COMPUTATIONAL COMPLEXITY

→ **Set** – A collection of objects, none of which is the set itself.

$A = \{1, a, \frac{1}{3}, \not{A}\}$

Ordinary Sets – Doesn't contain itself.

Extra-ordinary Sets – Contains itself.

Define $\mathcal{S}$ = {Set of all ordinary sets}

If $\mathcal{S}$ is ordinary, it should contain itself ⇒⇐

If $\mathcal{S}$ is extra-ordinary, $\mathcal{S}$ must be present in itself ⇒⇐

Thus, it leads to a paradox. $\mathcal{S}$ is neither ordinary nor extra-ordinary.

⇒ $\mathcal{S}$ is not a set. ⇒ Extra-ordinary sets are not sets.

⇒ New def^n of sets.

→ **Computation** –

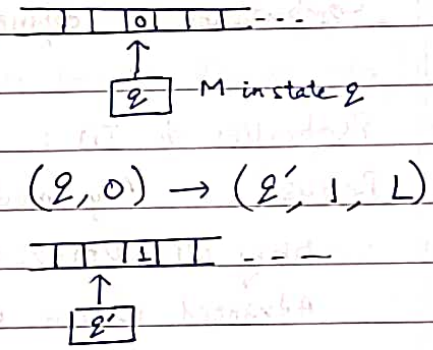
- wrt Turing Machines

Σ – set of alphabets

Q – set of states

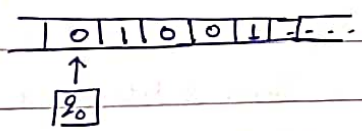
δ – set of transitions

$q_0 \in Q$ – initial state



Eg: Change all 0's to 1's and vice-versa.

* At the start, our head ptr. is located at left-end of input and machine is in initial state.



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

$Q = \{q_0, q_f\}$

$(q_0, 0) \rightarrow (q_0, 1, R)$

$(q_0, 1) \rightarrow (q_0, 0, R)$

$(q_0, B) \rightarrow (q_f, B, S)$ → Not necessary

B – Blank, S – Stay

- If there is no instruction for turing machine, it will stop.
- Whenever machine enters final state, it halts.

→ **Formal Def^n** :

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

(Σ : input alphabets; Γ : all alphabets)

$\Sigma \subseteq \Gamma$

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

$B \in \Gamma$ (B : Blank)

→ **Church Turing Thesis** :

- Any physically realisable computational model can be simulated by a turing machine.
- **Strong version** : with polynomial slowdown

$t \to t^c$ in turing machine for constant c

But quantum computers defy strong version of this thesis.

→ **Computation** is defined as whatever a ~~a~~ turing machine can do

→ **Properties of TM** :

→ **Robust** : Any modifications to T.M. (like increasing no. of tapes, or taking all Utf-8 characters) affect a T.M. only polynomially

Advanced version of T.M takes T time → Basic version of T.M. will take T^c time, for constant c

→ **M = 0-1 string** : Code of a T.M. can be expressed as a 0-1 bit string.

Integer (n) – normal binary

Tuple (<n,m>) – Use 00 for 0, 11 for 1, 01 for ,

Eg : $3 \to 011$

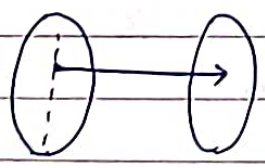
Eg : $<3,4> \to 11\ 11\ 01\ 11\ 00\ 00$

Take 2 bits at a time

If we encounter 10, encoding is incorrect.

$M \to \alpha_M$

→ For given TM, there exist infinitely many bit strings of it (for a fixed encoding-decoding algorithm)



$M \xrightarrow{f} \alpha_M \xrightarrow{g} \alpha'_M \xrightarrow{h} \alpha'_M 01 xxx\ldots$

(01 : Separator (code-ended); xxx… : garbage)

$x \to \alpha'_M 01 x$ → i) find separator and decode

↳ ii) extract this to get M

* **Universal Turing Machine (U)** : Simulate any TM on an input.
Given (x, α) as input to U, it simulates M_x on x.
If running time of M_x was $T(|x|)$, then running time of U is $O(T(|x|) \log T(|x|))$.

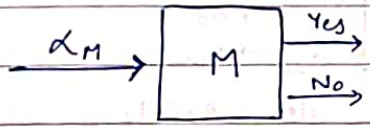
$M \rightarrow$ inf. many α_M

→ **For every string $x \rightarrow M_x$**

We can have a T.M. that does nothing.

$$M_0 = \{[Q_0], \{0,1\}, \{0,1\}, Q_0, Q_0, \emptyset\}$$

For given string, we'll try to decode it. If decoding doesn't exist, we say $x \rightarrow M_0$, i.e., assign this encoding to M_0.

→ 

It's possible to give input to M, it's code α_M itself.

Data and code are not distinguishable.

→ $L(M) = \{x \mid M(x) = 1\}$ * $L(M)$ = all strings s.t. M says Yes on it

M solves L iff $L(M) = L$

→ Language ≡ Decision Problem (whether to say Yes or No)

→ **Diagonal Language** :

→ $L = \{x \mid M_x(x) = 0\}$, i.e., all strings s.t. machine represented by it doesn't accept it.

i.e., machines which don't accept it's own code

↓

Q. Does ~~there exist~~ $\exists$ M s.t. $L(M) = L$? No

- It is possible M doesn't halts for its code α_M

Hence, * $M(x) = 1$ → M halts on x & o/p 1

* $M(x) = 0$ → M halts on x & o/p 0 OR M doesn't halts

- Say there exists such $M \rightarrow x_M \Rightarrow L(M) = L$

I) $M(x_M) = 1$, then $x_M \in L$, $\Rightarrow M(x_M) = 0$ $\Rightarrow\Leftarrow$

II) $M(x_M) = 0$, then $x_M \notin L \Rightarrow M(x_M) = 1$ $\Rightarrow\Leftarrow$

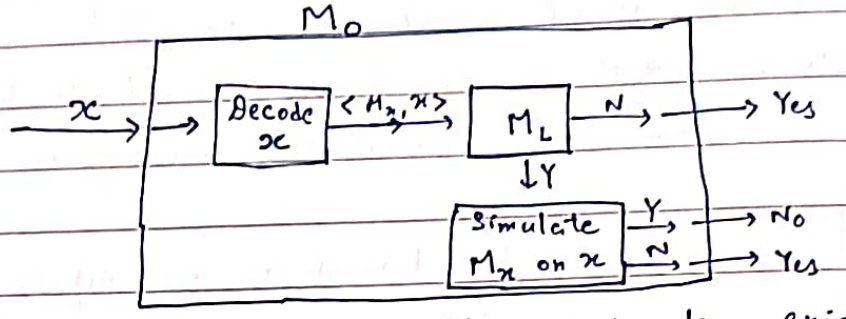
Hence, there doesn't exist such M s.t. $L(M) = L = \{x \mid M_x(x) = 0\}$

→ Thus, it is a shortcoming of TM. Certain problems like this are uncomputable.

