P vs. NP vs. NP-Hard vs. NP-complete

- We have been discussing **Complexity Theory**:
  - classification of problems according to their difficulty.
- We introduced the classes P, NP, NP-hard and NP-complete.
  - P = {Decision problems solvable in polynomial time}.
  - NP = {Decision problems that are "verifiable" in polynomial time}.
- A major open question in theoretical computer science is:

  P = NP or P $\subset$ NP?

Polynomial time reductions

We also introduced the notion of **polynomial time reductions**:

- A $\leq_p$ B: A is polynomial-time reducible to B if there exists a "fast", i.e., poly-time, transformation algorithm F, such that $\forall$ instance α of A:
  - F(α) is an instance of B.
  - the answer of A for α is "yes" $\Leftrightarrow$ the answer of B for F(α) is "yes".

NP-Complete problems

- A decision problem L is in NPC if

  a) L $\in$ NP

  b) L' $\leq_p$ L for all L' $\in$ NP (L is NP-hard).

- If any NPC-problem can be solved in polynomial time, then every NPC-problem has a polynomial-time solution.

- By now, a lot of problems have been proved NP-Complete.

- Many smart-person-years have been spent on trying to solve NPC problems efficiently, to no avail.

  $\Rightarrow$ We regard L $\in$ NPC as strong evidence for L being hard!

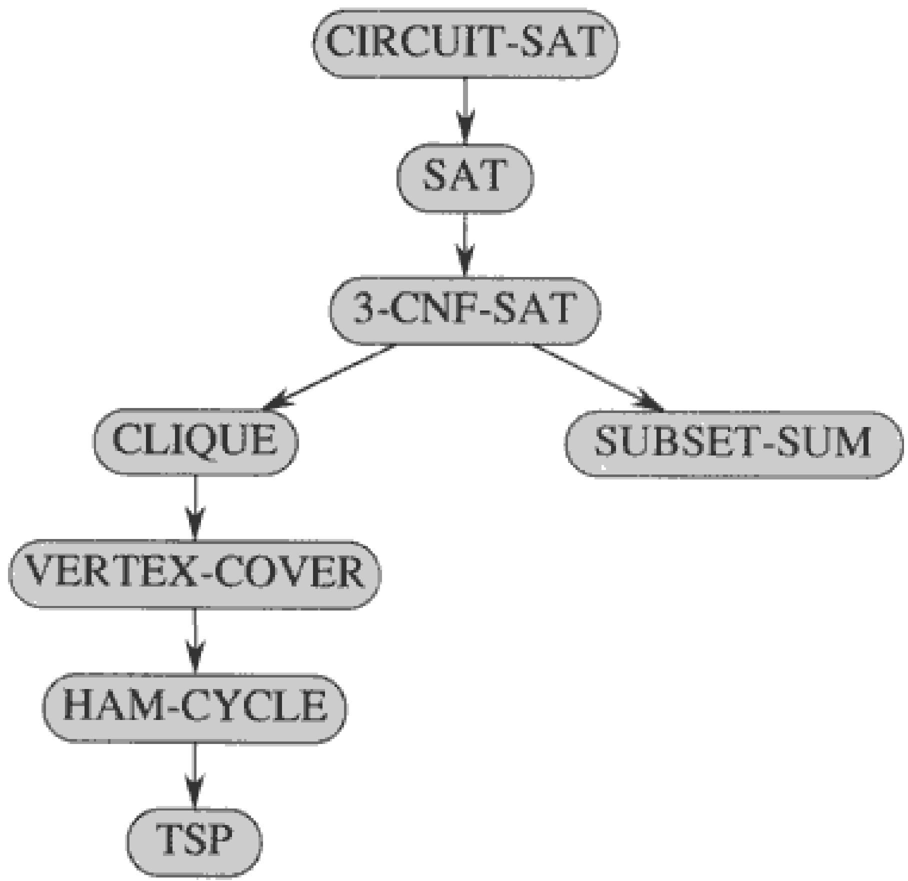
NP-Completeness Proofs

To prove a decision problem (language) L is NPC:

- Step 1: prove L $\in$ NP.
- Step 2: prove L $\in$ NP-hard.
  1. Select a known NPC problem (language) L'.
  2. Find a mapping algorithm (reduction) F, such that X $\in$ L' $\Leftrightarrow$ F(x) $\in$ L.
  3. Prove that the algorithm F runs in poly-time.

