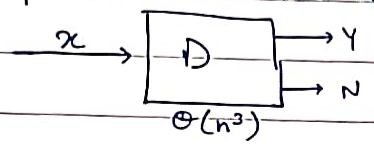
★ Deterministic Time Hierarchy

→ $\boxed{\text{DTIME}(n^c) \subsetneq \text{DTIME}(n^d) \quad ; \quad c < d}$

- Does $\exists L$ s.t. $L \in \text{DTIME}(n^d)$ but $L \notin \text{DTIME}(n^c)$
- The proof doesn't work if c and d are very close.

- **Proof** : $\text{DTIME}(n) \subsetneq \text{DTIME}(n^3)$
  - ↳ infinitely many languages / turing machines (say $M_1, M_2, \ldots$)

We'll construct a turing machine running in $\Theta(n^3)$ time and is not possible to run in $\Theta(n)$ time by ~~some~~ any modifications.

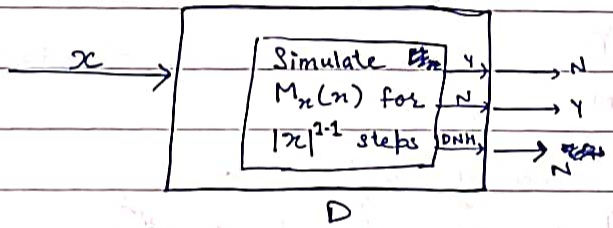


1) Simulate linear time turing machines.
   ↳ Take atleast quadratic time.

- **Idea** : D is going to contradict each machine $M_1, M_2, \ldots$
  i.e., $\exists x_i$ s.t. $M_i(x_i) \neq D(x_i)$

If we show this, we can say there is no linear time machine M_k that can run D. (Since, $\exists x_k$ s.t. $M_k(x_k) \neq D(x_k)$)
$\Rightarrow L_D \notin \text{DTIME}(n)$

- How to design such D ? Using idea of **diagonalisation**.



Runtime of D = $\Theta(|x|^{2.2})$ + some book-keeping

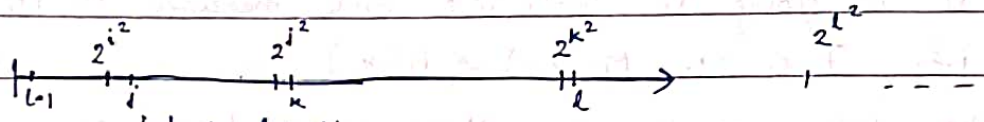
Runtime of D = $\Theta(n^3)$ at worst

Say machine M solves $L(D)$ in $\Theta(n)$ time, i.e., $L(M) = L(D)$
Give $x = \alpha_M$ as input.

If $M(\alpha_M) = 1 \Rightarrow$ Simulation gives Yes $\Rightarrow D(\alpha_M) = 0 \Rightarrow L(M) \neq L(D)$
If $M(\alpha_M) = 0 \Rightarrow$ ——,,—— No $\Rightarrow D(\alpha_M) = 1 \Rightarrow L(M) \neq L(D)$

- What if Simulation doesn't halts in $|x|^{1.1}$ steps ?
  It could happen as runtime of M maybe say $100|x|$ and $100|\alpha_n| > |\alpha_n|^{1.1}$
  We know there are infinitely many encodings of M.
  We'll choose α_n s.t. $|\alpha_n|$ is large enough to satisfy $100|\alpha_n| < |\alpha_n|^{1.1}$.
- But we don't know the constant (100) !
  $M \to \alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_k, \ldots$
  $\exists \alpha_i$ s.t. $M(\alpha_i) \neq D(\alpha_i) \Rightarrow L(M) \neq L(D)$
  ↳ runtime of $M < |\alpha_i|^{1.1} \Rightarrow$ M can't solve L(D).
  Hence, $\exists$ L s.t. $L \in DTIME(n^3)$ but $L \notin DTIME(n)$.
  $DTIME(n) \subsetneq DTIME(n^3)$

★ Non-Deterministic Time Hierarchy



- Partition input length & $i \to 2^{i^2}$ and each partition is indexed.
- We have infinitely many NTMs $N_1, N_2, \ldots \in NTIME(n)$
- We'll construct $D \in NTIME(n^3)$
  D contradicts N_i in i^{th} partition $(i \to 2^{i^2})$

$O(n) \leftarrow 0)$ If x contains 0, simply reject x.

$O(n^2) \leftarrow 1)$ Now $x = 1^k$. Find l s.t. $l \leq k \leq m = 2^{l^2}$
↳ Say partition i

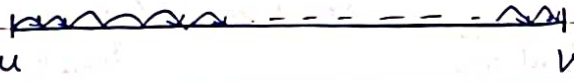
$O(n^{2.2}) \leftarrow 2)$ If $l \leq k < 2^{l^2}$
almost
Sim $N_i(1^{k+1})$ for $(k+1)^{1.1}$ steps and produce same result.

$O(n) \leftarrow 3)$ If $k = 2^{l^2}$
$N_i(1^l)$ — find correct ans. using computation tree, and contradict it.
(It will take exp. time $\Rightarrow 2^{l^{1.1}}$ atmax $\Rightarrow$ linear in input 1^k, $k = 2^{l^2}$)

- Runtime of D = $O(n^3)$ including some book-keeping activities.

- Say N running in $O(n)$ solves L(D), i.e., L(N) = L(D).
  N → j^{th} NTM → look at j^{th} partition ($[u, 2^{u^2}]$)
  Then ⇒ $1^u \in L(N_j) \iff 1^u \in L(D)$
  $D(1^u) = N_j(1^{u+1})$
  ⇒ $1^u \in L(N_j) \iff 1^u \in L(D) \iff 1^{u+1} \in L(N_j)$
  $D(1^{u+1}) = N_j(1^{u+2})$ $\iff 1^{u+1} \in L(D)$
  ⋮
  ⇒ $1^u \in L(N_j) \iff 1^{u+1} \in L(N_j) \iff \cdots\cdots \iff 1^{2^{u^2}} \in L(N_j)$
  $\iff 1^{2^{u^2}} \in L(D)$ ----(i)
  $D(1^{2^{u^2}}) =$ Contradict $N_j(1^u)$
  i.e., $1^{2^{u^2}} \in L(D) \iff 1^u \notin L(N_j)$ ----(ii)
  From (i) & (ii)
  $1^u \in L(N_j) \iff 1^u \notin L(N_j)$ ⇒⇐

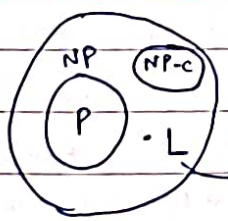
- Hence, no such N_j exists $\left\{ 1^u \in L(N_j) \iff 1^v \in L(N_j) \iff 1^v \in L(D) \iff 1^u \notin L(N_j) \quad (v = 2^{u^2}) \right\}$

★ 

"Lazy Diagonalization"

we want to Compute answer at start but push it to end so that input is large enough in order to make it linear in input length.

★ Ladner's Theorem

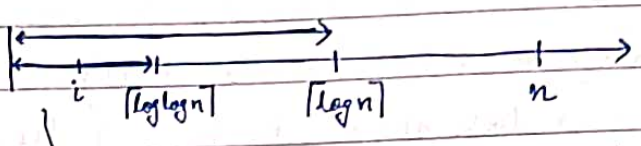


The language $L \in NP$ but $L \notin P$ and $L \notin$ NP-complete.
Such languages — NP-intermediate.

- If $P \neq NP$, then ∃ an NP-intermediate language L.

* $SAT_H = \{ \varphi \# 1^{|\varphi|^{H(|\varphi|)}} \mid \varphi \in SAT \}$

- $H(n)$:



M_i solves $SAT_H|_{\leq \lceil \log n \rceil}$ (, i.e., for strings upto length $\lceil \log n \rceil$) in $i|x|^i$ steps.

If M_i says Yes — $H(n) = \min_i (M_i \text{ solves } SAT_H)$

If all M_i says No — $H(n) = \lceil \log \log n \rceil$

- Time to compute $H(n)$?

$= \log\log n \times n \times \log\log n (\log n)^{\log\log n}$ = polynomial in n

↓ no. of TMs we have to check

↓ strings upto length $\lceil \log n \rceil$ i.e., $2^{\log n} = n$ strings

↓ $i|x|^i$ steps for each string; max $i \to \log\log n$; max $|x| \to \log n$

- $M_i(x) = 1 / 0$

How to know if this answer is correct ?

We have to check $x \in SAT_H$? (if $M_i(x) = 1$)

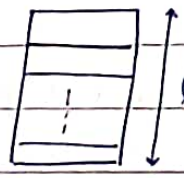
$x = \varphi \# 1^{|\varphi|^{H(|\varphi|)}}$

↓ atmost $\lceil \log n \rceil$ size; Take exp. time $O(2^{\log n}) = O(n)$ to solve

↓ need to know $H(|\varphi|)$ again; ⇓ atmax $\lceil \log n \rceil$; ⇓ need to know $H(\log n)$

For $H(n)$ — need $H(\log n)$ — need $H(\log\log n)$ --- "Recursion"

We can have a look-up table.



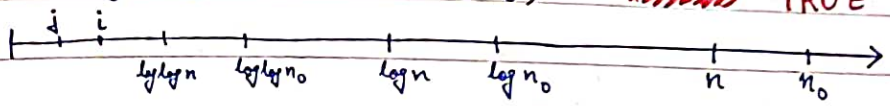
- We are providing $H(|\varphi|)$ and $|\varphi| = n$ i.e., φ has n characters.

So, input to H has n characters.

∴ input length $\neq \log n$ but $= n$ only.

