NP-Completeness

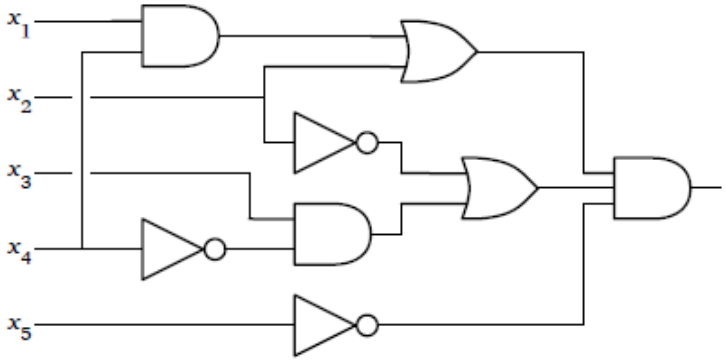
Hardness of Problems

- We say that an algorithm is ***efficient*** if its running time is ***polynomial*** - $O(n^k)$ for some constant k.
- Problems that are solvable by polynomial-time algorithms are called *tractable* or *easy*.
- Problems that require superpolynomial time algorithms are called *intractable* or *hard*.
- "***NP-complete***" problems – status unknown :
  - no polynomial time algorithm has yet been discovered
  - no proof that no polynomial-time algorithm can exist.

Circuit Satisfiability Problem

- Input : a boolean circuit (a collection of AND, OR, and NOT gates connected by wires).
- The input to the circuit is a set of m boolean values $x_1, \ldots, x_m$.
- The output of the circuit is a single boolean value.
- Given specific input values, we can calculate the output of the circuit in *linear* time using depth-first-search, since we can compute the output of a k-input gate in O(k) time.

An AND gate, an OR gate, and a NOT gate.



A boolean circuit. inputs enter from the left, and the output leaves to the right.

Circuit Satisfiability Problem

- The circuit satisfiability problem asks, given a circuit, whether there is an input that makes the circuit output TRUE, or conversely, whether the circuit always outputs FALSE.
- Nobody knows how to solve this problem faster than just trying all 2^m possible inputs to the circuit (which requires *exponential* time.)
- Nobody has proved that this is the best we can do.

P, NP, co-NP

A ***decision problem*** is a problem whose output is either YES or NO.

- ***P*** : the set of decision problems that are solvable in polynomial time
- ***NP*** : the set of decision problems that are "verifiable" in polynomial time – if the answer is YES, then there is a "certificate" that can be checked in polynomial time.
- ***co-NP*** : the opposite of NP. If the answer is NO, then there is a "certificate" that can be checked in polynomial time.

NP, co-NP

- The circuit satisfiability problem is in NP.
  - If the answer is YES, then any set of *m* input values that produces TRUE output is a “certificate”.
  - We can verify the “certificate” in polynomial time by evaluating the circuit.
- Nobody knows whether the circuit satisfiability problem is in P or in co-NP, though it is widely believed that it is not in P or in co-NP.

co-NP

- TAUTOLOGY : Let ϕ be a boolean formula constructed from the boolean input variables $x_1, x_2, \ldots, x_k, \neg, \wedge, \vee, \Rightarrow$, .., and parentheses. The formula ϕ is a ***tautology*** if it evaluates to 1 for every assignment of 1 and 0 to the input variables.
- The problem of deciding whether a formula is a tautology is in co-NP.
  - If the answer is NO, then any set of *k* input variables that produces FALSE output is a "certificate".
  - We can verify the "certificate" in polynomial time by evaluating the formula.
- UNSAT : Given a formula, determine if it is unsatisfiable.

P, NP, co-NP

- Every decision problem in P is also in NP.
- Every decision problem in P is also in co-NP.
- Open question : Is P ≠ NP? ($1,000,000 Millennium Prize Problem)
- It seems easier to verify an answer than to solve a problem, but nobody knows how to prove it.
- Open question : Is NP ≠ co-NP?
- Even if we can verify YES answers quickly, there is no reason to believe we can also verify NO answers quickly. For example, there seems to be no short “certificate” to show that a boolean circuit is *not* satisfiable.