→ **Halting Problem** : $L = \{ \langle M, x \rangle \mid M \text{ halts on } x \} = H$

Say, M_L be turing machine s.t. $L(M_L) = L$

Say M_0 be turing machine for $L = \{ x \mid M_x(x) = 0 \}$



∴ If M_L exists, then M_0 must also exists

But we know M_0 doesn't exist.

Hence, no M_L exists s.t. $L(M_L) = L = \{ \langle M, x \rangle \mid M \text{ halts on } x \}$

Q. Which problem is harder ? Diagonal language or Halting Problem

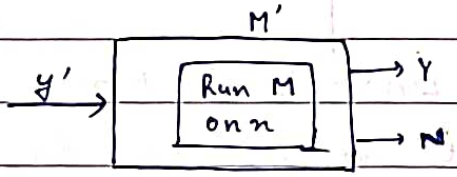
If we can solve halting problem, we can solve diagonal ~~proble~~ language. Hence, Halting problem is harder.

→ $L = \{ M \mid M \text{ halts on all i/p} \}$

• deems harder than halting problem

Say M_L be turing machine s.t. $L(M_L) = L$

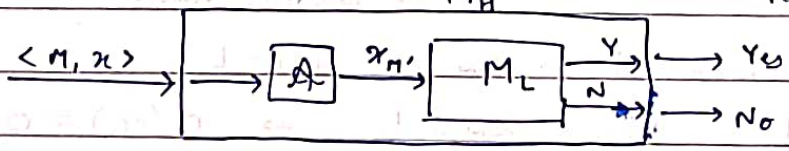
and M_H be —"— for halting problem H.



$\langle M, x \rangle \xrightarrow{A} M'$

$\langle M, x \rangle \in H \iff M' \in L$

M' is input-independent. If M halts on x, M' halts on all input. If M doesn't halt on x, M' halts on no input.



∴ If M_L exists, M_H exists ⇒⇐

Hence, no such M_L exists.

Thus, L is uncomputable.

$\star$ $D \leq_p H \leq_p L$

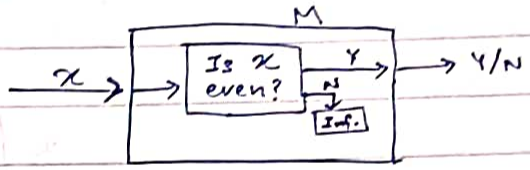
We say diagonal language is reducible to halting problem in polynomial time.

→ $L = \{x \mid x \text{ is even}\} \leq HALT$

Here, we can solve L.

$x \in L \iff \langle M, y \rangle \in HALT$

Let $y = x$



If x is even, M halts

If x is odd, M goes in infinite loop.

Thus, we have designed M in such a way that it halts iff x is even

Hence, we have reduced L to HALT.

→ HALT $\leq$ L ? No

We know L is solvable. If such reduction exists, we can solve the halting problem, but we know HALT is un-solvable. Hence, this reduction doesn't exists.

→ $\{0,1\}^* \leq HALT$

$x \longmapsto \langle M, y \rangle$

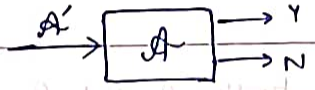
M : a TM which always halts

→ $\emptyset \leq HALT$

$x \longmapsto \langle M', y' \rangle$

M' : a TM which never halts

→ Check whether a TM correctly solves odd-even problem ?



① Convert to decision problem

↳ $L = \{M \mid M \text{ halts and solves odd-even problem}\}$

↳ $L' = \{M \mid \text{If } x \text{ is even, } M(x) = 1 \text{ otherwise } 0 \text{ or may not halt}\}$ "relaxed version"

② Halt $\leq$ L ✓ or L $\leq$ Halt ✗

⇓

Since Halt is not-solvable, then L is also not-solvable

we know this

★ $A \leq B$

$x \mapsto y_x$

$x \in A \iff y_x \in B$

★ If B is solvable, A can be solved.

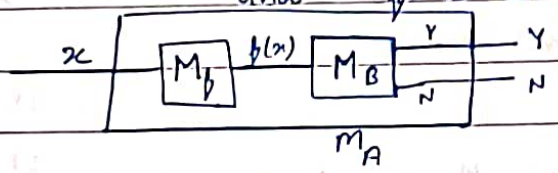
★ If B is not solvable, A can still be solvable.

$x \xrightarrow{f} y_x \xrightarrow{A_B} 0/1$

→ Reduction function (f) : $\{0,1\}^* \to \{0,1\}^*$

s.t. $x \in A \iff f(x) \in B$

and f is computable



Claim : $L(M_A) = A$

$x \in A \iff f(x) \in B \iff M_B(f(x)) = 1$

$\iff M_A(x) = 1$

$\therefore \quad x \in A \iff M_A(x) = 1$

$\therefore \quad A = L(M_A)$

→ $A \leq B$

↳ $A_B \Rightarrow A_A$

↳ $M_B \Rightarrow M_A$

→ Given X and Y. We know X is not computable. We want to prove Y is not computable.

So, we should reduce X to Y, i.e., $X \leq Y$

→ If $A \leq B$ and $B \leq A$, both A and B are of equal hardness

→ $I = \{\langle G, k\rangle \mid$ size of largest ind. set in G is k$\}$ — optimization problem (Harder) V/S

$I' = \{\langle G, k\rangle \mid \exists$ an ind. set of size $\geq k\}$ — decision problem (Easier)

I' is easier than I

↳ In I', we can just give a set S, $|S| \geq k$ to solve it,

But for I, we also want to justify that S is largest set.

→ $C = \{\langle G, k\rangle \mid \exists$ a clique of size atleast $k\}$

★ $C \leq I'$ or $I' \leq C$? Both

$\langle G, k\rangle \in C \iff \langle \bar{G}, k\rangle \in I'$

Because in G there are edges b/w every two vertices of k.

⇒ in $\bar{G}$, no edge b/w them. Hence, it forms an independent set.

∴ $C \leq I'$ as well as $I' \leq C$.

→ $SAT = \{\varphi \mid \varphi$ is satisfiable$\}$; $\varphi = \bigwedge_{i=1}^{k} C_i$ (cnf)

0-1 Integer Programming.

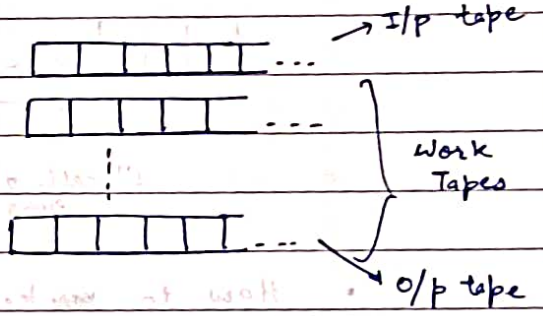
$\varphi \in SAT \iff P \in$ 0-1 int.