Up to this point, the only NPC problems we know are CKT-SAT and SAT.

Some NP-Complete problems



3CNF

- **Definition:**
  - A literal is x_i or $\neg x_i$.
  - A clause is $L_1 \vee L_2 \vee ... \vee L_k$, where L_i = literal.
  - A formula is in Conjunctive Normal Form (CNF) if it has the form:

    $$C_1 \wedge C_2 \wedge C_3 \wedge ... \wedge C_r, \text{ where } C_i = \text{clause}.$$

- This boolean formula is in **3-CNF**:
  $(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$
  - The first of its three clauses is $(x_1 \vee \neg x_3 \vee \neg x_2)$, which contains the three literals $x_1, \neg x_3$, and $\neg x_2$.

3CNF-SAT

- 3CNF-SAT is the problem of deciding if a formula in 3-CNF is satisfiable.
- 3CNF-SAT $= \{\langle\phi\rangle$: ϕ is a satisfiable formula in CNF with 3 literals per clause$\}$.

3CNF-SAT is NP-Complete

Recall, to prove a problem is NPC:

- Step 1: prove L $\in$ NP.
- Step 2: prove L $\in$ NP-hard.
  1. Select a known NPC problem (language) L'.
  2. Find a mapping algorithm (reduction) F, such that X $\in$ L' $\Leftrightarrow$ F(x) $\in$ L.
  3. Prove the algorithm F runs in poly-time.

Up to this point, the only NPC problems we know are CKT-SAT and SAT.

Step 1: 3CNF-SAT is NP

Step 1: 3CNF-SAT is NP.

- Easy – the certificate is the “truth assignment”, we replace each variable in the formula with its corresponding value and then evaluate the expression.

Step 2: 3CNF-SAT is NP-hard

Step 2: 3CNF-SAT is NP-hard by proving SAT $\leq_p$ 3CNF-SAT.

- Starting with an instance ϕ of SAT, we need to find a poly-time reduction algorithm F, such that:

  $$\phi \in \text{SAT} \Leftrightarrow \text{F}(\phi) \in \text{3CNF-SAT}.$$

  In other words:

  $$\phi \text{ is satisfiable} \Leftrightarrow \text{F}(\phi) \text{ is 3-CNF and satisfiable}.$$

Step 2: Outline

We will show how to transform any formula ϕ into an equivalent formula in 3CNF in three steps:

- **step 2.1**, transform ϕ into ϕ', which is a conjunction ($\wedge$) of clauses with at most three literals. ϕ is equivalent to ϕ'.
- **step 2.2**, transform ϕ' into ϕ'', by rewriting each of the clauses of ϕ' in conjunctive normal form (CNF). ϕ' is equivalent to ϕ''.
- **step 2.3**, transform ϕ'' into ϕ''', which is a 3-CNF. ϕ'' is equivalent to ϕ'''.

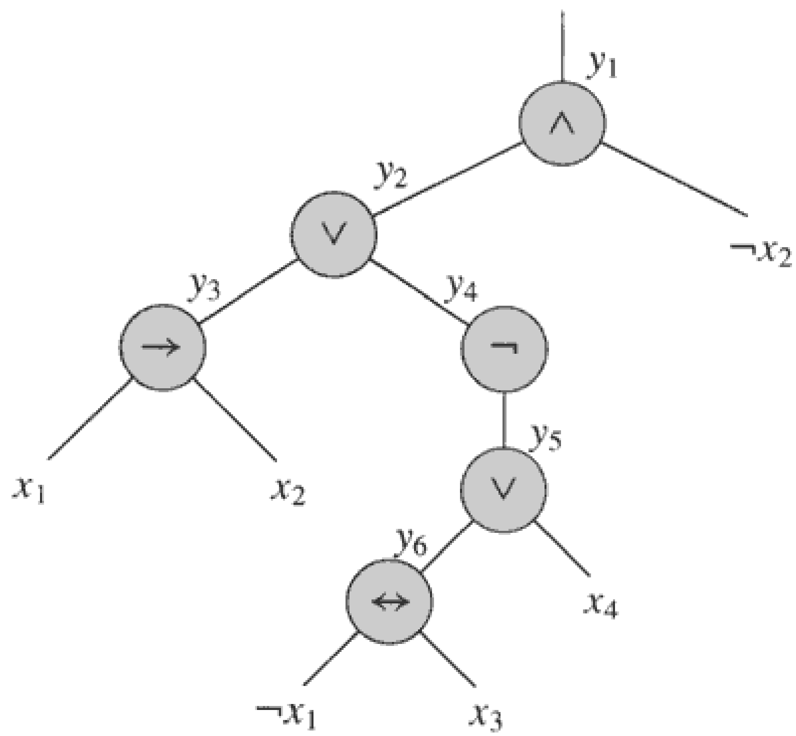
Step 2.1: ϕ to ϕ'

- **step 2.1**, We transform ϕ into ϕ', which is a conjunction ($\wedge$) of clauses with at most three literals.