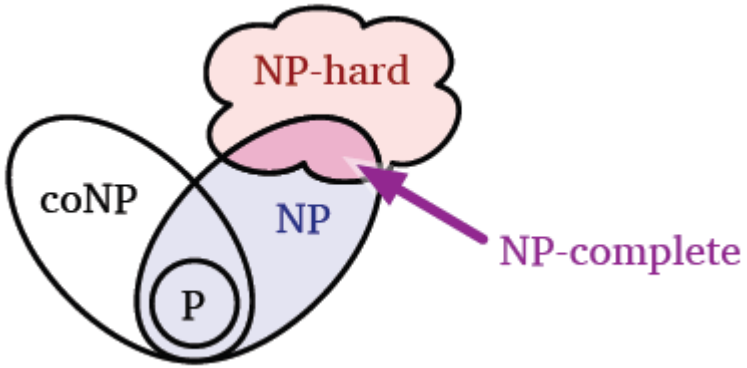
NP-hard, NP-complete

- A problem $\prod$ is ***NP-hard*** if a polynomial-time algorithm for $\prod$ would imply a polynomial-time algorithm for *every problem in NP*.
- (We can use the (mythical) polynomial-time algorithm for $\prod$ as a subroutine to solve any problem in NP in polynomial-time.)
- $\prod$ is NP-hard $\Leftrightarrow$ If $\prod$ can be solved in polynomial time, then P=NP.
- A problem is ***NP-complete*** if it is both NP-hard and in NP.
- NP-complete problems are the hardest problems in NP.

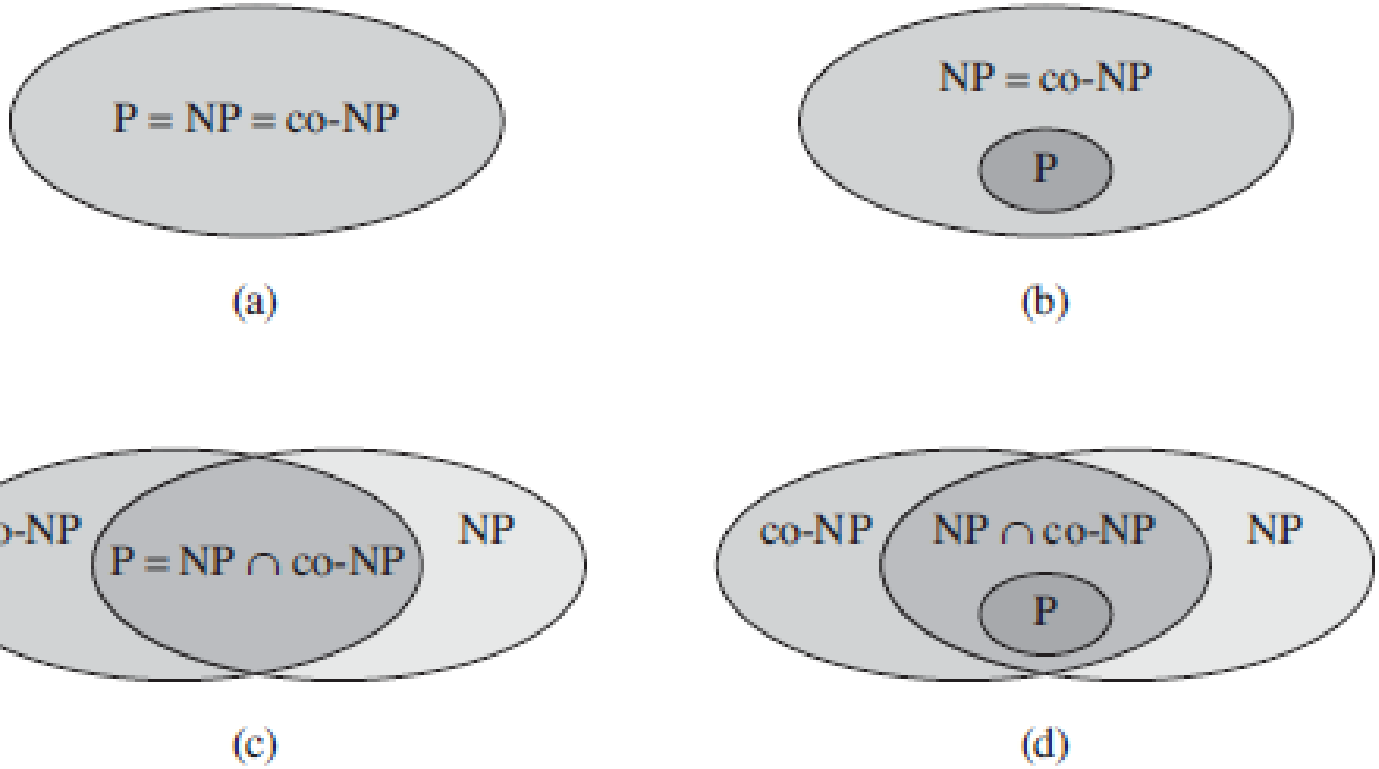
NP-complete

- If anyone finds a polynomial-time algorithm for one NP-complete problem, then that would imply a polynomial-time algorithm for *every* NP-complete problem.
- Thousands of problems have been shown to be NP-complete, so a polynomial-time algorithm for one (and therefore all) of them seems incredibly unlikely.