For a clause $x_1 \vee \bar{x}_2 \vee x_3$, we can have

$x_1 + (1 - x_2) + x_3 \geq 1$ in integer programming.

SAT ≤ 0-1 integer prog.

★ Turing Machine (Cont.)



- One-way infinite tape
- First tape : Input, read-only
- Remaining tapes : Work tapes (atleast two)
- Last work tape : output tape
- Output of TM = contents of output tape
- Each tape has it's own head
-

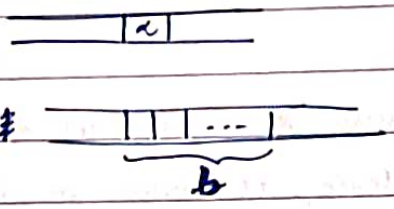
→ Any model of TM can be simulated on basic version of TM with polynomial cost. This is called Robustness of TM.

★ $M_1 : \Gamma_1 = \{\ldots, \alpha, \_\_\}$ UTF-8

$M_2 : \Gamma_2 = \{0, 1, B\}$

→ How much gain in using M_1 over M_2 ?

$\alpha \rightarrow 01\ldots101$, $|\alpha| = \lceil \log_2 |\Gamma_1| \rceil = b$

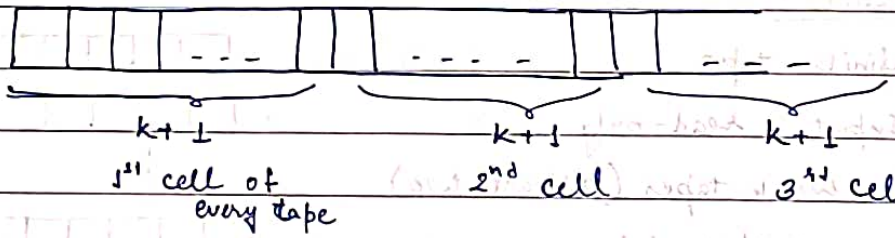


One transition in M_1 can be simulated in 3 b steps on M_2

1) Read α → read b symbols on M_2
2) write γ → write b symbols backwards
3) Move L/R → we are at start location again. So, to move L/R, we have to move only b steps.

- Hence, if M_1 takes $O(T)$ time, M_2 will take $O(3bT)$ time which is basically $O(T)$ asymptotically as b is constant for the problem, i.e., input independent.

★ k+1 -one way infinite tape TM V/s 1 one way infinite tape TM



- How to keep track where the k+1 read-write heads are?
  Increase symbols in alphabet: $\{0, 1, B, \hat{0}, \hat{1}, \hat{B}\}$
  At the location of head ptr, we'll use $\hat{0}/\hat{1}$ instead of 0/1.

- Now for transition step, we'll go over entire tape till last character and collect all special symbols (i.e., one having head-ptrs) and based upon it make the transition and then come back making the changes.
- If head moves L/R, we have to change 0/1 of new cell to $\hat{0}/\hat{1}$.

- States = $(q, \underbrace{-, -, -, -, \ldots, -}_{k+1})$

  $(q, 1, -, -, \ldots, -)$, $(q, 0, -, -, \ldots, -)$

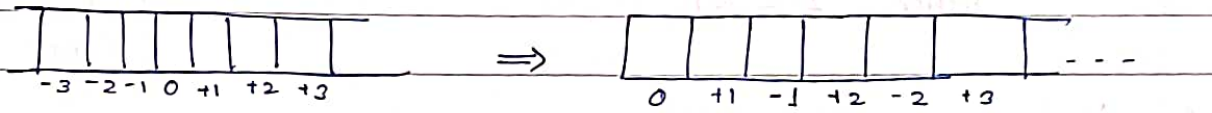
  When we read a symbol in a current transition, we can change to q_0 to q_1 and thus remember input 0/1 (to say).

  No. of states = $O(3^k) = O(1)$ constant. (k is fixed)

★ No. of states in TM should be finite.

- Hence, k+1 - one way infinite tapes TM can be simulated on 1 one-way infinite tape TM with only polynomial cost.

★ Two way infinite TM v/s one way infinite TM



$Q = \{q, \ldots\}$ ⇒ $Q = \{(q, +), (q, -), \ldots\}$

(+) : move same way ; (-) : move opp. way

→ Thus, we can define TM in various manner regardless of it's computability, i.e., they all have same computation power.

★ Complexity Classes

→ Set of languages or decision problems.

→ DTIME (n^2) (deterministic) – all languages L s.t. ∃ a TM M, L(M) = L in at most $c.n^2$ steps, where c is a constant.

- c is constant for particular language.
- n^2 is a function. Can be any function $f(n)$

→ $P = \bigcup_{c \geq 0} DTIME(n^c)$ – all languages that can be solved in polynomial time deterministically.

→ **NP** – a language $L \in NP$ if $\exists$ a polynomial $p(x)$ and

(I) a polynomial time DTM M s.t.

$x \in L \Leftrightarrow \exists y, |y| \leq c.p(|x|)$ and

$M(x, y) = 1$

M → verifier

y → certificate or witness

- eg: $I = \{\langle G, k\rangle \mid G$ has an independent set of size at least $k\}$

$I \in NP$?

$x = \langle G, k\rangle$ and y = set of nodes of size $\geq k$

M is a TM that checks if y is independent set of x or not.

M will take atmost $O(n^2)$ time.

$|y| \leq O(n)$

Hence, $I \in NP$

Hard to solve but easy to verify $\equiv$ NP

- Similarly, SAT $\in$ NP

- eg: $I' = \{\langle G, k\rangle \mid$ largest ind-set in G is of size $k\}$

$I' \notin NP$

Hard to solve as well as hard to verify ($\neq$ NP)

We have to check that size k set is largest ⇒ Have to check all possible sets of size $> k$.

↳ exponential

★ **Non-Deterministic TM**

→ For a particular transition, there are multiple possible moves to make.

$$(q, 0) \mapsto \{(q', 1, L), (q'', 0, R), \text{-----}\}$$

→ TM makes only one move at a time.

↳ no notion of making all possible transition.

→ It makes move arbitrarily (not randomly even)

↳ have certain probability distribution associated.

or we can say non-deterministically.

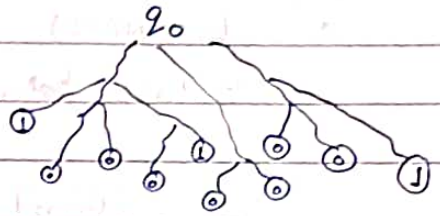
i.e., there is no probability associated with selection.