  To do so we first construct a "parse tree" from ϕ with literals as leaves and connectives as internal nodes.

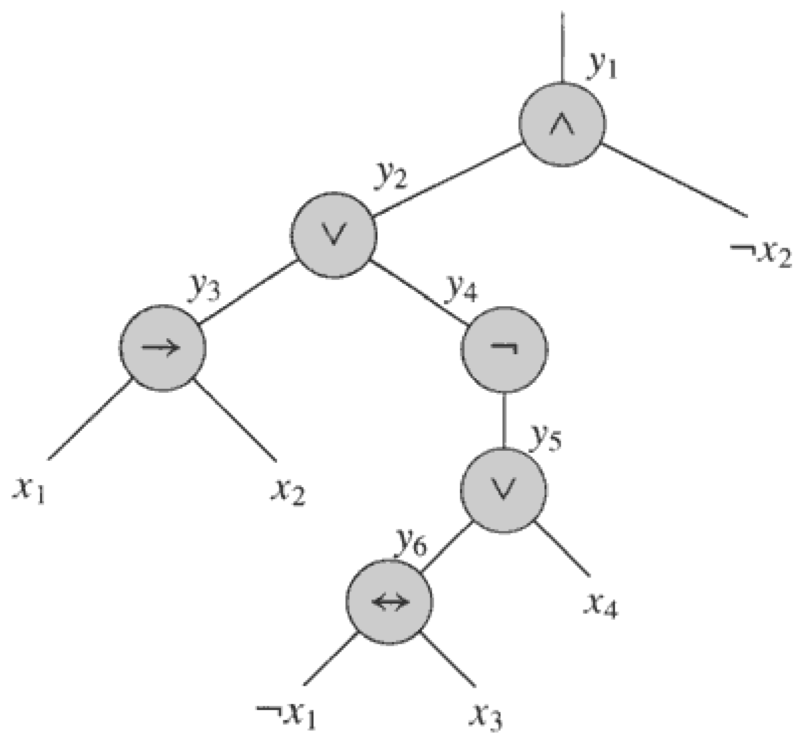
Step 2.1: ϕ to ϕ'

Example: $\phi = ((x_1 \to x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$.



Step 2.1: ϕ to ϕ'

We then introduce variables for the output of each node and rewrite formula as the AND of root edge and the formulas corresponding to internal edges.



Cont'd

$$\begin{aligned}\phi' = y_1 \; & \wedge(y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\ & \wedge(y_2 \leftrightarrow (y_3 \vee y_4)) \\ & \wedge(y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\ & \wedge(y_4 \leftrightarrow \neg y_5) \\ & \wedge(y_5 \leftrightarrow (y_6 \vee x_4)) \\ & \wedge(y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))\end{aligned}$$

Step 2.1: $\phi = \phi'$

1) ϕ' satisfied $\Rightarrow$ each tree edge clause corresponds to node value and $y_1 = 1 \Rightarrow$ conjunctions in ϕ' satisfied $\Rightarrow \phi$ satisfied.

2) ϕ satisfied $\Rightarrow$ parse tree satisfied $\Rightarrow \phi'$ satisfied.

Put 1) and 2) together, ϕ' is equivalent to ϕ and formula (ϕ') is now a Conjunction (AND) of clauses of at most three literals.

Step 2.2: ϕ' to ϕ''

Step 2.2: We transform each clause of ϕ' into conjunctive normal form (CNF)

- To do so for clause ϕ'_i, we first construct truth table for ϕ'_i. Using only entries that evaluate to 0, we then construct a formula equivalent to $\neg\phi'_i$.
- Example: Clause $\phi'_i = y_1 \leftrightarrow (y_2 \wedge \neg x_2)$

y_1	y_2	x_2	$y_1 \Leftrightarrow (y_2 \wedge \neg x_2)$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

Step 2.2: ϕ' to ϕ''

New formula:
$\neg\phi'_i = (y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$

Convert this formula into CNF using DeMorgan's laws:

$$\begin{aligned} \neg(A \vee B) &\equiv \neg A \wedge \neg B \\ \neg(A \wedge B) &\equiv \neg A \vee \neg B \end{aligned}$$

For example, formula ϕ''_i is defined as $\neg(\neg\phi'_i)$, which is:
$(\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$

Formula ϕ'', equivalent to ϕ', is obtained from the ϕ''_is.