- **Claim** : $H(n) = O(1) \iff SAT_H \in P$

- $n \le n_0 \Rightarrow H(n) \le H(n_0)$ → $H(n)$ is non-decreasing. TRUE



i & j solves SAT_H for strings upto length $\log n$

$\because i > j$, we say $H(n) = j$

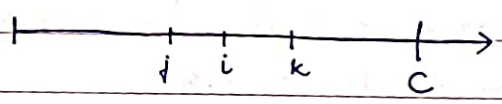
But it may happen that i solves SAT_H for length upto $\log n_0$ but not j

$\therefore H(n_0) = i \not> j = H(n)$ — if some $k < j$ solves upto $\log n_0$, it also solves upto $\log n \Rightarrow H(n) = k \Rightarrow\Leftarrow$

$\Rightarrow H(n) \not\ge H(n_0)$

Hence, $H(n)$ is a non-decreasing function.

- Say, $H(n) = O(1)$, i.e., $H(n)$ is bounded above by some constant (say C).



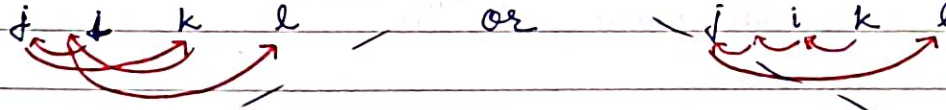
Say $n < n_0 < n_1$

if $H(n) = j$, $H(n_0) = i$ and $H(n_1) = i$

Not possible since if i solves upto $\log\log n_1$, it also solves upto $\log\log n_0 \Rightarrow H(n_0) = i \Rightarrow\Leftarrow$

Not of any use now

But $H(n_1) = k$ → can be true.



- $H(n)$ can go down or remain constant but if it increases, it will always increase beyond maximum value taken till now.

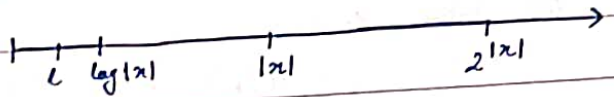
- Since $H(n) \le C$, it will happen that some 'l' occurs infinitely often for value of $H(n)$.

$H(n)$ is non-decreasing, $\therefore \lim_{n\to\infty} H(n)$ will be bounded (say l)

- 'l' is occuring infinitely often $\Rightarrow$ M_l solves SAT_H

Say $L(M_l) \neq SAT_H$

$x \in SAT_H \setminus L(M_l)$



Since l occurs inf. often, for some $n' > 2^{|x|}$; $H(n') = l$

$\therefore$ l solves SAT_H for strings upto length $\log n' > |x|$

$\Rightarrow$ M_l solves SAT_H $\Rightarrow\Leftarrow$

Runtime of $M_l = i|x|^i$, i.e., $l|x|^l$ = polynomial in input

$\therefore$ $SAT_H \in P$

Hence, $H(n) = O(1) \Rightarrow SAT_H \in P$

$\rightarrow$ $SAT_H \in P \Rightarrow H(n) = O(1)$

$SAT_H \in P \Rightarrow \exists$ DTM M s.t. $L(M) = SAT_H$ in poly. time

$\hookrightarrow O(c.n^d)$

$M \rightarrow l \geq c, d$ (as there are inf. many encodings of M)

Runtime of $M_l = l.n^l$

Definition of $H(n)$ implies for $n > 2^{2^l}$, $H(n) \leq l$.

$\Rightarrow$ $H(n) = O(1)$.

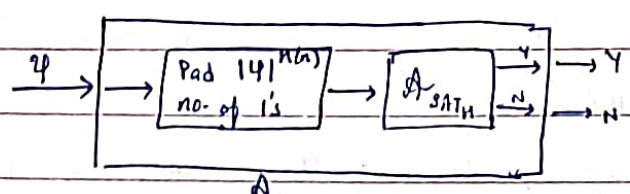
$\rightarrow$ Using this claim, we can show $P \neq NP \Rightarrow SAT_H \notin P$, $\notin$ NP-comp

- Suppose $SAT_H \in P \Rightarrow H(n) = O(1)$

SAT_H is simply SAT padded with $|\varphi|^{H(n)}$ (i.e., polynomial since $H(n)=O(1)$ no. of 1's.

$\Rightarrow$ This polynomial time algorithm for SAT_H can be used to solve SAT

$\Rightarrow$ $P = NP$ $\Rightarrow\Leftarrow$ Hence, $SAT_H \notin P$



$\varphi \# 1^{|\varphi|^{H(n)}} \in SAT_H$

Hence, it can be used to solve SAT

$SAT \leq SAT$

→ $P \neq NP \Rightarrow SAT_H$ is not NP-complete

Assume $SAT_H \in$ NP-complete

$\Rightarrow SAT \leq_P SAT_H$

We have already shown $SAT_H \notin P \Rightarrow H(n) \to \infty$ (unbounded)

- Reduction function $f \to O(cn^d)$ time

$$\varphi \longmapsto \varphi \# 1^{|\varphi|^{H(n)}}$$

Since f has only cn^d time, it can print atmax $|\varphi|^d$ no. of 1's.
But $H(n) \to \infty$, $\therefore$ for some n large, $H(n) > d$
$\therefore$ reduction can't work in cn^d time.

- But it may happen reduction works as $\varphi \longmapsto \varphi' \# 1^{|\varphi'|^{H(|\varphi'|)}}$

~~s.t. $H(n) \leq d$ always~~

$$\varphi \begin{cases} \to \varphi \# 1^{|\varphi|^{H(|\varphi|)}} \\ \to \varphi' \# 1^{|\varphi'|^{H(|\varphi'|)}} \end{cases}$$

time $= |\varphi|^d$

ⓐ if $H(|\varphi'|) < H(|\varphi|)$

$\Rightarrow |\varphi'| < |\varphi|$ (since $H(n)$ is non-decreasing)

No. of 1's to print $= |\varphi'|^{H(|\varphi'|)}$

ⓑ if $H(|\varphi'|) = H(|\varphi|) > d$

Since we only have $|\varphi|^d$ time $\Rightarrow |\varphi'| < |\varphi|$

- ⓐ & ⓑ $\Rightarrow |\varphi'| < |\varphi|$

if $H(|\varphi'|) > d$, then we can print φ'', s.t. $|\varphi''| < |\varphi'|$

$$\vdots$$

$$|\varphi| > |\varphi'| > |\varphi''| > \cdots > |\emptyset|$$

$H(\cdot)$ values decreases or remain same, we'll finally get some $\emptyset$ s.t. $H(|\emptyset|) \leq d$

$$\varphi \longmapsto \emptyset \# 1^{|\emptyset|^{H(|\emptyset|)}}$$

s.t. $|\varphi| > |\emptyset|$

$H > d$ ↙ ↘ $H \leq d$

Max. no. of times, red^n need to be applied $= |\varphi|$

Time $= |\varphi| \times c|\varphi|^d =$ polynomial

- There will be only constant no. of ϕ's s.t. $H(|\phi|) \leq d$ (as d is constant)
  we can solve all these ϕ's → constant time (2^d → constant)
  Thus, we can reduce any formula $\psi \mapsto \phi$ s.t. $H(|\phi|) \leq d$ and then look-up the table for answer of ϕ which takes constant time.
  Max. no. of times red^n need to be applied = $|\psi| \times c|\psi|^d$ = polynomial
  Hence, we solve SAT in poly. time.

$$\psi \mapsto \phi \# 1^{|\phi|^{H(|\phi|)}}$$

$$\psi \in SAT \iff \phi \# 1^{|\phi|^{H(|\phi|)}} \in SAT_H \iff \phi \in SAT$$

$\Rightarrow$ $SAT \in P \Rightarrow P = NP$ $\Rightarrow\Leftarrow$

Hence, SAT_H can't be NP-complete.

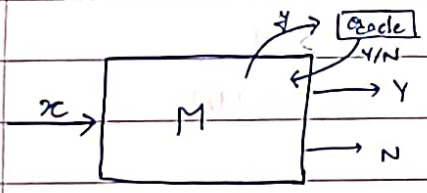
→ ~~If~~ $P \neq NP \Rightarrow SAT_H \notin P$ and $SAT_H \notin$ NP-comp.

Clearly, $SAT_H \in NP$

$\Rightarrow$ SAT_H is NP-intermediate language.

★ **Oracle Turing Machines**

→ Uses a oracle to solve



- Oracle can be used at any step within M
- Return Y/N for input y, based on what oracle solves, i.e; $y \in L_{oracle}$?

- Returns response in just 1 unit of time.
- Oracle can be of any language (Eg: SAT, HALT, Diagonal, etc.)
- Time required to make a query $\propto$ Length of query.

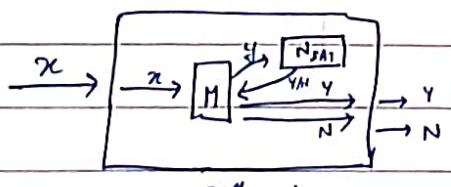
★ $P^{SAT} = NP$

P^{SAT} : Set of all languages that can be solved in polynomial time deterministically using SAT as an oracle.

- Proof : $L \in P^{SAT}$

$\exists$ DTM M s.t. $L(M) = L$ running in poly. time

M uses SAT-oracle. Replace this oracle in M with NDTM of SAT.



M'

$x \in L \iff x \in L(M')$

$\Rightarrow L(M') = L$, but M' is NDTM running in poly. time (since both M & N_{SAT} are poly.)