→ we have randomized TM that makes random moves and have certain probability associated for every possible transition.

→ How to decide if a NDTM solves L ?

$L(N) = L$

Construct a tree for every possible transition for $x \in L$ on N.



- $x \in L(N)$ iff some path in tree leads to 1 (atleast one path)
  $x \notin L(N)$ iff all paths in tree leads to 0.
- $N(x) = 1 \Rightarrow x \in L(N)$
  But $N(x) = 0 \Rightarrow$ no conclusion can be made

→ Eg : $SAT = \{ \varphi \mid \varphi \text{ is satisfiable} \}$

$\varphi(x_1, x_2, \dots, x_n)$

Execute $(q, B) \mapsto \{(q, 0, R), (q, 1, R)\}$ for n cells.

This will give an arbitrary string of length n.

Then evaluate φ on this assignment of variables and output correspondingly.

Thus, NDTM solves SAT in $O(n)$ time.

If $\varphi \notin SAT$, then for every possible choice of moves, NDTM will say No.

If $\varphi \in SAT$, there exists a possible choice of moves that gives a satisfiable assignment of φ and NDTM will say Yes.

→ Similarly, for independent set problem, we can non-deterministically generate a subset and check if it's independent.

★ NDTM → generate a solution (guess) → verify in polynomial time

→ $NTIME(f(n))$ – all languages L s.t. ∃ a NDTM N, $L(N) = L$ in atmost $c \cdot f(n)$ steps, where c is a constant.

(non-deterministic)

→ $NP = \bigcup_{c \geq 0} NTIME(c^n)$ – all languages that can be solved in polynomial time non-deterministically.

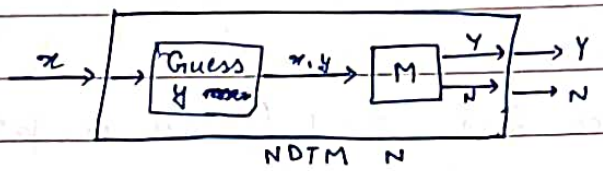
(II)

★ Does NP(I) = NP(II) ? Yes

- $L \in NP(I) \Rightarrow L \in NP(II)$

We have polynomial p and DTM M

$\therefore |y| \leq c.p(|x|)$



$\therefore$ Guessing y takes polynomial time and $M(x,y)$ also takes polynomial time

Thus, N takes poly. time.

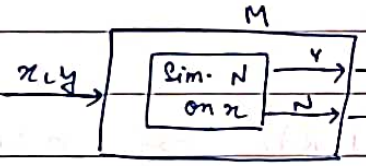
Runtime of $N = p(|x|) + (Q(x + p(|x|)))^2$

↳ $Q(x)$ is runtime of M

$x \in L_1 \Rightarrow \exists y \Rightarrow$ NDTM N will guess y on some path correctly $\Rightarrow x \in L_2$

$x \notin L_1 \Rightarrow$ No y exist $\Rightarrow$ for all path NDTM will say No $\Rightarrow x \notin L_2$

- $L \in NP(II) \Rightarrow L \in NP(I)$



$L \in NP(I) \Rightarrow L \in NP(II) \Rightarrow x \notin L_2$

$\Downarrow$

$NP(I) \subseteq NP(II)$ ....(i)

Simulate N s.t. it takes transition specified by y.

(eg.: $y \rightarrow 1\ 3\ 2\ 5\ 7 \cdots$, it takes 1st transition on 1st move, then 3rd —"— 2nd —"— and so on)

$x \in L_2 \Rightarrow$ For some path N accepts $x \Rightarrow \exists y$ s.t. $M(x,y) = 1 \Rightarrow x \in L_1$

$x \notin L_2 \Rightarrow$ For all path N rejects $x \Rightarrow$ No y exists s.t. $M(x,y) = 1 \Rightarrow x \notin L_1$

$\Rightarrow L \in NP(II) \Rightarrow L \in NP(I) \Rightarrow NP(II) \subseteq NP(I)$ ....(ii)

From (i) & (ii), $NP(I) = NP(II)$.

→ $\underline{EXP} = \bigcup_{c \geq 0} DTIME(2^{n^c})$ — all languages that can be solved in exponential time deterministically.

→ $\underline{NEXP} = \bigcup_{c \geq 0} NTIME(2^{n^c})$ — all languages that can be solved in exponential time non-deterministically.

★ $P \subseteq NP \subseteq EXP \subseteq NEXP$

→ $P \subseteq NP$ (II) is trivial, since DTM is a special case of NDTM.

→ $P \subseteq NP$ (I)

$L \in P \Rightarrow \exists\, M_0,\ L(M_0) = L$

What will be $p(x)$, M and y?

We can solve language L, so take $y = \epsilon$

$p(x) = 0$ and $M = M_0$

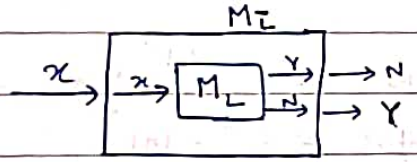
$\therefore\ x \in L \Leftrightarrow M_0(x, \epsilon) = 1$ and $x \notin L \Leftrightarrow M_0(x, \epsilon) = 0$

Thus, $P \subseteq NP$ (I) $\Rightarrow$ $P \subseteq NP$

→ (What if $x \in L$ but $\nexists y$? $L \notin NP$ in this case)

→ Similarily, $EXP \subseteq NEXP$ is also trivial.

→ $L \in P \Leftrightarrow \bar{L} \in P$

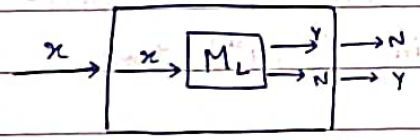


$x \in L \Leftrightarrow M_L(x) = Y$

$\Leftrightarrow M_L(x) = N$

$\Leftrightarrow x \notin \bar{L}$

→ $L \in NP \not\Leftrightarrow \bar{L} \in NP$



$M_{\bar{L}}$

$x \in L \Rightarrow M_L(x) = N$ sometimes

$\Rightarrow M_{\bar{L}}(x) = Y$ sometimes

$\Rightarrow x \in \bar{L}$ which is not possible

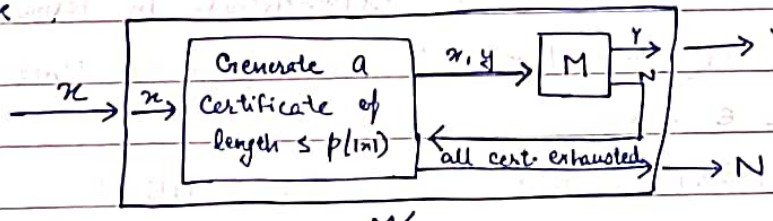
$\Rightarrow\Leftarrow$ $L \in NP \not\Leftrightarrow \bar{L} \in NP$

• Thus, we can complement o/p of DTM but not of NDTM.

