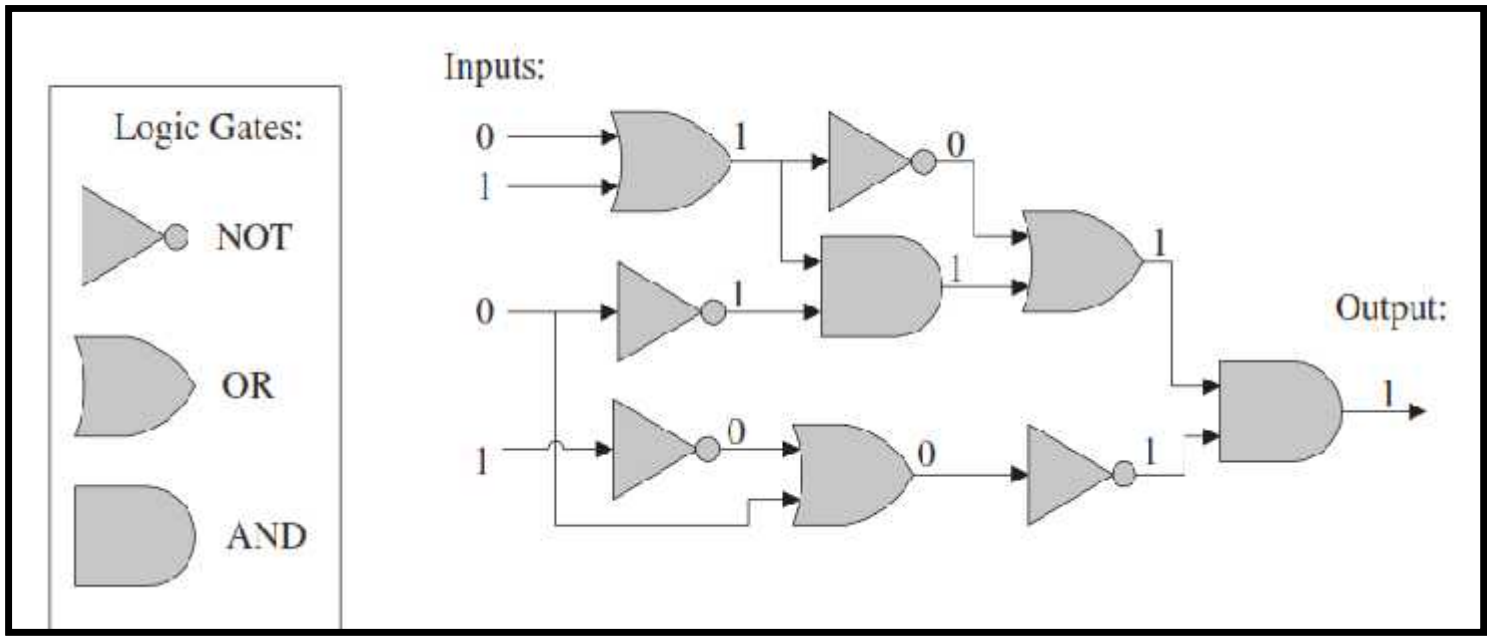
NP problems

Hamiltonian Cycle/Circuit is NP

- **Non Determinism:**
  Input is Graph G(V,E).
  Select the vertices from set “V”.
  Total number of selections = n+1
  If the selected vertex is already present in the set, then again re-start the selection. (Once unique selection set is generated, perform verification).
- **Verification:**
  Check if there exist edge between each two selected vertices and including “n” and “n+1”.
  The solution generation requires O(n) time and solution verification requires O(E) time.
  The algorithm results in “YES” or “NO” depending upon the condition.
  Hence HC problem is NP

Circuit SAT is NP

- Circuit SAT



Circuit SAT

- A Boolean circuit is called as directed graph, where each node is represented as Logic Gate [AND, OR, NOT].
- Circuit SAT problem takes input and “ask question whether assignment of input results in output = 1.
- Such problems are also called as “satisfying assignment” problems.
- **Non Determinism**

  First apply an arbitrary set of input x[1001] to the circuit and at each gate generate the output.
- **Verification**

  Check whether the final output is “1”, if not again change the input.