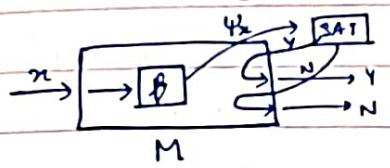
$\therefore$ M' is NDTM that solves L in poly. time

$\Rightarrow \quad L \in NP \quad \Rightarrow \quad P^{SAT} \subseteq NP \quad ----(i)$

- Say, $L \in NP$

  We know $\quad L \leq SAT$

  $x \mapsto \psi_x \quad$ s.t. $\quad x \in L \iff \psi_x \in SAT$



  $x \in L \iff \psi_x \in SAT \iff$ [illegible] $x \in L(M)$

  $\Rightarrow \quad L(M) = L$ and $f(.)$ takes poly. time

  $\Rightarrow \quad L \in P^{SAT}$

  $\Rightarrow \quad NP \subseteq P^{SAT} \quad ----(ii)$

  From (i) & (ii) ; $\quad P^{SAT} = NP$

  Hence Proved

★ $O_{exp} = \{ \langle M, x, 1^n \rangle \mid M$ accepts x in 2^n steps $\}$

$P^O = NP^O = EXP$

- $P^O \subseteq NP^O$ is obvious.
- $NP^O \subseteq EXP$. Let $L \in NP^O$.

  $\exists$ [illegible] N^O be NDTM s.t. $L(N^O) = L$ running in Cn^d steps

  No. of nodes $= O(2^{Cn^d})$

  Makes an oracle query here : $\langle M, x, 1^m \rangle$

  To find answer : Simulate M on u for 2^m steps.

  Worst possible value of $m = Cn^d$

  Time to find answer for 1 query $= O(2^{Cn^d})$

  No. of oracle query atmax $= O(2^{Cn^d})$

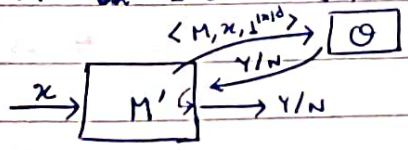
  Total time required to solve $= O(2^{Cn^d}) \times O(2^{Cn^d}) = O(2^{C'n^d})$

  This is deterministic machine without oracle taking exp. time.

  $\therefore \quad L \in EXP \quad \Rightarrow \quad NP^O \subseteq EXP$

- $EXP \subseteq P^O$. Let $L \in EXP$

  $\exists$ M [illegible] s.t. $L(M) = L$ running in $O(2^{n^d})$



  Time to make query $= O(|x|^d)$

  M' solves L in $O(n^d)$ time using O.

  $L \in P^O$

- Thus, we have $P^O \subseteq NP^O \subseteq EXP \subseteq P^O$

  Hence, $P^O = NP^O = EXP$

  Hence Proved

★ $\exists$ a Language O s.t $P^O \neq NP^O$

- We have to find O and a language L s.t. $L \in NP^O$ but $L \notin P^O$
- We'll take an O s.t. $U_O \in NP^O$ but $U_O \notin P^O$.

  $U_O \rightarrow \exists\ N^O,\ L(N^O) = U_O$

  $\hookrightarrow \forall\ M^O,\ L(M^O) \neq U_O$ ---- listed sequentially (similar to proof of D-TIME-Hierarchy)

- Assume for now:

1) all oracle TM can be encoded using $\mathbb{N}$.
2) There are infinitely many representations.
3) Universal TM exists with oracle O.

- We have O : strings to put in O

  O^x : strings not to put in O

- Two counters i : contradicted all M_j^O for all $j < i$

  n : decided for $\forall x$, $|x| < n$ whether to put x in O or O^x.

→ Proceed iteratively:

$i \rightarrow M_i^O$

Run $M_i^O(1^n)$

i) M_i^O queries to O for query length $< n$

↳ we have already decided for strings $< n$ → reply accordingly.

ii) query length $\geq n$ (say y)

↳ put y in O^x and return no.

- Run this simulation for $\frac{2^n}{10}$ steps (we only need poly-time machines)

Case I : $M_i^O(1^n) =$ Yes, i.e., $1^n \in L(M_i^O)$

↳ Put all strings of length n in O^x

There there is no string of length n in $O \Rightarrow 1^n \notin U_O$

$\therefore U_O \neq L(M_i^O)$

Case II : $M_i^O(1^n) = NO$, i.e., $1^n \notin L(M_i^O)$

↳ Some string of length n might already be put in O^x for case $n' < n$ & query length of n.
that all strings are in O^x

It is not possible, since there are total 2^n strings & we run only for $2^n/10$ steps.

Put remaining strings in $O \Rightarrow 1^n \in U_O$

$\therefore L(M_i^O) \neq U_O$

Case III : $M_i^O(1^n)$ doesn't halts in $2^n/10$ steps.

It doesn't matter whether to put remaining strings in O or O^x.

→ Claim : $U_O \in NP^O$ but $U_O \notin P^O$

- For x, if x is not of form 1^m → reject

Non-deterministically guess y, $|y| = m$ & ask $O(y)$?

$1^m \in U_O \iff \exists y, |y| = m, y \in O$

Hence, $U_O \in NP^O$

- Say, M_j^O solves U_O in cn^d time.

Check the length m from construction in j^{th} iteration

If $cm^d \leq \frac{2^m}{10}$ → we have contradiction (Case I or case II)

Else if $cm^d > \frac{2^m}{10}$ → $\exists k, k > j$ representing M_j^O (inf. representations)

For that k, take m'

$c(m')^d \leq \frac{2^{m'}}{10}$ → Contradiction

Hence, $U_O \notin P^O$

→ Thus, $\exists$ language O s.t. $P^O \neq NP^O$

Hence Proved.

★ Infinitely many natural number ($\mathbb{N}$) representations of oracle TM.

- One special state q_O to make oracle query.

$q_O \swarrow \quad \searrow$ depending on reply.

$q_{yes} \quad q_{NO}$

- Similar, to two tape TM with this special states.
  & we can encode this similarly.
- What oracle we use doesn't matters. We only need to encode the code of TM and not the whole model.

$Q = \{q_0, q_1, q_O, q_{yes}, q_{no}, \dots\}$

To make query : $(q, x, x) \mapsto (q_O, \dots)$

$\swarrow \quad \searrow$

$q_{yes} \quad q_{NO}$ in 1 step.

$M \to \alpha \to i$,

$M^O \to \alpha' \to i'$ (similarly)

- Inf. many α's
- Machine doesn't know what oracle is ; only designer knows.

Code of M^O = Code of $M^{O'}$

Oracle is part of model.

★ $\exists\ L_1, L_2$ s.t. $P^{L_1} = NP^{L_1}$ and $P^{L_2} \neq NP^{L_2}$

$\Downarrow$

$P = NP$?

★ Diagonalization alone can't prove / disprove $P = NP$!!

I) TMs can be encoded as infinetly many $\mathbb{N}$. } "Diagonalization"

II) Simulation is possible.

- Say, using these properties, we proved $P = NP$ (i.e., using diagonelization)
  In the proof, we replace $P \to P^{L_2}$; $NP \to NP^{L_2}$

$M \to M^{L_2}$; $N \to N^{L_2}$

(i.e., these two properties hold under L_2 oracle)

↳ proof remains correct and we'll get $P^{L_2} = NP^{L_2}$

· But $P^{L_2} \neq NP^{L_2}$ $\Rightarrow\Leftarrow$

Hence, our assumption that the proof uses only diagonalization is wrong.

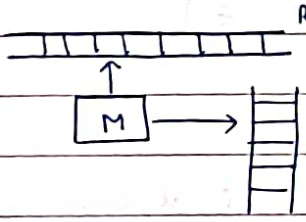
- If someone prooves $P \neq NP$; replace L_2 with L_1 to get $P^{L_1} \neq NP^{L_1}$ $\Rightarrow \Leftarrow$

→ Thus, proving $P = NP$ or $P \neq NP$ is not possible by just using diagonalization. The proof must use some property that doesn't holds in oracle domain.

☆ Space Hierarchy

→ **SPACE (f(n))** – all languages that can be solved by DTM using atmost $O(f(n))$ space.

- Study space using two-tape TM



- Space used on work-tape is considered only when considering space used by the TM.

→ **NSPACE (f(n))** – all languages that can be solved by NDTM using atmost $O(f(n))$ space.

→ $PSPACE = \bigcup_{c \geq 0} SPACE(n^c)$

→ $NPSPACE = \bigcup_{c \geq 0} NSPACE(n^c)$

~~PSPACE = NPSPACE~~

~~Enumerate all machines~~

→ **L** = logspace complexity class, i.e., SPACE ($\log^c n$)

→ **NL** = non-det. logspace comp. class, i.e., NSPACE ($\log^c n$)

} Sub-linear complexity classes

★ $L \subseteq NL$

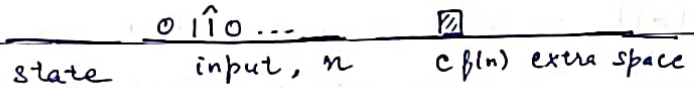
☆ $DTIME(f(n)) \subseteq SPACE(f(n)) \subseteq NSPACE(f(n)) \subseteq DTIME(2^{f(n)})$

- $DTIME(f(n)) \subseteq SPACE(f(n))$ is true since a machine running for $f(n)$ steps can't use more than $f(n)$ space.
- $SPACE(f(n)) \subseteq NSPACE(f(n))$ is obvious with definition of NDTMs and DTMs.

→ How to prove things in SPACE ?

$M \rightarrow |x| = n \rightarrow cf(n)$ space.