→ $NP \subseteq EXP$

$L \in NP \Rightarrow \exists\, y$, $|y| \leq p(|x|)$ s.t. $M(x, y) = 1$

We can generate all possible y's and feed it to $M(x, y)$ to check.



Running time of $M' = \underline{2^{p(x)} \cdot q(x)} \Rightarrow$ Exponential time DTM $\Rightarrow L \in EXP$.

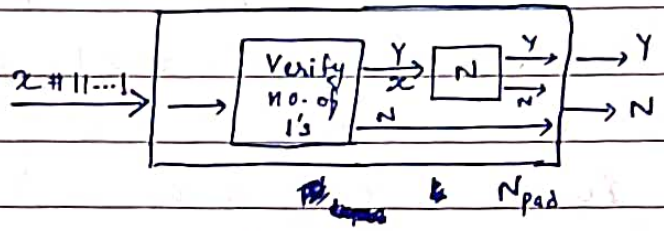
★ ● $P = NP \Rightarrow EXP = NEXP$

- We know $EXP \subseteq NEXP$ and just need to show $NEXP \subseteq EXP$
- Say $L \in NEXP \Rightarrow \exists$ a NDTM N s.t. $L(N) = L$ in $O(2^{n^c})$ time

We define $L_{pad} = \{ x \# 1^{2^{|x|^c}} \mid x \in L \}$

We count no. of 1's in input after # and verify it.

Then drop 1's and verify x using N.



$L(N_{pad}) = L_{pad}$

N_{pad}

To verify no. of 1's → get length of x → compute $2^{|x|^c}$ in time
→ verify if there are $2^{|x|^c}$ 1's after x

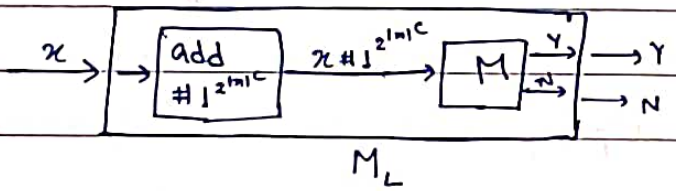
Thus, this step takes polynomial time in length of input.

Now, to check x using N takes $2^{|x|^c}$ time which is again polynomial in length of input $(= 2^{|x|^c} + |x|)$

Thus, running time of N_{pad} is polynomial

$\therefore L_{pad} \in NP$

$\Rightarrow L_{pad} \in P \Rightarrow \exists$ a DTM M s.t. $L(M) = L_{pad}$ in $O(n^d)$ time



M_L

M says No if no. of 1's is incorrect (which can't happen)

or $x \notin L$

$\Rightarrow L(M_L) = L$

Running time of $M_L = O\left(2^{|x|^c} + (|x| + 2^{|x|^c})^d\right)$

= exponential in terms of input length.

$\exists$ a DTM M_L s.t. $L(M_L) = L$ in exponential time

$\Rightarrow L \in EXP$

Hence, $NEXP \subseteq EXP \Rightarrow EXP = NEXP$

★ $EXP \neq NEXP \Rightarrow P \neq NP$

→ $\overline{NP} = \{ L \mid L \notin NP \}$

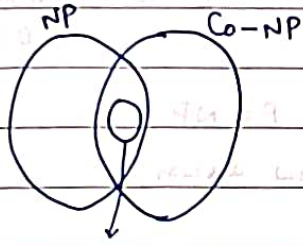
→ $co\text{-}NP = \{ \bar{L} \mid L \in NP \}$

★ $\overline{NP} \neq co\text{-}NP$

Eg: HALT $\notin$ NP

HALT $\notin$ Co-NP

→ Claim : If N solves L in $O(f(n))$ time, then $\exists$ a DTM M solving L in $O(2^{f(n)})$ time.



★ $P = co\text{-}P \subseteq NP \cap co\text{-}NP$

Q. Is $P \subset NP \cap Co\text{-}NP$?

Q. Or $P = NP \cap Co\text{-}NP$?

Q. Does $\exists L$ $L \in NP$, $L \notin Co\text{-}NP$?

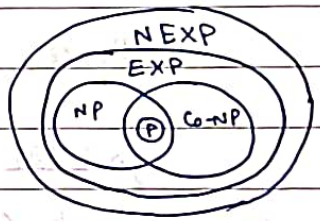
Q. Does $\exists L$ $L \in NP \cap Co\text{-}NP$ but $L \notin P$?

→ NP-hard - $L \in$ NP-hard iff $\forall L' \in NP$ $L' \leq_p L$

(Not a complexity class)

→ NP-Complete - $L \in$ NP-complete iff i) $L \in$ NP-hard

ii) $L \in NP$



★ Properties

1) Reduction is transitive.

$L_1 \leq_p L_2$ and $L_2 \leq_p L_3 \Rightarrow L_1 \leq_p L_3$

Proof: $L_1 \leq_p L_2$ $\exists f$ s.t. $x \in L_1 \Leftrightarrow f(x) \in L_2$

$L_2 \leq_p L_3$ $\exists g$ s.t. $x \in L_2 \Leftrightarrow g(x) \in L_3$

$x \in L_1 \Leftrightarrow f(x) \in L_2 \Leftrightarrow g(f(x)) \in L_3$

$\therefore L_1 \leq_p L_3$ $\underbrace{n^d \quad n^c}_{O(n^{cd})}$

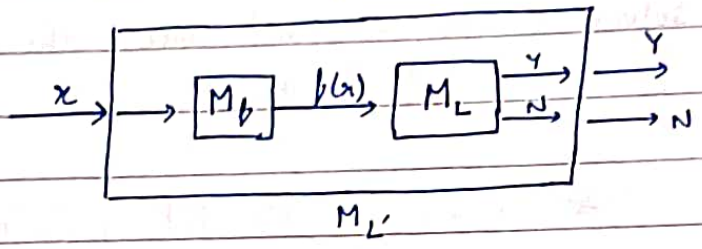
$\left(|x| \xrightarrow{f} |x|^c \xrightarrow{g} (|x|^c)^d \right)$

2.) If a NP-hard language is in P, then P = NP

Let $L \in$ NP-hard, $L \in P$

$L' \in NP$, then by definition $L' \leq_P L$

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



M_f & M_L are polynomial & deterministic

$\therefore M_{L'}$ is polynomial time DTM

Hence, $L' \in P \Rightarrow NP \subseteq P \Rightarrow P = NP$

$P \subseteq NP$ (we knew already)

3.) Let L be NP-complete, then $L \in P \iff P = NP$

$L \in$ NPComp. $\Rightarrow L \in$ NP hard and $L \in P \Rightarrow P = NP$ (we just saw)

$L \in$ NP-Comp $\Rightarrow L \in NP$ and $P = NP \Rightarrow L \in P$

Hence, $L \in$ NP-comp $\Rightarrow L \in P \iff P = NP$

★ $L = \{ \langle \alpha, x, l, t \rangle \mid \exists y, |y| = l \text{ s.t. } M_\alpha(x, y) = \perp \text{ in time } t \}$ (TMSAT)

$L \in$ NP-complete

- Say $L' \in NP$

  $\exists$ poly p and $\exists$ a DTM M s.t.