What we *think* the world looks like.



- (a) P = NP = co-NP. Most researchers regard this possibility as the most unlikely.
- (b) If NP is closed under complement, then NP = co-NP, but it need not be the case that P = NP.
- (c) P = NP ∩ co-NP, but NP is not closed under complement.
- (d) NP ≠ co-NP and P ≠ NP ∩ co-NP. Most researchers regard this possibility as the most likely.



The Cook-Levin Theorem

- Steve Cook (in 1971) and Leonid Levin (in 1973) independently published the following remarkable theorem.
- The Cook-Levin Theorem : Circuit satisfiability is NP-complete.

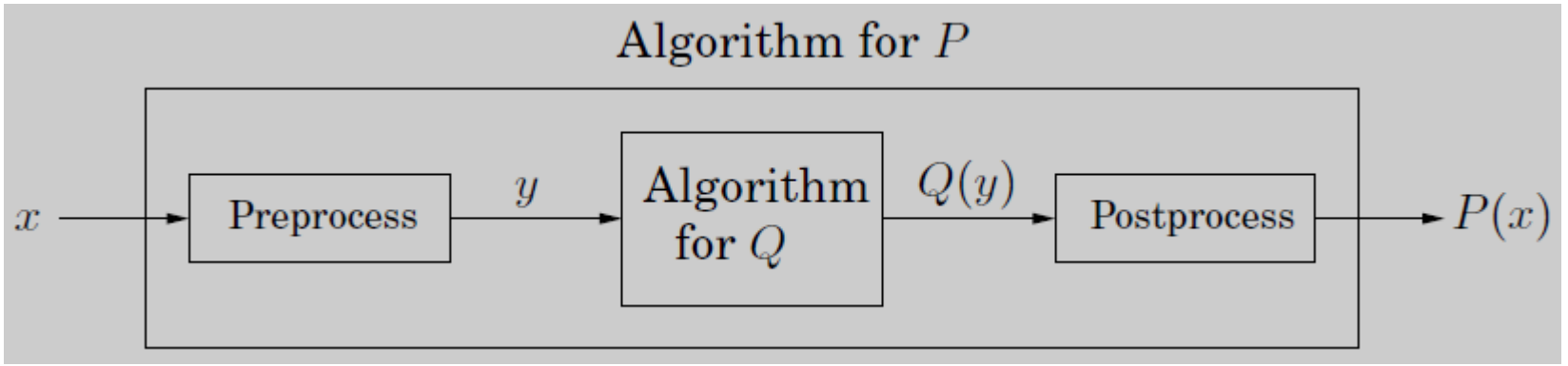
Reduction

- We saw that an algorithm for finding the longest path in a dag can be used for finding longest increasing subsequences.
- We say that the longest increasing subsequence problem ***reduces to*** the longest path problem in a dag.
- The longest path in a dag reduces to the shortest path in a dag :

  Function LONGEST PATH(G)

  negate all edge weights of G

  return SHORTEST PATH(G)
- If any subroutine for task Q can also be used to solve P, we say ***P reduces to Q*** ($P \leq_p Q$)



NP-hardness proofs

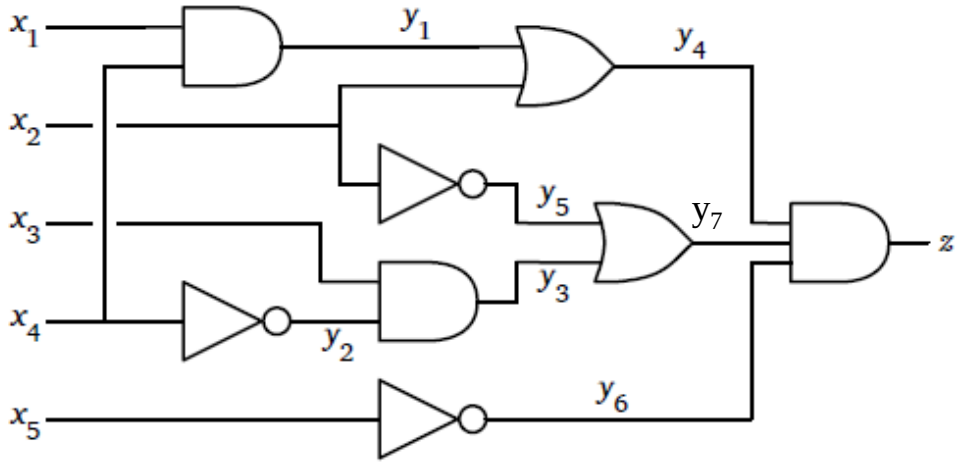
- To prove that a problem is NP-hard, we use a ***reduction*** argument.
- ***To prove that problem A is NP-hard, reduce a known NP-hard problem B to A*** **(** $B \leq_p A$)
- You need to describe an algorithm to solve B, which you already know is hard, using a mythical algorithm for A as a subroutine.
- The essential logic is a proof by contradiction.
  - If there is a polynomial-time algorithm for A then there is a polynomial-time algorithm for B (if A were easy, then B would be easy, too.)
- Equivalently, since B is hard, A must also be hard.
- By reduction from B to A, you showed that A is at least as hard as B.
- To prove that a problem A is ***NP-complete***,
  - Prove A is NP.
  - Prove A is NP-hard : reduce a known NP-hard problem B to A.