- Define configurations & configurations graph (M, x)
- Configuration :



No. of possible configurations = $|Q| \times n \times |\Gamma|^{cf(n)}$

- Configuration Graph : $G_{M,x} = (V, E)$

V = all configurations

$|V| = O(2^{cf(n)})$

$E = \{(c, c') \mid$ directed edge, if a valid transition exists from $c \rightarrow c'\}$

- $NSPACE(f(n)) \subseteq DTIME(2^{f(n)})$

Say, $L \in NSPACE(f(n))$

i.e; $\exists$ NDTM N solving L using $cf(n)$ space.

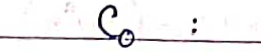
⇓

Construct M solving L in $O(2^{cf(n)})$ steps.

$x \in L(N) \Leftrightarrow M$ accept x.

M : construct $G_{N,x}$ ---- time req. $\leq O(2^{cf(n)})$

Now just check reachability $c_0 \rightsquigarrow c_f$

c_0 : 

↳ unique

C_f : q_f, ^ (unique), _____, B (we define in such way, else there could be multiple C_f)

Time taken to check reachability $= O(|V| + |E|) = O(2^{cf(n)})$

Thus, $x \in L(N) \iff x \in L(M) = L$

$L \in DTIME(2^{f(n)})$

Hence Proved

★ PSPACE = NPSPACE

• Non-determinism doesn't give any advantage w.r.t. space

★ $NSPACE(f(n)) \subseteq SPACE(f^2(n))$; $f(n)$ → Time constructible

$L \in NSPACE(f(n)) \Rightarrow \exists N$ using $cf(n)$ space solving L

↳ $(G_{N,x}, C_0, C_f)$ reachability ?

$\exists$ path $C_0 \xrightarrow{\leq 2^{f(n)}} C_f \Rightarrow \exists C$ (midpoint), $C_0 \xrightarrow{\leq 2^{f(n)-1}} C$ & $C \xrightarrow{\leq 2^{f(n)-1}} C_f$

```
REACH (C1, C2, l) {   // C1 ⇝ C2 in 2^l steps
    for all C {   → Base case !!
        a1 = REACH (C1, C, l-1)
        a2 = REACH (C, C2, l-1)
        If (a1 && a2) return TRUE
    }
    return FALSE
}
```

Base case :

```
If (l == 0) {
    If (C1 == C2) return T
    else return F
}
```

↗ reuse space for a_1 & a_2

$s(l) = s(l-1) + f(n)$ → for storing a configuration

$= s(l-2) + 2f(n)$

$\boxed{s(l) = (f(n))^2}$

Hence, $NSPACE(f(n)) \subseteq SPACE((f(n))^2)$

$\Rightarrow$ NPSPACE = PSPACE

Hence Proved

☆ PSPACE - complete

→ Notion of reduction in PSPACE ?

$L \leq_p L'$; $p \to$ poly. space

M, n^c ; n', n^d

$x \mapsto f(x)$; $f(x)$ is poly. space computible

$x \in L \Leftrightarrow f(x) \in L'$ $\quad f(x) : \Sigma^* \to \Sigma^*$

- But issue here is $L \in$ PSPACE and we are giving poly. space to $f(x)$. So, red^n machine can find out correct answer of $x \in L$? and simply output U/V $(U \in L', V \notin L')$ for any input x.
- Resource available to reduction machine is same as that available to L, thus resource available to machine should be less than that available to L.

- We stick to old def^n of reduction using poly-time $f(x)$. $f(x)$ if poly-space can take exponential time but we limit it to poly. time.

☆ $L \leq_p L'$ and $L' \in$ PSPACE $\Rightarrow$ $L \in$ PSPACE

$L' \to M$, cn^d space

$x \xmapsto{M'} f(x)$, M' take en^b steps

Max. length of $f(x) = O(en^b)$

Space req. to solve $x \in L^* = c(en^b)^d = c'n^{d'}$

$\Rightarrow$ $L \in$ PSPACE

☆ $L = \{ \langle M, x, 1^n \rangle \mid M$ accepts x in atmost n space $\}$ is PSPACE complete.

- $L' \in$ PSPACE $\to M$, cn^d

$x \longmapsto \langle M, x, 1^{c|x|^d} \rangle$

Time req. = $O(1) + O(cn^d)$, i.e., polynomial

$L' \leq_p L$, $\forall L' \in$ PSPACE

- $L \in$ PSPACE since it takes linear space atmost

★ Quantified Boolean Formulae : $\exists x_1 \forall x_2 \ldots \psi(x_1, x_2, \ldots)$

→ Boolean formulae in which all variables are quantifies ($\exists, \forall$) and none of the variable is free / un-quantified.

Eg : $\exists x, \exists y, \forall z\ (x \vee y \vee z)$

- QBF ∈ PSPACE since we can list all variables (size = n) and check. Time will be exponential but space is re-usable hence, polynomial.

★ QBF is PSPACE-complete.

- $L \in$ PSPACE, M s.t. $L(M) = L \rightarrow cn^d$ space

  $x \longmapsto \psi_x$ s.t. $x \in L \Leftrightarrow \psi_x \in$ QBF

- $\psi_i(C_1, C_2^*)$: There is a path from $C_1 \rightarrow C_2^*$ in conf. graph of atmost 2^i steps.

  $\psi_i(C_1, C_2^*) = \exists C^* (\psi_{i-1}(C_1, C^*) \wedge \psi_{i-1}(C^*, C_2))$

- $\psi_0(C_1, C_2)$ = True iff $C_1 = C_2$ (As done in SAT proof)

Size of formula (S_i) = $2S_{i-1} + cn^d$

$= 2^i + icn^d$ ---- exponential size formula

We have to solve $\psi_{cn^d}(C_0, C_f)$

Size = $O(2^{cn^d})$ ---- red^n machine can't write it.

- Some modifications need to be done to remove the factor of 2.

$$\psi_i(C_1, C_2) = \exists C \forall C', C'' \{((C' = C_1) \wedge (C'' = C)) \vee ((C' = C) \wedge (C'' = C_2)) \Rightarrow \psi_{i-1}(C', C'')\}$$

(First clause will be true only twice & for that value ψ_{i-1} should be true)

$S_i = S_{i-1} + k \cdot cn^d$, k is constant

$S_i = ikcn^d$

- We start with $i = cn^d$; Size = $(cn^d)^2$

  i.e; polynomial in size. Hence, red^n machine can print it.

$x \longmapsto \psi_{cn^d}(C_0, C_f)$

$x \in L \Leftrightarrow \exists$ path from C_0 to $C_f \Rightarrow \psi_{cn^d}(C_0, C_f)$ is true

$L \leq_p$ QBF, $\forall L \in$ PSPACE & QBF ∈ PSPACE

⇒ QBF is PSPACE-complete.

★ PATH = $\{\langle G, s, t\rangle \mid \exists$ path in G from s to $t\} \in NL$

n - vertices ⇒ space req. to store a vertex or edge = $O(\log n)$

- Take a counter of no. of vertices guessed : $O(\log n)$ space
- Start with s, guess a vertex u_1 and check if there is an edge $s \to u_1$ in G and increment the counter.
- If such edge exists, take u_1 & guess u_2 ---
- Repeat until we reach t (path exists) or counter reaches $\log n$ (path doesn't exists).
- Non-determinism allows to make mistakes.

⇒ PATH $\in$ NL

- Notion of reduction in NL ?
  - $L_1 \leq_p L_2$,

  $x \xmapsto{f} f(x)$ ⇒ Since L_2 takes NL space.

  No. of conf. = $O(\log n)$

  Size of conf. graph = $O(2^{c \log n}) = O(n^c)$.

  We can solve L_1 in polynomial time.
- Thus, we can't have poly-time reductions in NL.

★ $f : \Sigma^* \to \Sigma^*$

$f(x)$ is implicitly log-space computable if :

i) for any x, $|f(x)| \leq cn^d$

ii) $L_1 = \{(x, i) \mid i \leq |f(x)|\}$ is log-space computable by DTM.

iii) $L_2 = \{(x, i) \mid f(x)|_i = 1\}$ is —,, —,, —,, —

↳ i^{th} bit of $f(x)$

★ Then for $L, L' \in NL$

$L \leq L'$

iff $\exists f$ s.t. $x \in L \iff f(x) \in L'$

and f is implicitly log-space computable.

and $|f(x)| \leq cn^d$

implicit log-space computable

$x \longrightarrow$ [M_f] $\longrightarrow f(x)$; prints $f(x)$ bit by bit

DTM

DTM takes log-space to compute $f(x)$

1) $i = 1$
2) $i \le |f(x)|$ → using $(x, i) \in L_1$
3) print 0/1 → using $(x, i) \in L_2$ ↔ 1 if Yes, 0 if No
4) i++

L_1, L_2 are log-space computable

Space by counter = $\log |f(x)| \le \log(cn^d) = \log(c|x|^d)$

- Thus, DTM takes log-space for computation of $f(x)$.

★ $L_1 \le L_2$ & $L_2 \le L_3$, $\Rightarrow L_1 \le L_3$ in NL domain? Yes

$L_1 \le L_2$: $\exists f(x)$, implicitly log space computible ; $|f(x)| \le cn^d$

$x \in L_1 \Leftrightarrow f(x) \in L_2$

$L_2 \le L_3$: $\exists g(x)$, ——— ,, ——— ,, ——— ; $|g(x)| \le en^\beta$

$x \in L_2 \Leftrightarrow g(x) \in L_3$

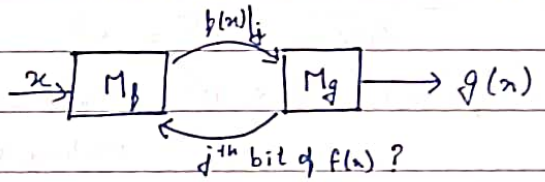
- If we use old proof :

$$x \mapsto f(x) \mapsto g(f(x))$$

$x \longrightarrow$ [M_f] $\xrightarrow{f(x)}$ [M_g] $\longrightarrow g(x)$

M_f prints $f(x)$ bit-by-bit. So, we need to store $f(x)$ which requires poly. space but we only allow log-space.

