→ Converse of this theorem is not true!

if $X \leq_T^P Y$ then $X \not\leq_m^P Y$
(need not be true)

TSP → consists of finding a tour in a graph, that begins and ends at the same node after having visited each node exactly once.

TSPD → a bound L is provided; to decide if a valid tour (TSP) exists whose cost $\leq$ L.

<u>Theorem</u>: HAMD $\leq_m^P$ TSPD.

<u>Proof</u>: Let $G = \langle N, A \rangle$ be a graph with 'n' nodes.

→ <u>To decide</u>: if G is Hamiltonian.

<u>Define</u>: f(G) as the instance of TSPD consisting of a complete graph $H = \langle N, N \times N \rangle$, the cost function,

$$c(u,v) = \begin{cases} 1 & \text{if } \{u,v\} \in A \\ 2 & \text{otherwise} \end{cases}$$

and the bound $L = n$.

Observe: Any Hamiltonian cycle in 'G' translates into a tour in H that has cost exactly 'n'. If there are no Hamiltonian cycles in 'G', any tour in 'H' must use atleast one edge of cost 2 and thus be of total cost atleast $n+1$

∴ 'G' is an yes-instance of HAMD iff, $f(G) = \langle H, C, L \rangle$ is an yes-instance of TSPD.

∴ HAMD $\leq_m^P$ TSPD [as 'f' is computable in polynomial time].

NP-Complete problems

Defn: A decision problem X is NP-complete if
$X \in NP$ and
$Y \leq_T^P X$ for every problem $Y \in NP$.

Q What would happen if some NP-Complete problem 'X' could be solved in polynomial time?

How to prove that a problem is in NP-Complete?

Theorem: Let X be an NP-complete problem. Consider a decision problem $Z \in NP$ such that $X \leq_T^P Z$. Then Z is also NP-complete.

Proof: To be in NP-complete, Z should satisfy 2 conditions:

$Z \in NP$ ✓ (premise)

let $Y \in NP$,

As $X \in$ NP-complete $\Rightarrow Y \leq_T^P X$

$X \leq_T^P Z$ (by premise)

By transitivity,

$Y \leq_T^P Z$

$\Rightarrow Z$ is NP-complete.

Steps: To prove that Z is NP-complete:

(i) Choose an appropriate problem from the pool of NP and show that it is polynomially reducible to Z

(ii) Any solution to Z should be polynomially verifiable (efficient proof system)

Paradox: When the NP-pool is empty, how do we prove that the very first problem is NP-complete?

→ Steven Cook and Leonid Levin (in 1970)
  - proved that NP-complete problems exist.

Defn: A Boolean formula is satisfiable if $\exists$ a way of assigning values to its variables so as to make it true. We denote by SAT, the problem of deciding, given a boolean formula, whether or not it is satisfiable

$(p \vee q) \Rightarrow (p \wedge q)$ → satisfiable when $p = T$ and $q = T$

$(\neg p) \wedge (p \vee q) \wedge (\neg q)$ → not satisfiable

→ intractable to decide when the number of boolean variables 'n' involved is large as there are 2^n possible assignments

→ But any assignment of values to the variables, which satisfies the formula is easy to verify ⇒ SAT $\in$ NP.

Defn: A literal is either a Boolean variable or its negation. A clause is a literal or a disjunction of literals. A Boolean formula is in conjunctive normal form (CNF), if it is a clause or a conjunction of clauses. It is in k-CNF for some $k > 0$, if it is composed of clauses, each of which contains atmost k-literals.

$(p + \bar{q} + r)(\bar{p} + q + r) q \bar{r}$

└ 3CNF (not in 2-CNF)

$(p + qr)(\bar{p} + q(q + r))$ [not in CNF]

$(p \Rightarrow q) \Leftrightarrow (\bar{p} + q)$ [not in CNF]

Defn: SAT-CNF is the restriction of the problem SAT to boolean formulas in CNF. For any $k > 0$, SAT-K-CNF is the restriction of SAT-CNF to boolean formulae in K-CNF.

Theorem: (Cook) SAT-CNF is NP-Complete

- Can be proved (but very convoluted) {Read the intuitive reasoning given in your textbook instead}.

Theorem: SAT is NP-complete.

Proof: SAT $\in$ NP

To show: SAT-CNF $\leq_T^P$ SAT

This is trivial. Since, Boolean formulae in CNF are a special case of general boolean formulae. It is easy to find out if a given boolean formula is in CNF.

$\therefore$ Any algorithm capable of solving SAT can be used directly to solve SAT-CNF.

Hence Proved

Theorem: SAT-3-CNF is NP-Complete

Proof:

WKT SAT-3-CNF is in NP.

We can either prove SAT-CNF $\leq_T^P$ SAT-3-CNF

(or)

SAT $\leq_T^P$ SAT-3-CNF.

