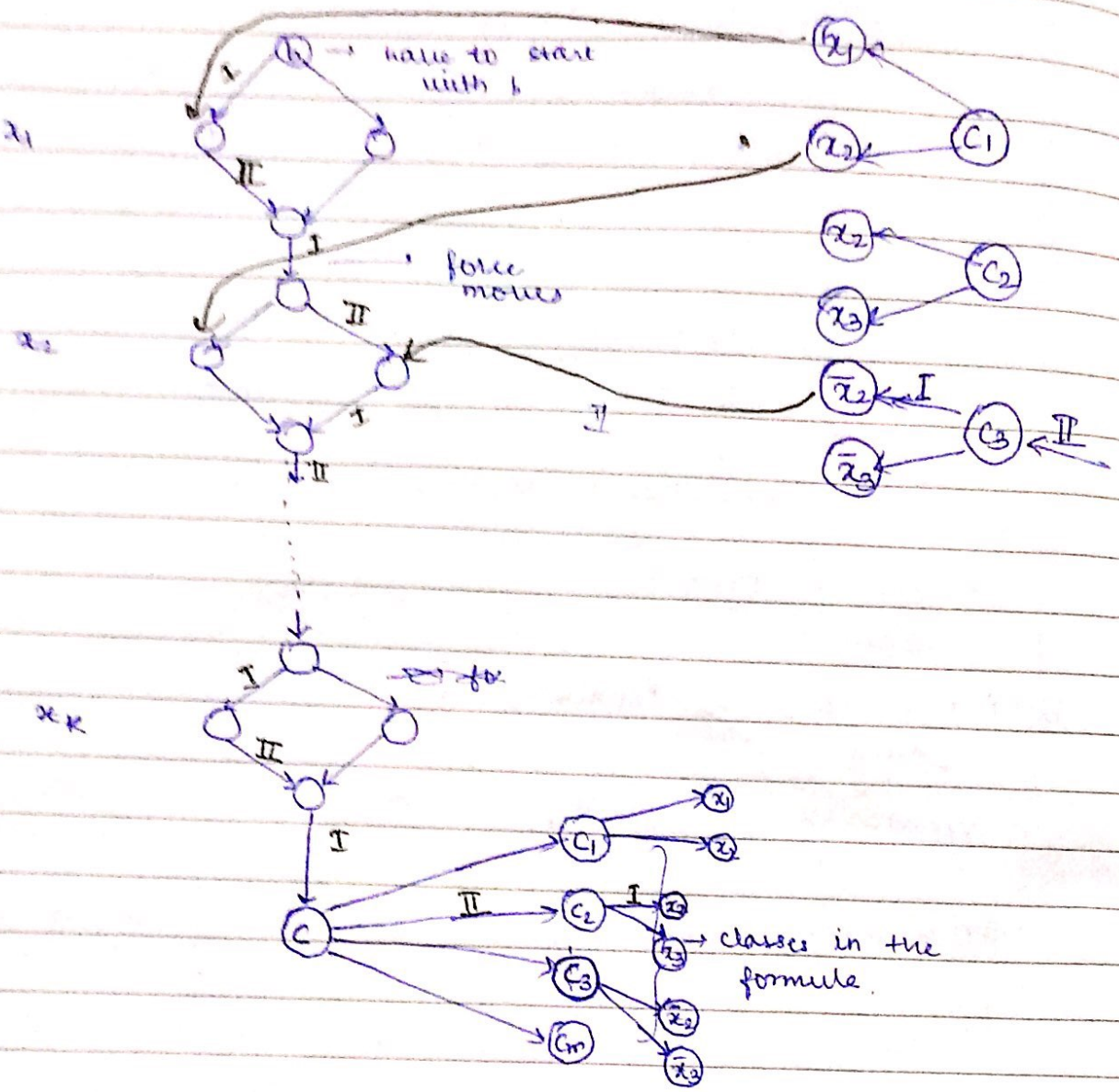
$\Downarrow$

$\exists x_1 \forall x_2 \exists d_1 \forall x_3 \exists d_2 \forall x_4 \exists d_3 \ [\ \ldots\ ]$