Formula Satisfiability Problem (SAT)

- A boolean formula ϕ consists of variables, x_1, x_2 , … , operators, $\neg$, $\wedge$, $\vee$, $\Rightarrow$, .., and parentheses.
- The formula satisfiability problem (SAT) asks whether it is possible to assign boolean values to the variables so that the formula evaluates to TRUE.
- To show that SAT is NP-complete.
  - 1) Show that SAT is NP.
  - 2) Show that SAT is NP-hard
- SAT is NP because if the formula is satisfiable, a satisfying truth assignment is the certificate that we can verify in polynomial (linear) time.
- To show that SAT is NP-hard, we use reduction from the circuit satisfiability problem.

Reduction from circuit satisfiability to SAT

- Given a boolean circuit, we can transform it into a boolean formula by creating new output variables for each gate and then writing down the list of gates separated by ANDs.

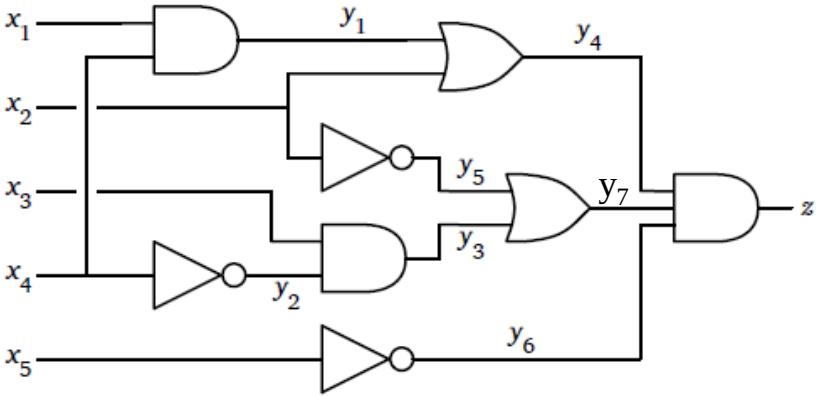


$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge$$
$$(y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

A boolean circuit with gate variables added, and an equivalent boolean formula.

Reduction from circuit satisfiability to SAT

- The original circuit is satisfiable if and only if the resulting formula is satisfiable.
- => Given a satisfying input to the circuit, we can get a satisfying assignment for the formula by computing the output of every gte.
- <= Given a satisfying assignment for the formula, we can get a satisfying input to the circuit by just ignoring internal and output variables.

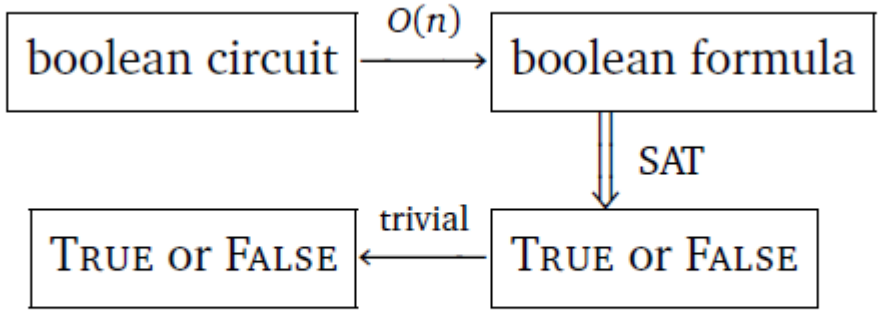


$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge$$
$$(y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

A boolean circuit with gate variables added, and an equivalent boolean formula.