$\therefore$ We prove: SAT-CNF $\leq_m^P$ SAT-3-CNF.

Consider an arbitrary boolean formula φ in CNF.

To construct: Efficiently a boolean formula $\xi = f(\varphi)$ in 3-CNF that is satisfiable iff φ is satisfiable.

Case 1: $(K \leq 3)$

φ contains only one clause (which is a disjunction of k literals)

$\Rightarrow$ already in 3-CNF $\Rightarrow \xi = \varphi$

Case-2: $(K = 4)$

Let l_1, l_2, l_3, l_4 be literals such that

$\varphi = l_1 + l_2 + l_3 + l_4$

Let u be a new boolean variable.

$$\xi = (l_1 + l_2 + u)(\bar{u} + l_3 + l_4)$$

$\rightarrow$ If atleast one of the l_i's true $\Rightarrow \varphi$-true and it is possible to select a truth value such that ξ is also true.

$\rightarrow$ If all the l_i's are false, whatever truth value is assigned to 'u' ξ is false.

$\Rightarrow$ φ is true iff ξ is satisfiable with a suitable truth assignment for 'u'

Case-3: $(k \geq 4)$

let $l_1, l_2, \dots, l_k$ be the literals such that $\varphi = l_1 + l_2 + \dots + l_k$.

let $u_1, u_2, \dots, u_{k-3}$ be new boolean variables.

Take,

$$\xi = (l_1 + l_2 + u_1)(\bar{u}_1 + l_3 + u_2) \dots (\bar{u}_{k-3} + l_{k-1} + l_k)$$

Given, any fixed truth values for the l_i's φ is true iff ξ is satisfiable with a suitable choice of assignments for the u_i's.

If the formula φ consists of several clauses, treat each of them independently - using different 'u' variables for each clause and form the conjunction of all the expressions in 3-CNF thus obtained. For example, if

$$\varphi = (p + \bar{q} + r + s)(\bar{r} + s)(\bar{p} + s + \bar{x} + v + \bar{w})$$

we obtain,

$$\xi = (p + \bar{q} + u_1)(\bar{u}_1 + r + s)(\bar{r} + s)(\bar{p} + s + u_2)$$
$$(\bar{u}_2 + \bar{x} + u_3)(\bar{u}_3 + v + \bar{w})$$

→ As each clause is "translated" with the help of different 'u' variables and as the only way to satisfy φ is to satisfy each of its clauses with the same truth assignment for the boolean variables. ⇒ Any satisfying assignment for φ gives rise to a satisfying assignment for ξ and vice versa.

$\Rightarrow \psi$ is satisfiable iff ξ is.

But ξ is in 3-CNF.

$\therefore$ We have shown how to transform an arbitrary CNF formula into one in 3-CNF efficiently, in a way that preserves satisfiability.

$\Rightarrow$ SAT-CNF $\leq^p_m$ SAT-3-CNF

$\Rightarrow$ SAT-3-CNF is NP-Complete.

<u>Defn</u>: Let 'G' be an undirected graph and let 'k' be an integer. A colouring of 'G' is an assignment of colours to the nodes of 'G' such that any two nodes joined by an edge are of different colours. It is a k-colouring if it uses no more than k-distinct colors. The smallest 'k' such that a k-colouring exists is called the graph's chromatic number. and any such k-colouring is an optimal colouring. We define the following 4 problems.

* 3COL: Can 'G' be painted with 3 colors?
* COLD: Given 'k' can 'G' be painted with k-colours?

NP-complete

* COLO: find the chromatic number of G
* COLC: find an optimal colouring of G.

→ All these problems are polynomially equivalent.

NP-hard problems

→ To provide evidence that a problem can't be solved efficiently, there is no need to prove that it belongs to NP.

→ A problem X is NP-hard, if there is an NP-complete problem Y, that can be polynomially Turing reduced to it: $Y \leq_T^P X$

→ Any polynomial time algorithm for X would translate into one for Y. Since Y is NP-complete ⇒ P = NP. [Contrary to the generally accepted belief].

∴ No NP-hard problem can be solved in polynomial time in the worst-case assumption that $P \neq NP$.

Reasons to study NP-hardness :

* NP-hard problems do not have to be decision problems.

$$3COL \leq_T^P COLO \leq_T^P COLC \Rightarrow \text{they are NP-hard}$$

* Sometimes interesting for decision problems as well. [are known to be in NP-hard but not in NP-complete]

COLE : Given 'G' and 'k' can a graph be painted with 'k' colours but no less?

$$3COL \leq_T^P COLE$$

As a graph is 3-colorable iff its χ is 0, 1, 2 or 3. ⇒ COLE is NP-hard.

* NP-hardness is often the only thing which is of practical use. Saves time in establishing a proof system. [for it in NP].