- With slight modification, we can achieve this.



- We start with $i = 1$

Check $(f(x), i) \in L_{1,g}$

↳ To check this, it will need some j^{th} bit of $f(x)$ at any step, which can be provided by M_f any time.

Thus, we need not require to store $f(x)$ anymore.

- M_g can't change $f(x)$ value, since input-tape is read only. Thus, every time we need original $f(x)$ value only.

Hence, $x \in L_1 \Leftrightarrow g(f(x)) \in L_3$; $g(f(x))$ is implicit log space computable.

$|g(f(x))| \le e(cn^d)^\beta = c'n^{d'}$

$\Rightarrow L_1 \le L_3$

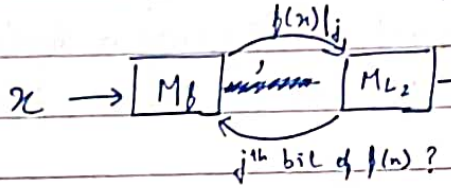
★ $L_1 \leq L_2$ & $L_2 \in \mathbf{L} \Rightarrow L_1 \in \mathbf{L}$

$L_1 \leq L_2$: $\exists f(x)$, implicitly log space computable ; $|f(x)| \leq cn^d$

$x \in L_1 \Leftrightarrow f(x) \in L_2$

L_2 take $e \log(f(n))$ space

$\leq e \log(cn^d) = e' + ed \log|x| = O(\log|x|)$



• We can use similar argument and hence $L_1 \in \mathbf{L}$.

★ NL - complete

1) $L \in NL$

2) $\forall L' \in NL$, $L' \leq L$ (using red^n def. in domain of NL)

Then, we say L is NL complete.

★ PATH is NL-complete

• We already proved PATH $\in$ NL

• Take $L \in NL \rightarrow$ NDTM N, $L(N) = L$, takes $\log n^c$ space.

$x \in L$, $x \xrightarrow{f} \langle G_{N,x}, c_0, c_f \rangle$

Size $= 2^{\log n^c} = n^c$

$L_{1,f}$: $(x, i) \in L_{1,f}$? $\Rightarrow$ check $i \leq n^c$ can be done in log space by writing them in binary

$L_{2,f}$: $(x, i) \in L_{2,f}$? $f(x)|_i$?

Enumerate configurations in pair (i, j)

(i, j) represents two conf. C_i & C_j and checking if edge exists b/w them can be done by using N.

↳ This too can be done in log-space.

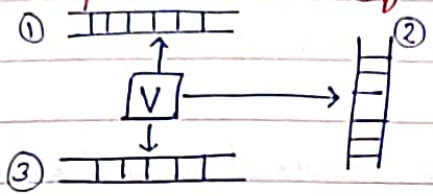
$\Rightarrow$ $x \in L \Leftrightarrow f(n) = \langle G_{N,x}, c_0, c_f \rangle \in$ PATH

$|f(n)| \leq |x|^c$ & $f(n)$ is implicit log-space computable

$\Rightarrow$ $L \leq$ PATH, $\forall L \in NL$

$\Rightarrow$ PATH is NL-complete.

★ Certificate - verifier def^n in domain of NL.



① Read-only input tape
② Read-write work tape
③ Read-once certificate tape
(moving Left not allowed)
{S/R}

★ PATH ∈ NL using cert-ver def^n.

Certificate is path from 's' to 't' in G.

From certificate, read first vertex and write it in work tape. Check it it is 's'.

Read second vertex & store. Check for edge b/w first & second vertex. Read third vertex & overwrite on first. Check for edge b/w 2nd & 3rd vertex. Continue till last vertex & compare it with 't'.

Hence, PATH ∈ NL. (Certificate is being read once only).

→ We can't see old non-deterministic choices made if we have read-once certificate and can only store constant length of certificate in work-tape. Thus, it prevents extra advantage.

→ In NP, it doesn't make sense as everything is polynomial time & space.

★ Both def^n of NL are equivalent.

i) L ∈ NDTM log-space def^n → N, L(N) = L

⇒ The non-deterministic choices made by N on correct path can be used as certificate.

⇒ L ∈ cert-ver. log space def^n

ii) L ∈ cert-ver. log space def^n → $\exists y$ V, V(x, y) = 1

⇒ NDTM N guess the certificate y and give it to verifier.
So, if certificate is correct, verifier will say Yes
It's NDTM, so mistakes are allowed.

⇒ L ∈ NDTM log-space def^n

★ $\overline{PATH} \in NL$

$\overline{PATH} = \{ \langle G, s, t \rangle \mid$ No path exists from s to t in $G \}$

→ Cert.-Ver. def^n of NL

In worst case, certificate will be all vertices (visited from s)

Size $= O(n \log n)$ but not $O(c \log n)$ for some constant c

- We want our computations to be $O(c \log n)$ in space.

→ Define $C_i = \{ v \mid s \rightsquigarrow v$ in $\leq i$ steps $\}$

- We don't know full set C_i, but $|C_i|$.

i) $v \in C_i$? certificate = path $s \rightsquigarrow v$ of $\leq i$ steps

ii) $v \notin C_i$? $s \rightsquigarrow u_1 \# s \rightsquigarrow u_2 \# s \rightsquigarrow u_3 \# \cdots$

$|C_i|$ no. of paths → need a counter

Check all paths one by one ↪ $\leq i$ steps → need $2^{\log n}$ counter
$u_j \neq v$

But paths could be repeated!

We ensure $u_1, u_2, \ldots u_j$ are in lexicographic order.

Thus, we only need to check $u_j > u_{j-1}$, then only path is valid.

- All this process will take constant log-space & computations are local. To check $u_j > u_{j-1}$, we store u_{j-1} (only need to store 1 vertex)

→ $|C_0| = 1$, If we can compute $|C_i| \rightarrow |C_{i+1}|$;

we can compute $|C_n|$

and just need to check $t \notin C_n$.

⇒ No path exists from $s \rightsquigarrow t$ of $\leq n$ steps;

and path length can't be more than n.

- we need some certificate to get $|C_{i+1}|$ from $|C_i|$.

For each vertex u_j ↗ $u_j \in C_{i+1} \rightarrow C_{+,u_j} = s \rightsquigarrow u_i$ of $\leq i+1$ steps

↘ $u_j \notin C_{i+1} \rightarrow C_{-,u_j} = s \rightsquigarrow v_1 \# s \rightsquigarrow v_2 \# \cdots$,

i.e., all paths of C_i

For C_{-,u_j}, we check if any of $v_k = u_j$ or any of v_k's neighbour $= u_j$.

If none, then $u_j \notin C_{i+1}$

- Final certificate : $C_0 \to C_1 \,||\, C_1 \to C_2 \,||\, \dots \,||\, C_{n-1} \to C_n \,||\, t \notin C_n$

  C_1: $v_1 \in C_2 \,\#\, v_2 \in C_2 \,\#\, \dots \,\#\, v_n \in C_2$, C_0/C_-

  C_n: $S \to u_1 \# S \to u_2 \# \dots \# S \to u_{|C-1|}$

  Size of certificate = poly. in terms of input

  $\Rightarrow$ $\overline{PATH} \in NL$

  Hence Proved

★ $NL = Co\text{-}NL$

$Co\text{-}NL = \{ L \mid \bar{L} \in NL \}$, i.e; compliment of all languages in NL

★ PATH is NL - complete.

$\forall L \in NL$, $L \leq_f PATH \Rightarrow \bar{L} \leq_f \overline{PATH}$

$\Rightarrow \forall \bar{L} \in Co\text{-}NL$, $\bar{L} \leq \overline{PATH}$

$\Rightarrow$ $\overline{PATH}$ is Co-NL complete

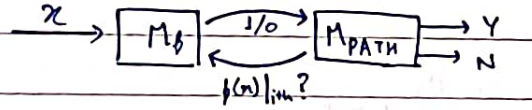
★ $\overline{PATH}$ is Co-NL Complete.

→ Take $L \in NL \Rightarrow \bar{L} \in Co\text{-}NL$

$L \leq_f PATH \Rightarrow \bar{L} \leq_f \overline{PATH}$

But $\overline{PATH} \leq PATH$, since PATH is NL - complete

$\Rightarrow \bar{L} \leq_f PATH$



$\Rightarrow \bar{L} \in NL$

$\Rightarrow \forall L \in NL$, $\bar{L} \in NL \Rightarrow NL \subseteq Co\text{-}NL$

$\bar{L} \in Co\text{-}NL$

→ $L \in Co\text{-}NL \Rightarrow \bar{L} \in NL$ but $NL \subseteq Co\text{-}NL$

$\Rightarrow \bar{L} \in Co\text{-}NL \Rightarrow Co\text{-}NL \subseteq NL$

Hence, $NL = Co\text{-}NL$

Hence Proved

★ $L \subseteq NL = Co\text{-}NL \subseteq P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXP \subseteq NEXP$

$SCG_G = \{ G \mid G$ is strongly connected digraph $\}$

↳ $\forall\ u, v \in V$, $\exists$ path $u \rightsquigarrow v$

SCG_G is NL-complete.

→ $SCG_G \in NL$

We can give paths blw every pair of vertices in lexicographic order as a certificate.

$C = v_1 \rightsquigarrow v_2 \# v_1 \rightsquigarrow v_3 \# \cdots \# v_1 \rightsquigarrow v_n \,||\, v_2 \rightsquigarrow v_1 \# v_2 \rightsquigarrow v_3 \cdots \,||\, \cdots \# v_n \rightsquigarrow v_{n-1}$

Check all paths; each computation is local with some constant no. of counters. ⇒ Require $O(\log n)$ space.