Step 2.3: ϕ'' to ϕ'''

Step 2.3: Make each clause with less than 3 literals have exactly three literals.

- If a clause contains two literals $l_1 \vee l_2$, we replace it with the equivalent three literal clause:
  $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$.

- If a clause contains one literal l, we replace it with the equivalent clause:
  $(l \vee p \vee q) \wedge (l \vee p \vee \neg q) \wedge (l \vee \neg p \vee q) \wedge (l \vee \neg p \vee \neg q)$

We have obtained formula ϕ''' in 3CNF equivalent to ϕ.

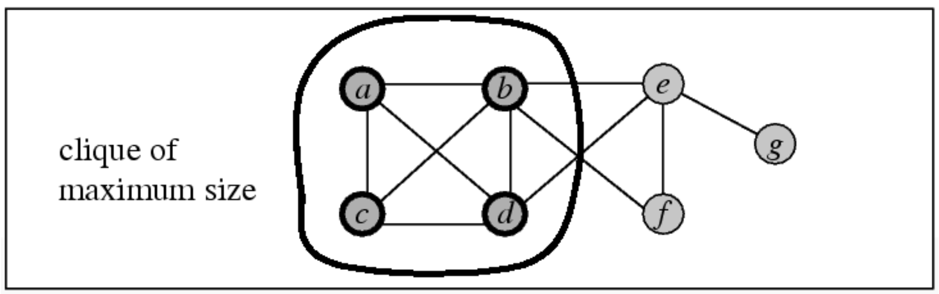
Step 3: Prove Polynomial Time Reduction

- The only thing left to prove is that we can perform the three steps in polynomial time. Easy since:
  - **step 2.1** introduces one new variable and clause per connective.
  - **step 2.2** introduces at most $2^3 = 8$ clauses for each old clause.
  - **step 2.3** introduces at most 4 clauses per clause

$\Rightarrow$ size of ϕ''' is polynomial in size of ϕ.

CLIQUE

- CLIQUE: Given a graph $G = (V, E)$, decide if there is a subset $V' \subseteq V$ of size k such that there is an edge between every pair of vertices in V' (i.e., V' makes a complete subgraph of G).



- The decision problem is to ask whether a clique of a given size k exists in the graph.

CLIQUE $= \{< G, k >: G$ is a graph with a clique of size $k\}$

CLIQUE is NPC

Theorem: CLIQUE is NP-complete.
Proof: It suffices to show that

- Step 1: CLIQUE $\in$ NP, and
- Step 2: 3CNF-SAT $\leq_p$ CLIQUE.

- Step 1: CLIQUE is NP.

  Proof: Given a subset V' as a certificate, we can check if V' makes a clique, i.e., check if for each pair $u, v \in V'$, $(u, v) \in E$. This checking can be done in $O(|V'|^2)$ time. So CLIQUE is NP.

Step 2: CLIQUE is NP-hard by 3CNF-SAT $\leq_p$ CLIQUE

It's somewhat surprising since formulas seem to have little to do with graphs.

- We need to find a transformation algorithm F, such that for each input instance ϕ of 3CNF-SAT, F can transform ϕ into an input instance for CLIQUE.
- ϕ is a 3CNF. An example is $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.
- An input instance for CLIQUE is a $< G, k >$, where G is a graph, and k is an integer.