  $x \in L' \iff \exists y, |y| \leq p(|x|)$ and $M(x, y) = 1$ (M runs in $q(\cdot)$)

  $x \mapsto y$

  $x \in L' \iff y \in L$

  $x \mapsto \langle \alpha_M, x, p(|x|), q(|x| + p(|x|)) \rangle$ } This can be done in polynomial time

- $x \in L' \Rightarrow \exists y, |y| \le p(|x|)$ s.t. $M(x,y) = 1$

  $\therefore$ By def^n of L, for $f(x) = \langle \alpha_M, x, p(|x|), q(|x| + p(|x|)) \rangle$

  also $\exists y$ s.t. $|y| = l$ and $M_{\alpha_M}(x,y) = M(x,y) = 1$

  in $t = q(\cdot)$ time.

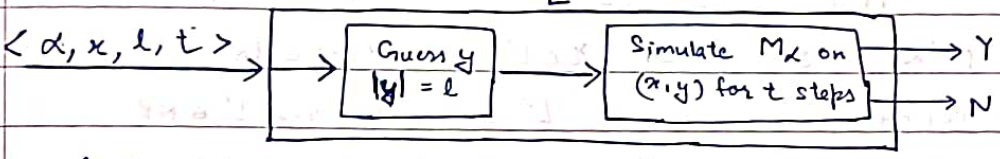
  $\therefore f(x) \in L$

- $x \notin L' \Rightarrow \nexists y \Rightarrow M(x,y) \neq 1$

  Thus, $f(x) \notin L$

  Hence, $L \in$ NP-hard

- Does $L \in NP$? If Yes, then $L \in$ NP-complete



$\text{Time}(N_L) = O(l + t^2)$ ↓ polynomial if input is given in unary

If Simulation halts and say Yes $\Rightarrow$ Accept the tuple

—— ,, —— ,, —— No $\Rightarrow$ Reject → guess y is wrong or → tuple $\notin L$

If simulates doesn't halts in t steps

↳ Stop and reject

$\therefore L \in NP$. Hence, $L \in$ NP-complete

- What if α is NTM ? We can still simulate. Simulation can make mistakes.

★ L_{HALT} is NP-hard (but $L_{HALT} \notin$ NP-comp)

$L' \in NP \Rightarrow \exists p(n)$ & verifier V s.t. $x \in L' \Rightarrow \exists y, |y| \le p(n)$ and $V(x,y) = 1$

(V: $q(n)$ time)

$x \xmapsto{f} \langle M', x' \rangle$ s.t. $x \in L' \Leftrightarrow f(x) \in L_{HALT}$

$x' = \alpha_V \# x \# 1^{p(|x|)}$

M' = Generate all possible certificate y of length $p(|x|)$ & simulate α_V on (x,y)

If some y, $\alpha_V(x,y) = 1 \rightarrow$ Halt

If $\forall y$, $\alpha_V(x,y) = 0 \rightarrow$ Go in ∞ loop.

$x \in L \Rightarrow$ for some y, $\alpha_V(x,y) = 1 \Rightarrow M'$ halts on x'

$x \notin L \Rightarrow$ for no y, $\alpha_V(x,y) = 1 \Rightarrow M'$ doesn't halts on x'

$x \in L \iff \langle M', x' \rangle \in L_{HALT}$

And $f(x)$ computation takes polynomial time.

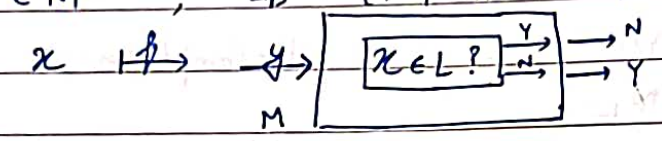
Hence, $L' \leq_p L_{HALT}$ for any $L' \in NP$

Hence, $L_{HALT} \in$ NP-hard

★ (L_{HALT} doesn't have any time bound on M', hence M' can take exponential time but the reduction should be in polynomial time)

★ L_D is NP-hard

$L' \in NP$, $L_D = \{x \mid M_x(x) = 0\}$



(kind of simulating x on Verifier by generating all certificates)

$x \in L' \Rightarrow M(x_M) = 0 \Rightarrow x \in L' \iff f(x) \in L_D$

$x \notin L' \Rightarrow M(x_M) = 1$

$L' \leq_p L_D$ for any $L' \in NP$

Hence, L_D is NP-hard

★ ~~P = comp~~ = P ?

★ P, P-hard and P-complete

P-complete = $P - \{\emptyset, \Sigma^*\}$

$L, L' \in P - \{\emptyset, \Sigma^*\}$

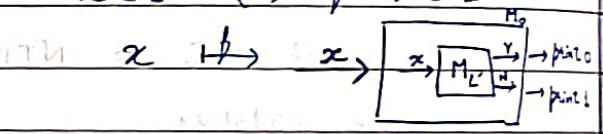
Say $0 \in L$ & $1 \notin L$

$L' \leq L$

$L \in$ P-complete

$x \xmapsto{f} f(x)$

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



$x \in L' \Rightarrow f(x) = 0 \in L$

$x \notin L' \Rightarrow f(x) = 1 \notin L$

→ Notion of hard/comp. is thus not well defined in class P and has different meaning.

★ SAT is NP-complete

We already know SAT $\in$ NP

If we proove SAT $\in$ NP-hard, we are done.