Hence, $SCG_G \in NL$

→ PATH $\leq$ SCG_G $\{ \forall L \in NL,\ L \leq PATH \leq SCG_G \}$

$\langle G, s, t \rangle \longrightarrow G'$

To construct G' :

i) Convert all edges $(u, v) \in E_G$ to two directed edges $u \to v$ and $v \to u$.

ii) Add addition edges :-

$\forall\ v \in V_G - \{s, t\}$, add edge $t \to v$ and $v \to s$.

- If $\langle G, s, t \rangle \in$ PATH

⇒ $\exists$ a path from s to t in G and $t \to s$

⇒ $\exists$ a dis-path from $s \rightsquigarrow t$, in G' due to converted edges.

$\forall\ v \in V - \{s, t\}$, there is path $v \to s$ & $t \to v$ due to additional edges.

$\forall\ a, b \in V$, $a \to s \xrightarrow{E_G} t \to b \Rightarrow a \rightsquigarrow b$

⇒ $G' \in SCG_G$

- If $\langle G, s, t \rangle \notin$ PATH ⇒ No path exists from s to t in G.

⇒ No path in G' due to only converted edges.

Also, new edges are from $v \to s$ & $t \to v$, no outgoing edge is created from s and no incoming edge is created to t.

Hence, path is not possible from $s \rightsquigarrow t$.

⇒ $G' \notin SCG_G$ ⇒ Hence, PATH $\leq$ SCG_G ⇒ SCG is NL-complete

★ Polynomial Hierarchy

→ $L \in P$

$x \in L \iff \underbrace{M(x) = 1}_{P(x) = 1}$ → Property

$L \in NP$

$x \in L \iff \exists y, \underbrace{M(x,y) = 1}_{P(x,y) = 1}$

- SAT : $\exists$ U $2^{p(x)}$ → we want optimal solution.
- Shortest-Path = $\{ \langle G, s, t, k \rangle \mid$ path $s \to t$ of $\leq k$ steps in $G \}$

  $P(n)$: $\exists$ path ; $|path| \leq k$, $(path)_{first} = s$, $(path)_{last} = k$

  We can find length of shortest path in poly. time if shortest Path is solvable in poly. time.

- Largest-Ind Set = $\{ \langle G, k \rangle \mid G$ has an ind. set of size $k \}$

  $\exists I$ $\forall I'$, I & I' are ind. set of G, $|I| \geq |I'|$

  Not NP-property, since this thing can't be handled by DTM in poly. time.

→ Such properties are called Σ_2^p properties.

Remaining quantifiers are alternating ← Σ : first quantifier is $\exists$

2 : No. of quantifiers

p : polynomial

→ NP-properties are Σ_1^p and P-properties are Σ_0^p properties.

★ Σ_i^p property : $\exists$ DTM poly-time M and a polynomial $q(x)$ s.t.

$L \in \Sigma_i^p$ iff

$x \in L \iff \exists y_1 \forall y_2 \exists y_3 \cdots Q_i y_i$, $M(x, y_1, y_2, \cdots y_i) = 1$

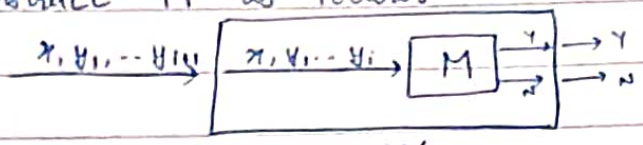
and $|y_1| \leq q(x)$, $|y_2| \leq q(x)$, $\cdots$ $|y_i| \leq q(x)$

$Q_i \begin{cases} \exists, & \text{if } i \text{ is odd} \\ \forall, & \text{otherwise} \end{cases}$

★ $\Sigma_i^p \subseteq \Sigma_{i+1}^p$

- $L \in \Sigma_i^p \Rightarrow \exists M$ and q, $x \in L \iff \exists y_1 \cdots Q_i y_i$ $M(x, y_1, \cdots y_i) = 1$ and $|y_k| \leq q(n)$ $\forall k = 1 \cdots i$

- We construct M' as follows :



$\Rightarrow M'(x, y_1, \dots y_{i+1}) = M(x, y_1 \dots y_i)$

- $x \in L \iff \exists y_1 \forall y_2 \dots Q_i y_i \; M(x, y_1, \dots, y_i)$

$\iff \exists y_1 \dots Q_i y_i Q_{i+1} y_{i+1} \; M'(x, y_1, \dots y_{i+1})$

For any value of y_{i+1}, this will hold for same values of $y_1 \dots y_i$ & $|y_k| \le p(x) \;\forall\; k = 1, \dots, i+1$

$\Rightarrow L \in \Sigma_{i+1}^p$

★ Π_i^p property : $\exists$ DTM poly time M and poly. p s.t.

$x \in L \iff \forall y_1 \dots Q_i y_i \; M(x, y_1, \dots y_i) = 1$ and $|y_k| \le p(x) \;\forall\; k = 1, \dots, i$

Q_i : $\forall$, if i is odd ; $\exists$, otherwise

→ same as Σ_i^p, but quantifier starts from $\forall$

$L \in \Pi_i^p$ iff

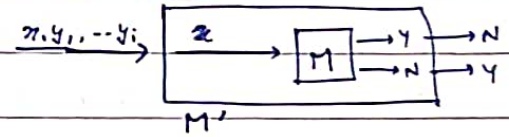
★ $\Pi_i^p = co\text{-}\Sigma_i^p = \{ \bar{L} \mid \bar{L} \in \Sigma_i^p \}$

- $L \in \Pi_i^p \iff \exists M, p$ s.t. $x \in L \iff \forall y_1 \dots Q_i y_i \; M(x, y_1 \dots y_i) = 1$
- ~~Construct M as follows~~

$x \notin L \iff \neg \left(\forall y_1 \dots Q_i y_i \; M(x, y_1 \dots y_i) = 1 \right)$

$x \notin \bar{L} \iff \exists y_1 \dots \bar{Q}_i y_i \; M(x, y_1, \dots y_i) = 0$

$\iff x \in \bar{L}$



$\Rightarrow x \in \bar{L} \iff \exists y_1 \dots \bar{Q}_i y_i \; M'(x, y_1 \dots y_i) = 1$

$\Rightarrow \bar{L} \in \Sigma_i^p \Rightarrow L \in co\text{-}\Sigma_i^p$

Hence, $\Pi_i^p \subseteq co\text{-}\Sigma_i^p$

Similarly, we can show $co\text{-}\Sigma_i^p \subseteq \Pi_i^p$

Hence $\Pi_i^p = co\text{-}\Sigma_i^p$

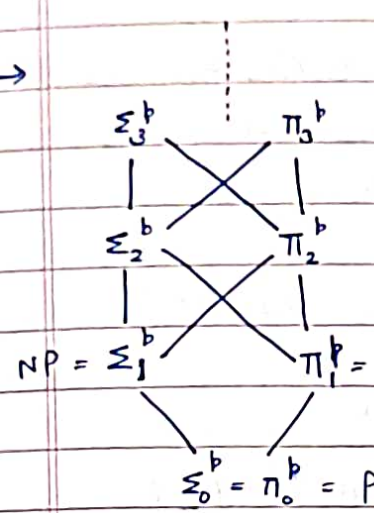
★ $\Pi_i^p \subseteq \Pi_{i+1}^p$

$L \in \Pi_i^p \Rightarrow \bar{L} \in \Sigma_i^p \Rightarrow \bar{L} \in \Sigma_{i+1}^p \Rightarrow L \in \Pi_{i+1}^p$

★ $\Sigma_i^b \subseteq \Pi_{i+1}^b$ and $\Pi_i^b \subseteq \Sigma_{i+1}^b$

- Similar to proof of $\Sigma_i^b \subseteq \Sigma_{i+1}^b$, just ignore first argument instead of last.

→



- This is called polynomial hierarchy.
- This is an infinite hierarchy.

$$PH = \bigcup_{i \geq 0} \Sigma_i^b = \bigcup_{i \geq 0} \Pi_i^b = \bigcup_{i \geq j} \Sigma_i^b, \text{ for any } j \geq 0$$

complexity class

- If any links breaks (i.e., proved equal), complete poly. hierarchy collapses into P.