  The verification can be performed in polynomial time.

  Hence SAT problem is NP

Vertex cover problem is NP

- VC: It is set of vertices of graph, which covers all the edges present in the graph. All the edges of the graph are incident on the vertices present in the VC set.
- VC problem is to find a set of size [k], containing the vertices covering all the edges.
- **Non Determinism:**
  - Select the vertices from the given graph. The selection process is re-started if a vertex is selection is repeated. O(k)
- **Verification**
  - Verify if all the edges present in the graph are incident on the selected vertices O(ExE)

Application of VC

- Synonym tree / Ontology tree

The root word will have edges to all Possible synonyms.
The tree will be complex structure if constructed for more words.
The base idea can be useful in deriving The key word and synonym.

The key question to be answered:
Why non-deterministic approach is preferred or applied for solving problems like: SAT, VC or HC ?
Why deterministic approach is not feasible?

The other set of problems discussed during class room sessions: CLIQUE, GPT, ISP

NP Completeness:

- Any problem or language “M” is classified as **“NP-HARD”** if
- There exist a set of language “L” already classified as “NP”.
- L = {L1, L2, L3 ….. }
- If it is possible to perform the reduction of any one component of L to M, in polynomial time, then “M” is declared as “NP-Hard”
  - L ➔ M [In polynomial time]

- **The language “M” or problem “M” need not belong to class of “NP” problem for getting classified as NP-Hard.**

- **If the problem “M” is NP, with all above conditions, then it is called as NP-Complete problem.**

Proof for NP-Completeness

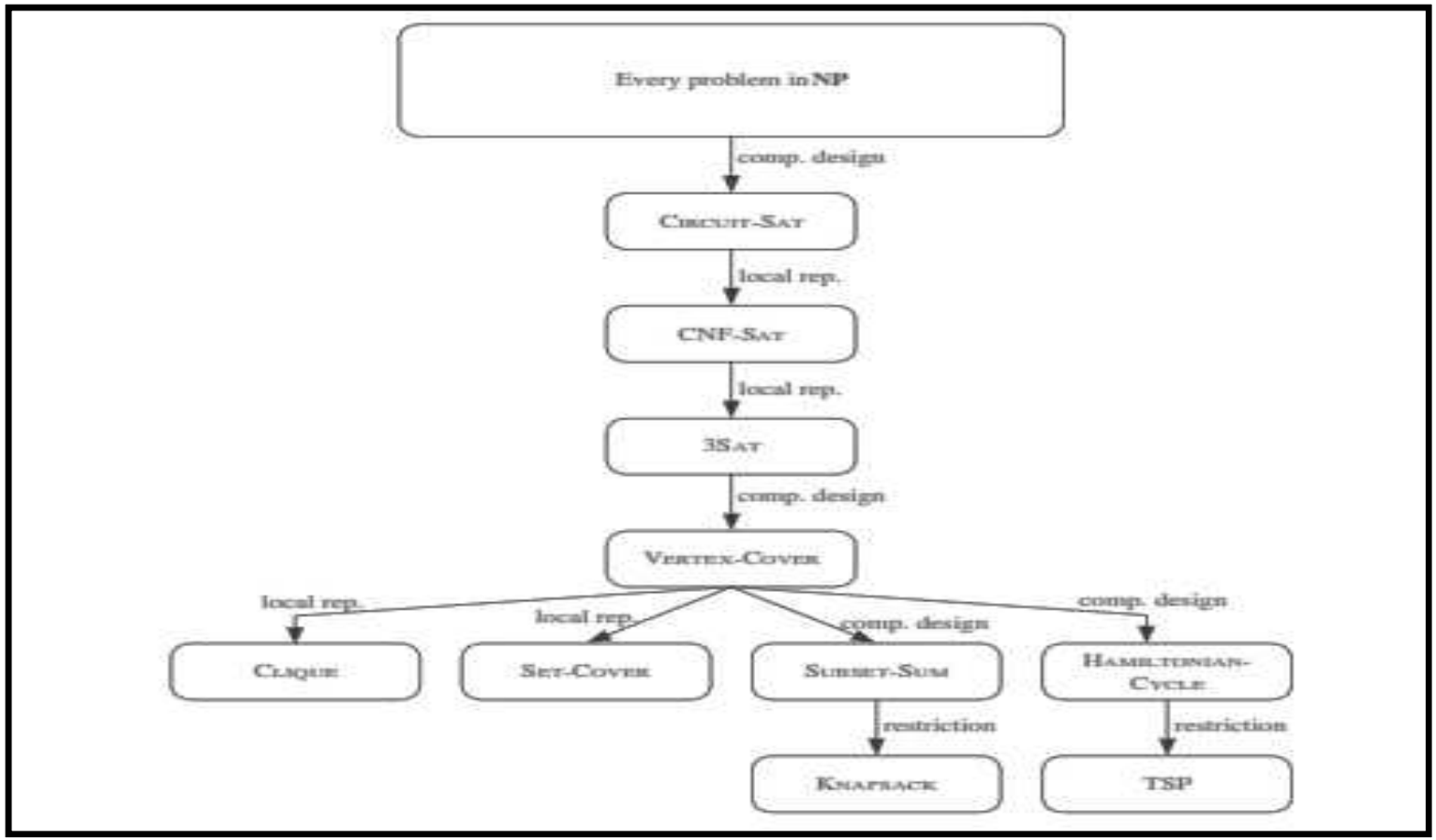
- Important concept
- If L1 → L2 [L1 reduces to L2 in polynomial time] and L2 → L3, then L1→L3.