Construction

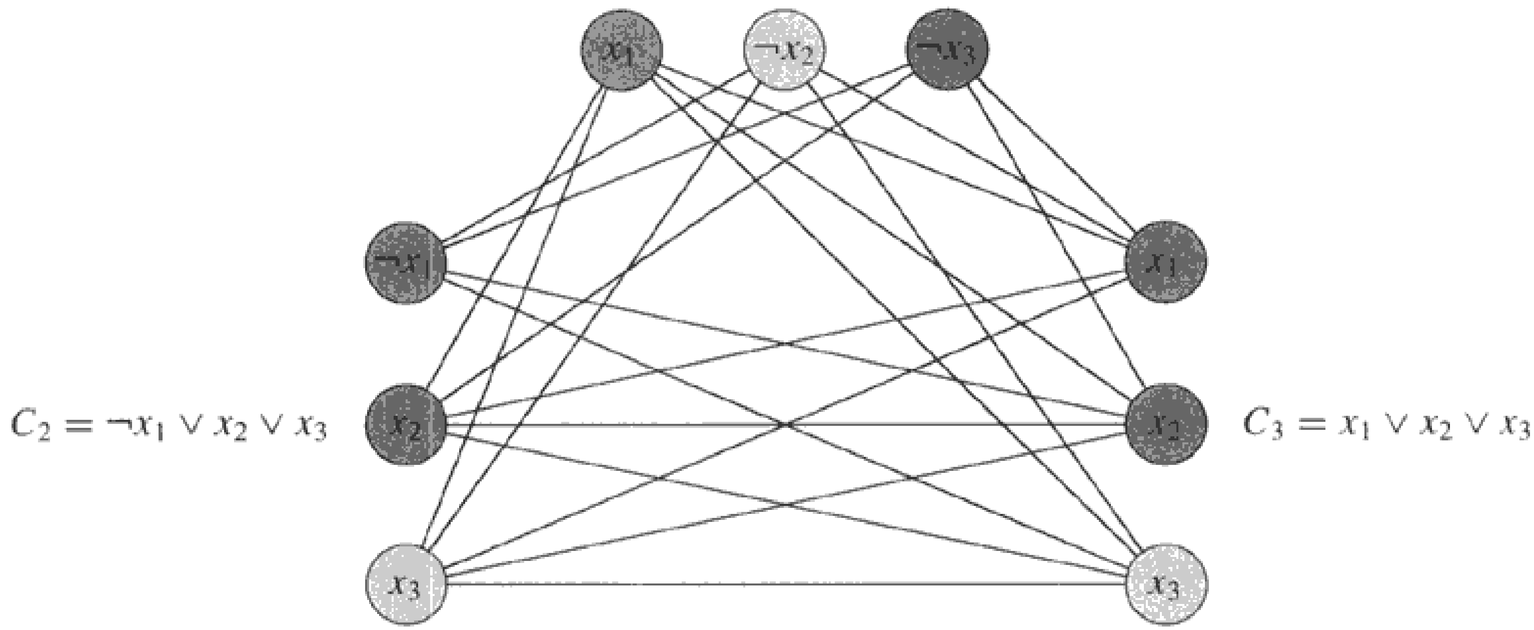
- We construct a graph $G = (V, E)$ from a k clause formula $\phi = C_1 \wedge C_2 \wedge C_3 ... \wedge C_k$ in 3-CNF.
- For each clause $C_r = (l_1^r \vee l_2^r \vee l_3^r)$, we place triple of vertices v_1^r, v_2^r, v_3^r in V.
- Vertices v_i^r and v_j^s are connected if:

  a) $r \neq s$.

  b) l_i^r and l_j^s are consistent (not negative of each other).

Construction example

Example: $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.



- Graph can be constructed in polynomial time

To prove $\phi \in$ 3CNF-SAT $\Leftrightarrow < G, k > \in$ CLIQUE

- We have ϕ satisfiable $\Leftrightarrow G$ has clique of size k:

  (Example: ϕ satisfiable by $x_2 = 0, x_3 = 1, x_1 = 0$ or 1 and the set of white vertices $(\neg x_2, x_3, x_3)$ is a clique of size 3.)

Proof:

- $(\Rightarrow)$
  - Each clause C_r contains at least one literal l_i^r assigned 1.
  - Each such literal corresponds to vertex v_i^r; pick such a vertex in each clause $\Rightarrow k$ vertices V'.

($\Rightarrow$), Cont'd

- For any two vertices $v_i^r, v_j^s \in V'(r \neq s)$, both corresponding literals l_i^r and l_i^s are mapped to 1

  $\Rightarrow$ they are not complements

  $\Rightarrow$ edge in G between v_i^r and v_j^s

  $\Rightarrow$ V' is a clique

$(\Leftarrow)$

- Let V' be clique of size $k \Rightarrow V'$ contains exactly one vertex for each triple (no edges between vertices in triple)
- We can assign 1 to each literal l_i^r corresponding to $v_i^r \in V$ since G contains no edges between inconsistent literals.
- Each clause is satisfiable $\Rightarrow \phi$ is satisfiable.