→ Poly. hierarchy collapses at level i means $\boxed{PH = \Sigma_i^b = \Pi_i^b}$

- $\Pi_i^b \subseteq \Sigma_{i+1}^b \subseteq PH \subseteq \Sigma_i^b \Rightarrow \Pi_i^b \subseteq \Sigma_i^b$ $\quad \forall j \geq i > 0$
- $L \in \Sigma_i^b \Rightarrow \bar{L} \in \Pi_i^b \Rightarrow \bar{L} \in \Sigma_i^b \Rightarrow L \in \Pi_i^b$

$\Rightarrow \Sigma_i^b \subseteq \Pi_i^b$

Hence, $\Pi_i^b = \Sigma_i^b = PH$

★ $\Sigma_i^b = \Pi_i^b \Rightarrow PH = \Sigma_i^b \quad ; \; i > 0$

Let $L \in \Sigma_{i+1}^b$

$$x \in L \Leftrightarrow \exists y_1 \underbrace{\left(\forall y_2 \ldots Q_i y_i \, Q_{i+1} y_{i+1} \; M(x, y_1 \ldots y_{i+1}) = 1\right)}_{P(y_1)}$$

and $|y_j| \leq s(x) \quad \forall j = 1 \ldots i+1$

Define $L' = \{ (x, u) \mid x \in L, \; \underbrace{\forall y_2 \ldots Q_{i+1} y_{i+1} \; M(x, u, y_2 \ldots y_{i+1}) = 1}_{P(u)} \}$

$(x,u) \in L' \Leftrightarrow P(u) : \underbrace{\forall y_2 \ldots Q_{i+1} y_{i+1}}_{i} \; M(x, u, y_2 \ldots y_{i+1}) = 1$

$\Rightarrow L' \in \Pi_i^b$

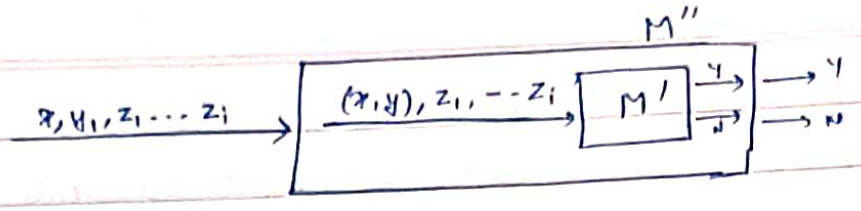
$\Rightarrow L' \in \Sigma_i^b \quad (\because \Sigma_i^b = \Pi_i^b)$

$(x,u) \in L' \Leftrightarrow \exists z_1, \ldots Q_i z_i \; M'((x,u), z_1, \ldots z_i) = 1$

We can say that:

$$x \in L \Leftrightarrow \exists y_1 \; (x, y_1) \in L'$$

$$\Leftrightarrow \exists y_1 \exists z_1 \ldots Q_i z_i \; M''(x, y_1, z_1, \ldots z_i) = 1$$



$x \in L \iff \exists (y_1, z_1)\ \forall z_2 \ldots Q_i z_i\ M'''(x, (y_1, z_1), z_2, \ldots z_i) = 1$

↳ This will increase size of argument

change $q(n) \rightarrow q'(n) = 2q(n)$

Hence ~~x ∈ L~~ *

$\Rightarrow$ $\exists$ DTM M''' and $q'(n)$ s.t. $x \in L \iff \exists y_1' \forall z_2 \ldots Q_i z_i\ M'''(x, y_1', z_2 \ldots z_i) = 1$

$\Rightarrow L \in \Sigma_i^p$

$\Rightarrow \Sigma_{i+1}^p \subseteq \Sigma_i^p$ and we know $\Sigma_i^p \subseteq \Sigma_{i+1}^p$

$\Rightarrow \Sigma_i^p = \Sigma_{i+1}^p \Rightarrow$ Inductively we can show $\forall j \geq i$

$\Rightarrow i \neq 0$ since we need one argument to merge in Σ_i^p.

★ $\Sigma_i^p = \Sigma_{i+1}^p \Rightarrow PH = \Sigma_i^p$, $i \geq 0$

- $\Pi_i^p \subseteq \Sigma_{i+1}^p = \Sigma_i^p$
- $L \in \Sigma_i^p \Rightarrow \bar{L} \in \Pi_i^p \subseteq \Sigma_{i+1}^p = \Sigma_i^p \Rightarrow \bar{L} \in \Sigma_i^p \Rightarrow L \in \Pi_i^p$

$\Rightarrow \Sigma_i^p \subseteq \Pi_i^p$

$\Rightarrow \Pi_i^p = \Sigma_i^p$

$\Rightarrow PH = \Sigma_i^p$, $i > 0$ (using prev. thm.)

Hence Proved

★ $P = NP \Rightarrow PH = P$ (i.e., for $i = 0$)

- $co\text{-}NP = co\text{-}P = P = NP \Rightarrow co\text{-}NP = P$

$\Rightarrow NP = co\text{-}NP \Rightarrow PH = NP$ (using above thm.)

$\Rightarrow PH = P$

$\therefore \Sigma_i^p = \Sigma_{i+1}^p \Rightarrow PH = \Sigma_i^p$ for all $i \geq 0$

Hence Proved

★ $\overline{SAT}$ is co-NP-complete.

$\forall L \in NP$, $L \leq_p SAT$

$\Rightarrow x \in L \Leftrightarrow \varphi_x$ is sat.

$\Rightarrow x \notin L \Leftrightarrow \varphi_x$ is not sat.

$\Rightarrow x \notin \bar{L} \Leftrightarrow \varphi_x$ is not sat.

$\Rightarrow \bar{L} \leq_p \overline{SAT}$ and $\bar{L} \in$ co-NP $\forall L \in NP$

$\Rightarrow \overline{SAT}$ is co-NP complete.

★ Replace any TM with a SAT formula, thus we can replace it as follows:

$L \in \Sigma_i^p$, $x \in L \Leftrightarrow \exists u_1 \forall u_2, \ldots Q_i u_i \; \varphi_x(u_1, u_2, \ldots u_i) = 1$

★ $\phi \in \Sigma_i^p SAT \Leftrightarrow \exists u_1 \ldots Q_i u_i \; \phi(u_1, \ldots u_i) = 1$

★ $\Sigma_i SAT$ is Σ_i^p-complete.

• $\phi \in \Sigma_i SAT \Leftrightarrow \exists u_1 \ldots Q_i u_i \; \phi(u_1 \ldots u_i) = 1$

$\Leftrightarrow \exists u_1 \ldots Q_i u_i \; M(\phi, u_1, \ldots u_i) = 1$

where, DTM M checks satisfiability of given ϕ on the given assignment, i.e., $\phi(z) = 0/1$.

$\Rightarrow \Sigma_i SAT \in \Sigma_i^p$

• $\forall L \in \Sigma_i^p$, we have to show $L \leq_p \Sigma_i SAT$

$L \in \Sigma_i^p$, $x \in L \Leftrightarrow \exists u_1 \ldots Q_i u_i \; M(x, u_1, \ldots u_i) = 1$

$\Leftrightarrow \exists v_1 \ldots Q_i v_i \; \phi_x(v_1, \ldots v_i) = 1$

$x \mapsto \phi_x$ using same reduction algo. as SAT

$\rightarrow x \in L \Leftrightarrow \phi_x \in \Sigma_i SAT$

$\Rightarrow L \leq_p \Sigma_i SAT$

$\Rightarrow \Sigma_i SAT$ is Σ_i^p-hard

$\Rightarrow \Sigma_i SAT$ is Σ_i^p-complete

Hence Proved

But we can't say $\phi_x \in SAT$ because machine M can't check for $\forall v_k$ variables and also we can't concatenate $\forall$ variables since $\forall \exists \forall$ order is not changeable

Hence, $\phi_x \in \Sigma_2^p SAT$ also can't be shown.

★ $\Pi_i SAT$ is Π_i^p-complete.

$\phi \in \Pi_i SAT \Leftrightarrow \forall u_1 \ldots Q_i u_i \; \phi(u_1 \ldots u_i) = 1$

★ If PH has a complete problem, then PH collapses.

• Let L be PH-complete and $L \in \Sigma_i^p$

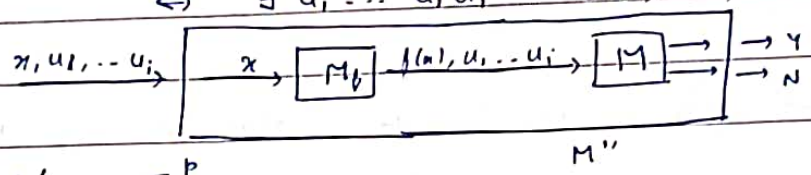
For some $L' \in \Sigma_{i+1}^p$, $L' \leq_p L$ since L is PH-complete

$$x \mapsto f(x)$$

$$x \in L' \Leftrightarrow f(x) \in L$$

$$\Leftrightarrow \exists u_1 \cdots Q_i u_i \; M(f(x), u_1, \ldots u_i) = 1$$

$$\Leftrightarrow \exists u_1 \ldots Q_i u_i \; M''(x, u_1, \ldots u_i) = 1$$



$$\Rightarrow L' \in \Sigma_i^p$$

Hence, $\forall L' \in \Sigma_{i+1}^p$, $L' \in \Sigma_i^p$

$$\Rightarrow \Sigma_{i+1}^p \subseteq \Sigma_i^p$$

$$\Rightarrow \Sigma_i^p = \Sigma_{i+1}^p$$

$$\Rightarrow PH = \Sigma_i^p$$

★ $PH \subseteq PSPACE$

• $L \in \Sigma_i^p$, $x \in L \Leftrightarrow \exists u_1 \cdots Q_i u_i \; M(x, u_1 \ldots u_i) = 1$

Machine M' assigns values for $u_1 \ldots u_i$ one-by-one & check using M

M' takes poly-SPACE but EXP-time.

Hence, $L \in PSPACE$.

$\Rightarrow PH \subseteq PSPACE$

★ $PH = PSPACE \Rightarrow PH$ collapses

• QBF is PSPACE complete

$\Rightarrow$ QBF is PH complete

$\Rightarrow$ PH collapses.