Proof: Since $L_1 \xrightarrow{\text{poly}} L_2$, any instance x for L_1 can be converted in polynomial-time $p(n)$ into an instance $f(x)$ for L_2, such that $x \in L_1$ if and only if $f(x) \in L_2$, where n is the size of x. Likewise, since $L_2 \xrightarrow{\text{poly}} L_3$, any instance y for L_2 can be converted in polynomial-time $q(m)$ into an instance $g(y)$ for L_3, such that $y \in L_2$ if and only if $g(y) \in L_3$, where m is the size of y. Combining these two constructions, any instance x for L_1 can be converted in time $q(k)$ into an instance $g(f(x))$ for L_3, such that $x \in L_1$ if and only if $g(f(x)) \in L_3$, where k is the size of $f(x)$. But, $k \leq p(n)$, since $f(x)$ is constructed in $p(n)$ steps. Thus, $q(k) \leq q(p(n))$. Since the composition of two polynomials always results in another polynomial, this inequality implies that $L_1 \xrightarrow{\text{poly}} L_3$. ■

NP Completeness: Proof (Three possibilities)

- ***Restriction***: This form shows a problem L is *NP*-hard by noting that a known *NP*-complete problem M is actually just a special case of L.
- ***Local replacement***: This forms reduces a known *NP*-complete problem M to L by dividing instances of M and L into "basic units," and then showing how each basic unit of M can be locally converted into a basic unit of L.
- ***Component design***: This form reduces a known *NP*-complete problem M to L by building components for an instance of L that will enforce important structural functions for instances of M. For example, some components might enforce a "choice" while others enforce an "evaluation" function.

NP Completeness: Problem Tree



NP Completeness: Example: Clique & Vertex Cover

A ***clique*** in a graph G is a subset C of vertices such that, for each v and w in C, with $v \neq w$, (v, w) is an edge. That is, there is an edge between every pair of distinct vertices in C. Problem CLIQUE takes a graph G and an integer k as input and asks whether there is a clique in G of size at least k.