* To proove: TMSAT $\leq_p$ SAT (∵ TMSAT is NP-hard, it will show SAT is NP-hard)

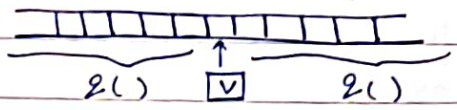
$\forall L' \in NP$, $L' \leq$ TMSAT

If TMSAT $\leq_p$ SAT $\Rightarrow L' \leq$ SAT, hence SAT $\in$ NP-hard

- TMSAT $\in$ NP , $\exists$ $p(n)$ and verifier V in time $q(n)$ s.t.
  $\forall x \in TMSAT \iff \exists y, |y| = p(n)$ and $V(x,y) = 1$

  $TMSAT \leq_p SAT$

  $x \longmapsto \varphi_x$ (Assume V is single tape DTM)



  Max. cells V can affect = No. of steps taken by V
  $= q(|x| + p(|x|))$

- Define configuration of TM as follows (in the i^{th} step) :

  $C_i \longrightarrow \underbrace{\overset{\text{Tape content}}{u}}_{\text{len} = 2q()-1} \# \overset{\text{current T state}}{Q} \# \overset{\text{head pos}^n}{n}$ (All configurations would be of same length)

As we know the TM, we can find $C_{i+1} \longrightarrow \underbrace{v}_{2q()-1} \# Q' \# n/n-1/n+1$

- We can verify given two configuration C_i and C' whether C' follows C or not.
  - Get the head alphabet & current state
  - Look up transition table
  - Verify new tape content, new state & L/R/S.
- Define $\varphi_n(C, C') = 0/1$ to check if C' follows C.
- Now, represent everything in boolean.

  Tape content → already in binary → $u_1\ u_2\ u_3\ \dots\ u_{2q-1}$

  States → $\log|Q|$ variables → $v_1\ v_2\ \dots\ v_{\log|Q|}$

  n → Atmax $2q-1$ → $\log(2q-1)$ variables → $w_1\ w_2\ \dots\ w_{\log(2q-1)}$

- For transition $(2, 1) \to (2', 0, R)$

  $\{$ If $(n = 1$, and $\delta = 2$ and $u_1 = T)$ then $[(n' = 2$ and $\delta' = 2'$ and $u_1' = F)$ and $(u_2 = u_2'$ and $u_3 = u_3'$ and $\dots)]\}$

  $\wedge \{(1, 2, F) \Rightarrow [\quad]\}$
  $\wedge \{(1, 2', T) \Rightarrow [\quad]\}$
  $\wedge \dots\dots$
  $\wedge \{(2, 2, T) \Rightarrow [\quad]\}$
  $\wedge \dots\dots$

  (premise
  • Out of all these, only one $\wedge$ would be true & rest all premise should be false, that will automatically make the clause true.)

- We can have this for every transition, joined with 'and' ($\wedge$).

- similarly, define $\varphi_e(C, C') = 0/1$ to check if $C = C'$

  $\langle \alpha, x, l, t \rangle \mapsto \varphi_x$

- Now, we know the initial configuration of V (except for certificate input)

  $C_0(y_1, y_2, \ldots, y_\ell)$ → Actual configurations of verifier (given in binary)

- Let V_i be the configuration (represented as boolean variables that can be assigned T or F).

- $\varphi_x = \varphi_e(C_0, V_0) \wedge \varphi_n(V_0, V_1) \wedge \varphi_n(V_1, V_2) \wedge \ldots \wedge \varphi_n(V_{t-1}, V_t) \wedge \varphi(V_t^2 = q_f^1)$

  ↓ to check if final cont state is final state

→ If $\langle \alpha, x, l, t \rangle \in$ TMSAT, then we can assign values of $V_0, V_1, \ldots V_t$ by looking at transition table of verifier on input (x, y).

→ If $\langle \alpha, x, l, t \rangle \notin$ TMSAT, then either $V_t^2 \neq q_f$ and even if we set V_t^2 s.t. $V_t^2 = f$, then $\varphi_n(V_{t-1}, V_t) = 0$.

If we set V_{t-1} s.t. $\varphi_n(V_{t-1}, V_t) = 1$ then $\varphi_n(V_{t-2}, V_{t-1}) = 0$ and so on.

If we set all $V_0, \ldots V_t$, then $\varphi_e(C_0, V_0) = 0$

and if $\varphi_e(C_0, V_0) = 1$, then $\exists$ a path of transitions in verifier for input (x, y) that ends in final state after t steps hence $\langle \alpha, x, l, t \rangle \in$ TMSAT

$\Rightarrow\Leftarrow$

- Thus, $\langle \alpha, x, l, t \rangle \in$ TMSAT $\iff \varphi_x \in$ SAT

→ length of formula φ = polynomial in terms of $\langle \alpha, x, l, t \rangle$

No. of clauses per transition $= O(q) \cdot O(\log |Q|) \cdot O(2) = O(q(|x|))$ (constant)

Length of clause $= O(q(|x|)) \times O(q(|x|))$

No. of formula $= O(t)$

Thus, given $\langle \alpha, x, l, t \rangle$ we can generate φ_x in polynomial time.

↳ evaluation of φ_x is not the job of reduction.

→ What about certificate ?

We can have variable for that as well in C_0.

Thus, $x \mapsto \varphi_x(\underbrace{y_1, y_2, \ldots, y_\ell}_{\text{independent}}, \underbrace{V_0, V_1, \ldots V_t}_{\text{dependent on } C_0(y)})$

$x \mapsto \varphi_x(y_1, y_2, \ldots y_\ell)$ in polynomial time.

Hence, if $\psi_x(y)$ is satisfiable then the values of y act as certificate for $V(x,y) = 1$

if $\exists y$ s.t. $V(x,y) = 1$ then we can use this y to make ψ_x satisfiable.

Thus, $\langle x, n, l, t \rangle \in$ TMSAT $\Leftrightarrow \psi_x(y_1, y_2 \cdots y_e) \in$ SAT

$\Rightarrow$ TMSAT $\leq_p$ SAT $\Rightarrow$ SAT is NP-hard $\Rightarrow$ SAT is NP-complete

★ SAT $\leq_p$ 3-SAT, i.e., 3-SAT is NP-complete

★ (Refer Book-49)

★ IND-SET is NP-complete

To proove : 3-SAT $\leq_p$ IND-SET

$\psi \longmapsto \langle G, k \rangle$

- $\psi = C_1 \wedge C_2 \wedge C_3 \wedge \cdots \wedge C_n$; (say $C_1 = \bar{x}_1 \vee x_2 \vee x_3$)

Declare 8 vertices for C_1

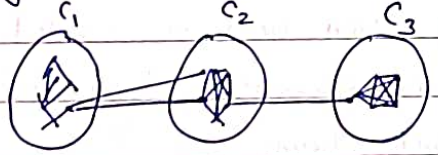
Label them as $(x_1=T, x_2=T, x_3=T)$, $(x_1=T, x_2=T, x_3=F)$, ----

- Say $C_2 = x_1 \vee x_4 \vee \bar{x}_2$

Repeat same for C_2 as well and C_3, C_4 ---

- If we choose, say $(x_1=T, x_4=T, x_2=T)$ as vertex of ind. set, then we'll assign $x_1=T$, $x_4=T$ and $x_2=T$ in formula ψ and vice-versa.