★ $\Sigma_2^p = NP^{SAT}$ (oracle TM)

$\exists$: Non-determinism takes care of it

$\forall$: by SAT oracle

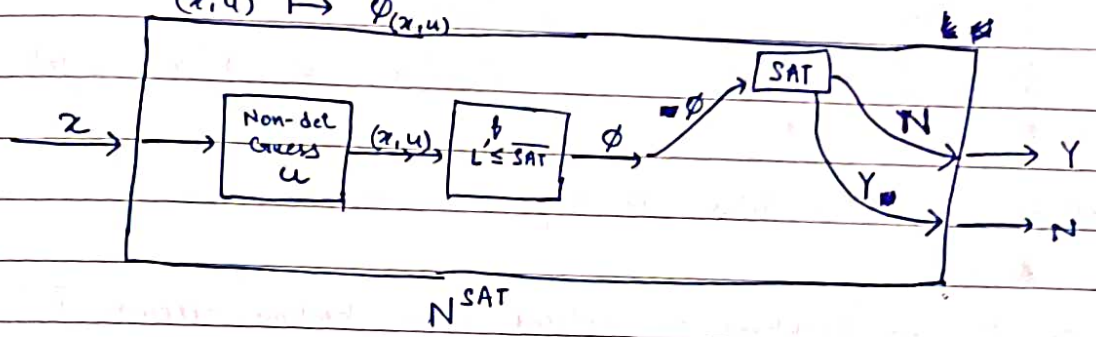
- $L \in \Sigma_2^p$, $x \in L \Leftrightarrow \exists u \, \forall v \; M(x,u,v) = 1$

  $\Leftrightarrow \exists u \, \forall v \; \phi_x(u,v) = 1$

  $(x,u) \in L' \Leftrightarrow \forall v \; \phi'_x((x,u),v) = 1$

  $L' \in \Pi_1^p$

  $L' \leq \overline{SAT}$



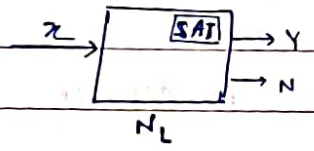
$x \in L \Leftrightarrow N^{SAT}(x) = 1$

$x \in L \Leftrightarrow$ guess u is correct $\Leftrightarrow (x,u) \in L' \Leftrightarrow \phi_{x,u} \in \overline{SAT} \Leftrightarrow \phi_{x,u} \notin SAT \Leftrightarrow N^{SAT}(x) = 1$

$\Rightarrow L \in NP^{SAT}$

$\Rightarrow \Sigma_2^p \subseteq NP^{SAT}$

- $L \in NP^{SAT}$,



① Oracle : $\phi_1 \in SAT$? $\rightarrow$ 0/1

$\phi_2 \in SAT$? $\rightarrow$ 0/1

⋮

② Guess

$x \in L \Leftrightarrow \underbrace{\exists m}_{\text{non-det moves of } N_L} \; \underbrace{\exists \phi_1, \phi_2 \ldots \phi_k}_{\text{oracle queries}} \; \underbrace{\exists a_1, \ldots a_k}_{\text{oracle answers}} \; \underbrace{\exists u_1, \ldots u_k}_{\text{If } a_i = 1 \text{ satisfying assignment}} \; \underbrace{\forall v_1 \ldots v_k}_{\text{If } a_i = 0, \text{ for all assignment } \phi_k \text{ is FALSE}}$

$$\left\{ \forall i \; (a_i = 1 \wedge \phi_i(u_i)) \vee (a_i = 0 \wedge \neg \phi_i(v_i)) \right\} \; M(\ldots) = 1$$

- All non-deterministic choices made by N_L is captured by $\exists$ part and oracle answers are verified using $\exists$ & $\forall$ for Yes/No respectively.
- Value of k can be guessed by running time of N_L.

$\Rightarrow L \in \Sigma_2^p$

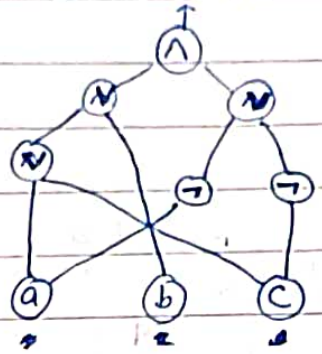
$\Rightarrow NP^{SAT} \subseteq \Sigma_2^p$

Hence, $\Sigma_2^p = NP^{SAT}$

Hence Proved

★ Boolean Circuits

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



- Take fixed no. of inputs (3 here)
- Represented as $C_3(a,b,c)$
- Similar to Algorithm, but input length is fixed.

$$C_n(x) \equiv A(x), \quad |x| = n$$

- Ordering of variables is important in notation.

→ Can decision problems be solved using boolean circuits ?

- One circuit won't work, but family of circuits for all possible input lengths.

$$L = \{00, 0000, \ldots\} \equiv \{C_n\}_n$$

↓ infinite family

$$x \in L \iff C_{|x|}(x) = 1$$

★ **SIZE(T(n))** : A $L \in SIZE(T(n))$ iff $\{C_n\}_n$ solves L and $|C_n| \leq T(n)$, where $|C_n|$ is no. of gates in circuit.

★ **P/poly** : $\bigcup_{c \geq 0} SIZE(n^c)$; poly. size circuits

★ L is a unary language ⇒ $L \in P/poly$

$1^n \in L$, we can construct C_n as all inputs joined by 'and' gates.

$$1^n = a_1 a_2 \ldots a_n \equiv a_1 \wedge a_2 \wedge \ldots \wedge a_n$$

$$C_n = a_1 \wedge a_2 \wedge \ldots \wedge a_n$$

$x \in L$, $x = 11\ldots1$ ⇒ $C_n(x) = 1$

$x \notin L$, some $a_i = 0$ ⇒ $C_n(x) = 0$

$$x \in L \iff C_{|x|}(x) = 1$$

⇒ $L \in P/poly$

Hence Proved

★ P/poly can solve undecidable languages

- Construct $U_H = \{1^n \mid x \in HALT, \langle x \rangle = n\}$

  U_H is unary and undecidable

  $\Rightarrow U_H \in P/poly$

- We can play such tricks with all languages.

★ $P \subsetneq P/poly$

$L \in P$, $\exists M, q$ s.t. $x \in L \Leftrightarrow M(x) = 1$ in time $q(x)$

$M(x) \mapsto \varphi_{M,x}$

But we need to make some formula for all x, $|x| = n$.

$\varphi_{M,x} = \varphi_M(x)$, it works for all x, $|x| = n$

$M(x) \mapsto \varphi_M(x)$ considering x as variable.

Size of φ_M is poly., it can be converted into circuit

$\Rightarrow x \in L \Leftrightarrow C_{|x|}(x) = 1$

$\exists \{C_n\}_n$ to solve L

$\Rightarrow P \subseteq P/poly$ $P \neq P/poly$, since P/poly can solve an undecidable language.

Hence Proved

★ P-uniform circuit families

→ If an algorithm A on input 1^n can generate circuit C_n (in poly. time), then $\{C_n\}_n$ is called P-uniform circuit family.

$A_p(1^n) \Rightarrow C_n \Leftrightarrow \{C_n\}_n$ is P-uniform

Eg: For any $L \in P$, we have SAT-red^n algo. to generate circuit.

For U_{HALT}, we can't have such algorithm.

★ $L \in P \Leftrightarrow L$ has a P-uniform circuit family

- $L \in P \Rightarrow$ It has SAT-red^n algo. to generate φ_M in poly. time

  $\Rightarrow \{C_n\}_n$ is P-uniform

- L has P-uniform circuit family, we can have an algo. to solve L.

  ↳ $\exists A(1^n) \to C_n$

1) $|x| = m$
2) $A(1^m) \to C_m$
3) $C_m(x) = 0/1$ output $\Rightarrow L \in P$

Hence Proved

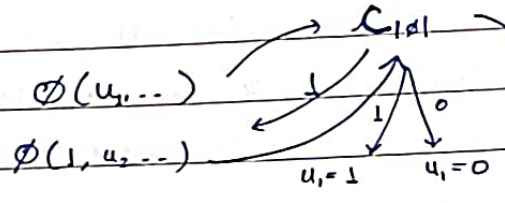
→ Algorithms are generalization of circuits family.
The size of an algorithm is fixed and behave similar irrespective of input size, whereas in circuit family we have infinite number of circuits which can completely differ for different input size.

★ NP ⊆ P/poly (Assumtion)

$L \in NP \Rightarrow \exists V, q$ s.t $x \in L \Leftrightarrow \exists y\ V(x,y) = 1$ and $|y| = q(x)$

$SAT \in NP \Rightarrow \phi \in SAT \Leftrightarrow C_{|\phi|}(\phi) = 1$
↳ it tell ∃ an assignment but doesn't tell what?

$\Rightarrow SAT \in P/poly$ $\phi(u_1, \ldots u_n)$



then repeat for all variables

We have $A(\phi) \rightarrow z_\phi$, a satisfying assignment of ϕ

$C_{|\phi|}(\phi) \rightarrow z_\phi$

$\Rightarrow \exists \{C_n\}_n$, given ϕ, $C_{|\phi|}(\phi) \Rightarrow z_\phi$

$\Rightarrow$ SAT can be solved in poly. time

★ $NP \subseteq P/poly \Rightarrow PH = \Sigma_2^P$

To show: $\Pi_2 SAT \in \Sigma_2^P$

$\phi \in \Pi_2 SAT \Leftrightarrow \forall u\ \exists v\ \phi(u,v) = 1$

$\Rightarrow (\phi, u) \in L' \Leftrightarrow \exists v\ \phi(u,v) = 1$

↓ To find v, we can have algo (as above)

$\Rightarrow L' \in NP \Rightarrow L' \in P/poly$ ↗ works for all u, of same length

$\Rightarrow (\phi, u) \in L' \Leftrightarrow \phi(u, C_{|\phi|}(\phi(u,v))) \Leftrightarrow C_{|\phi|}(\phi)$

$\Rightarrow \phi \in \Pi_2 SAT \Leftrightarrow \exists C_{|\phi|}\ \forall u\ \phi(u, C_{|\phi|}(\phi)) = 1$

$\Rightarrow \phi \in \Sigma_2^P$

$\Rightarrow \Pi_2 SAT \in \Sigma_2^P \Rightarrow PH = \Sigma_2^P$