From previous discussion, it can be proved that Clique is NP.
If it is required to prove Clique is NP Complete, we have to prove:

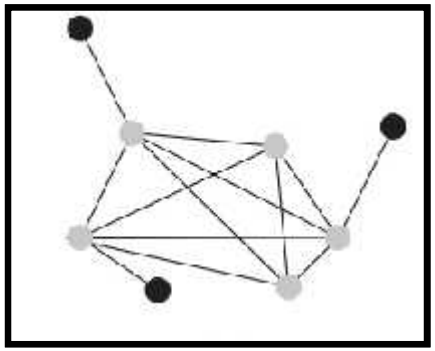
1. **Clique is NP *[refer the class discussion and previous slides]***
2. **Clique is NP Hard**

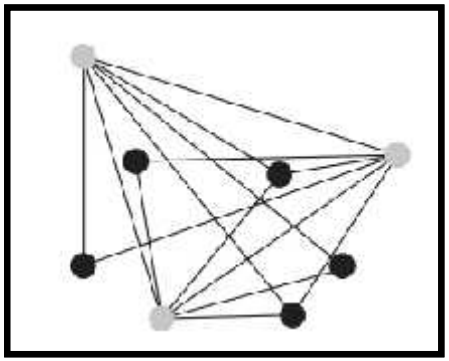
To prove Clique is NP-Hard, find a problem L, which is NP and can be reduced to Clique in polynomial time.
And
If both above claims are proved, then Clique is NP-Complete.
In the following proof, Vertex Cover problem is used for proving Clique as NP-Complete

a. **Vertex Cover is NP**
b. **The solution of vertex cover should be reduced in polynomial time to solve Clique**
c. ***For reduction, one of the three possibilities discussed, can be used.***

Also it is important to formulate a problem between Clique and VC, and use solution of one to derive solution of other problem.

Proof: Consider G1 and G2.





Let (G,k) be the instance of Vertex Cover problem. For clique, construct a Compliment Graph, which has:

- **Same number of vertices**
- **But**
- **Compliment edges**

We define integer parameter for clique as n-k where "k" is integer parameter for Vertex Cover.

If G has clique of size (n-k) if and only if its compliment graph has Vertex cover of size "k"

Further proof can be carried out using LOCAL REPLACEMENT

Some applications of NP problems

Example 13.10: *Suppose we are given a graph G representing a computer network where vertices represent routers and edges represent physical connections. Suppose further that we wish to upgrade some of the routers in our network with special new, but expensive, routers that can perform sophisticated monitoring operations for incident connections. If we would like to determine if k new routers are sufficient to monitor every connection in our network, then we have an instance of* VERTEX-COVER *on our hands.*

Example 13.14: *Suppose we have an Internet web server, and we are presented with a collection of download requests. For each each download request we can easily determine the size of the requested file. Thus, we can abstract each web request simply as an integer—the size of the requested file. Given this set of integers, we might be interested in determining a subset of them that exactly sums to the bandwidth our server can accommodate in one minute. Unfortunately, this problem is an instance of* SUBSET-SUM. *Moreover, because it is NP-complete, this problem will actually become harder to solve as our web server's bandwidth and request-handling ability improves.*

Reduction Principle

We say that a language L, defining some decision problem, is ***polynomial-time reducible*** to a language M, if there is a function f computable in polynomial time, that takes an input x to L, and transforms it to an input $f(x)$ of M, such that $x \in L$ if and only if $f(x) \in M$. In addition, we use a shorthand notation, saying $L \xrightarrow{\text{poly}} M$ to signify that language L is polynomial-time reducible to language M.

We say that a language M, defining some decision problem, is ***NP-hard*** if every other language L in ***NP*** is polynomial-time reducible to M. In more mathematical notation, M is ***NP***-hard, if, for every $L \in \boldsymbol{NP}$, $L \xrightarrow{\text{poly}} M$. If a language M is ***NP***-hard and it is also in the class ***NP*** itself, then M is ***NP-complete***. Thus, an ***NP***-complete problem is, in a very formal sense, one of the hardest problems in ***NP***, as far as polynomial-time computability is concerned. For, if anyone ever shows that an ***NP***-complete problem L is solvable in polynomial time, then that immediately implies that every other problem in the entire class ***NP*** is solvable in polynomial time. For, in this case, we could accept any other ***NP*** language M by reducing it to L and then running the algorithm for L. In other words, if anyone finds a deterministic polynomial-time algorithm for even one ***NP***-complete problem, then $\boldsymbol{P = NP}$.