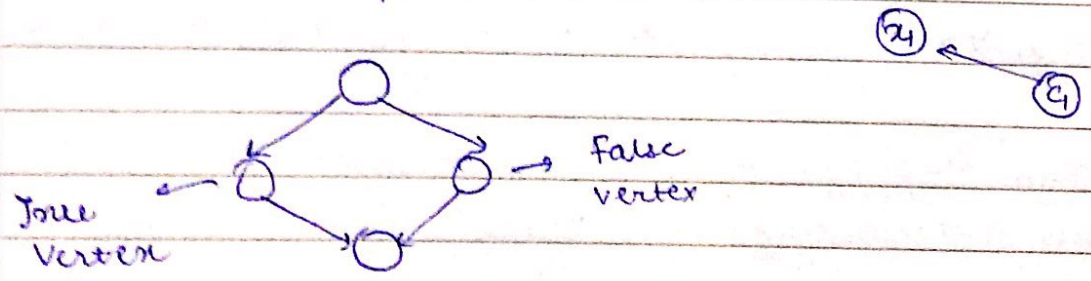
assignment of these dummy variable doesn't affect

satisfiability of original problem.

Converting into Graph :



connect literals present in a class.



if x_1 is ~~false~~ true, then connect x_1 to left

$\bar{x}_1$ ~~else~~ , x_1 to right

DATE : / /
PAGE NO. :

suppose formula is true when $x_1 = 1$, then player will take ↙ this edge if $x_1 = 0$ ↘ (forced edges)

→ 1st player plays '∃' part & 2nd player plays '∀' part

1st player : starts with b. if $x_1 = 1$ → Picks left edge
Player 2 only has 1 choice ↘

suppose $x_2 = 0$

Upto reaching C, our moves are defined based on assignments from here, we'll define our winning strategy.

Player - 1 winning strategy :
suppose formula is satisfiable for some assignment
⇓
each class will be true.

at C → II player will have to move.
↳ whichever path he chooses, ∃ a literal which is True
& " class " " → Player I can find a ~~variable~~ literal in that class which is true.

suppose C_2 : True (x_2 : True)
P2 → goes to C_2 : then Player I chooses x_2
(if more than 1 true variable → choose anyone)

→ Player 2 has no choice → x_2 was declared false & it took → this path.

DATE : / /
PAGE NO. :

Player 2 : winning strategy

If formula is false → ∃ a clause which is false

⇓

Player 2 will choose such a clause

suppose C_3 : false

→ $\bar{x}_2$: false & $\bar{x}_3$: false.

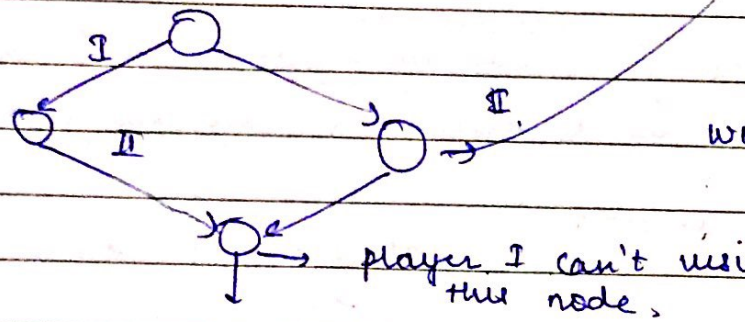
x_2 : T x_3 : f

⇓ used ↙

in x_2 ; we ~~started~~ move.

Same path as earlier :

So, player 2 can move because this node is unvisited



Now, player I won't have any move.

If formula is true → Player I is winning

" " false → " II " "

→ NL = NSPACE (log n) = { L | accepted by ND O(log n) space TM

L = SPACE (log n) { " " " D " "