- Drop the vertex in each clause that makes the clause FALSE.
- For every clause, add edge b/w all the vertices s.t. each clause is a 7-clique.
- In b/w two clause-vertex set, add edge b/w them if they have contradictory assignment ($x_i=T$ in one vertex and $x_i=F$ in other).
- There couldn't be two or more vertex in ind-set from one cluster and we want every clause to be TRUE. We take $k = n$ (no. of clauses in ψ)



→ If ψ is SAT, then $\exists$ a assignment (non-contradicting) of variables. We can select those vertices from each cluster. Since each clause is True, $\exists$ a vertex from each cluster, hence total n vertices & it is non-contradicting, thus no-edge b/w them. Hence, it gives an ind-set.

→ If there is ind. set of size n, it must be one vertex per cluster since each cluster is a clique. Thus, there is non-contradicting assignment of variables. Each assignment makes that clause true, thus φ is satisfiable.

Hence, $\varphi \in 3\text{-SAT} \iff \langle G, n \rangle \in \text{IND-SET}$

$3\text{-SAT} \leq_p \text{IND-SET} \rightarrow \text{IND-SET}$ is NP-complete

★ $2\text{-SAT} \in P$ H.W.

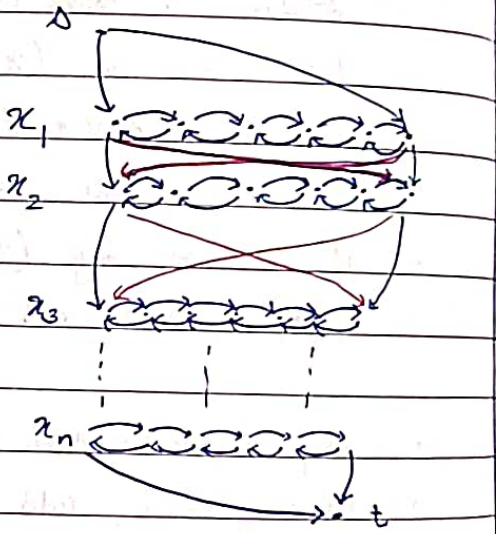
★ $\forall L \in NP$, $L \leq_p SAT \leq_p 3\text{-SAT} \leq \text{IND-SET}$

★ Directed Hamiltonian Path is NP-Complete

↳ A path that covers all vertices of graph exactly once.

To proove: $SAT \leq_p \text{Dir-HAM PATH}$

$\varphi(x_1, x_2, \ldots x_n) \longmapsto G_{\varphi}$



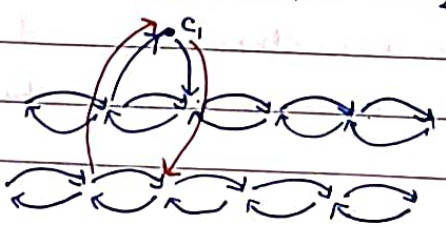
- For each variable x_i, we'll have certain no. of vertices in G_{φ} and having edges in form of chain.
- We'll have two vertices 's' and 't' having only ~~outgoing~~ outgoing & incoming edges respectively.
- If we traverse a chain of x_i from L to R, it is assigned True and if from R to L, it is assigned False and vice-versa
- We'll have edge from 's' to 'x_1' chain at both ends.
  From both ends of x_i to both ends of x_{i+1}.
  From both ends of x_n to 't'.

(1) For ham-path in G_{φ}, every chain must be visited completely.

(2) Every chain must be visited sequentially from top to down.

- We'll have a vertex for each clause in $\varphi(x)$.

Say $C_1 = x_1 \vee x_4 \vee \neg x_2$

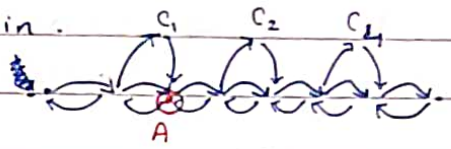


- If we visit x_1 from L to R and it it satisfies some clause visit that clause. (if clause is not visited already)

- We have to go systematically and can't jump to chains via clause vertices.
- Also start & end vertex of path should be 's' and 't' respectively.

③ If we are not systematic, we can't have ham. path in G_φ.
↳ Decide size of chain!
Don't overlap clause vertices & edges in chain.

Size of chain = $2 + 2$ (no. of clauses in which x_i appear)



If we don't come down to same chain after visiting C_4, vertex A would be last vertex of path which is not possible as 't' is last vertex.

- If $\varphi(x)$ is satisfiable, there is an assignment of x_i's.
Visit the G_φ accordingly and visiting clauses accordingly to get ham. path.
- If there is ham-path in G_φ, then there must be assignment of variables and since all clause-vertices are visited, all clauses in φ are satisfiable. Hence, φ is satisfiable.

$$\varphi(x) \in SAT \iff G_\varphi \in HAM\text{-}PATH$$

$$SAT \leq_p HAM\text{-}PATH$$

★ **Decision Problem v/s Search Problem**

- Decision Problem – $\langle G, k \rangle$ → Yes / No answer only to say if such a set exists or not.
- Search Problem – find solution of the problem, i.e., the independent set of size k in G.
- Decision Prob = Search Prob. (in terms of hardness)

→ Let's take SAT. Assume that M_D solves SAT in polynomial time.
We can use M_D to solve search-version of SAT.

- Solve φ using M_D.
- If φ is satisfiable, solves $\varphi \wedge (x_1)$. Say it is not satisfiable.
So, we can say $x_1 = F$ in final assignment.
- Then check $\varphi \wedge (\neg x_1) \wedge (x_2)$. Say it is satisfiable. Then we have $x_1 = F$ and $x_2 = T$ in final assignment.

- We can repeat this for every variable in φ to get full assignment.
- Time Required = (no. of variables in φ) × (Time of M_D) ↓ polynomial
  = polynomial in input length

Hence, we can solve search-version of SAT using M_D.

★ If we reduce $x \longmapsto y$, $x \in L_1 \Leftrightarrow y \in L_2$ and $L_1, L_2 \in NP$
Cert(x) is identical to Cert(y).

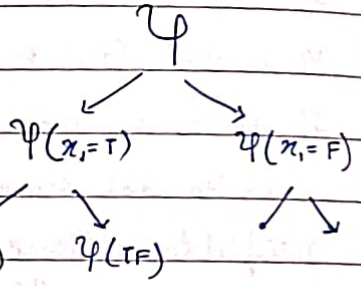
★ If an unary language ($\subseteq 1^*$) is NP-comp., then P = NP.
↳ only one alphabet

For length n, no. of possible strings ≤ 1

Say $L \subseteq 1^*$ & L is NP-comp.

$\therefore SAT \leq_p L$

$\varphi \longmapsto x$



$f \rightarrow$ polynomial time

- We generate labels of each formula in tree using $f(\varphi)$.
- Since $f()$ works in poly. time, for it can generate polynomial length unary strings only and, each length, there is atmax 1 possible string. Hence, there are atmax polynomial distinct strings available for labelling.