Reduction from circuit satisfiability to SAT

- We can transform any boolean circuit into a formula in linear time using depth-first search.
- The size of the resulting formula is O(size of the circuit).
- Thus, we have a ***polynomial-time reduction*** from circuit satisfiability to SAT.
- The reduction implies that if we had a polynomial-time algorithm for SAT, then we'd have a polynomial-time algorithm for circuit satisfiability, which would imply that P = NP. So SAT is NP-hard.

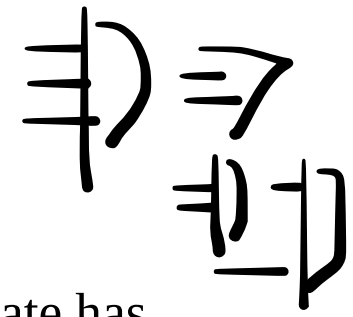


$$T_{\text{CSAT}}(n) \leq O(n) + T_{\text{SAT}}(O(n)) \quad \Longrightarrow \quad T_{\text{SAT}}(n) \geq T_{\text{CSAT}}(\Omega(n)) - O(n)$$

3-SAT

- Call a boolean variable or its negation a *literal*.
- A boolean formula is in *conjunctive normal form* (CNF) if it is a sequence of clauses connected by $\wedge$, and each clause is a sequence of literals connected by $\vee$.
- A 3-CNF formula is a CNF formula with exactly three distinct literals per clause.
- 3-SAT is SAT restricted to 3-CNF formula : Given a 3-CNF formula, is there an assignment to the variables that makes the formula evaluate to TRUE?
- We can prove that 3-SAT is NP-hard by SAT or circuit satisfiability.
- Lemma : SAT $\leq_p$ 3-SAT
- Proof : take a boolean formula ϕ and transform it into equivalent 3-CNF. (It's in the textbook (CLRS).)

Reduction from circuit satisfiability to 3-SAT



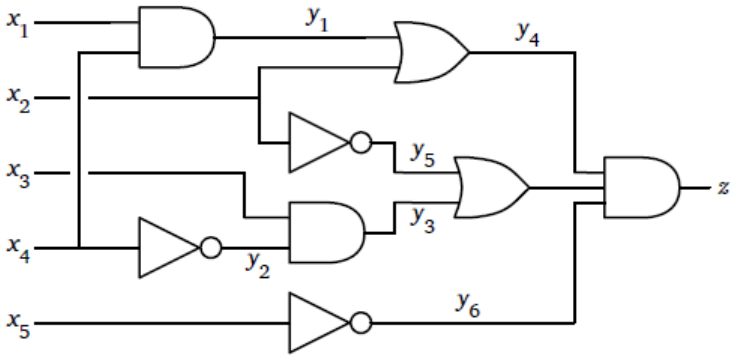
- We'll transform a boolean circuit into a 3-SAT.
- Make sure every AND, OR gate has only two inputs. If any gate has $k > 2$ inputs, replace it with a binary tree of k-1 two-input gates.
- Write down the circuit as a formula, with one clause per gate (as in the reduction from circuit satisfiability to SAT.)
- Change every gate clause into a CNF formula :

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

- Make sure every clause has exactly three literals by introducing new variables into one- and two-literal clauses as follows :

$$a \longmapsto (a \vee x \vee y) \wedge (a \vee \bar{x} \vee y) \wedge (a \vee x \vee \bar{y}) \wedge (a \vee \bar{x} \vee \bar{y})$$
$$a \vee b \longmapsto (a \vee b \vee x) \wedge (a \vee b \vee \bar{x})$$

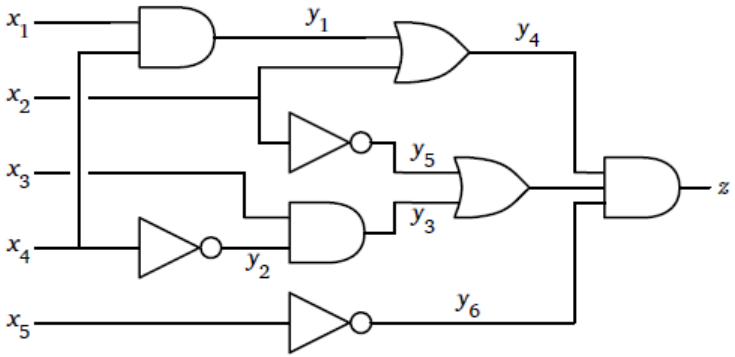
Reduction from circuit satisfiability to 3-SAT



$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge$$
$$(y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

A boolean circuit with gate variables added, and an equivalent boolean formula.

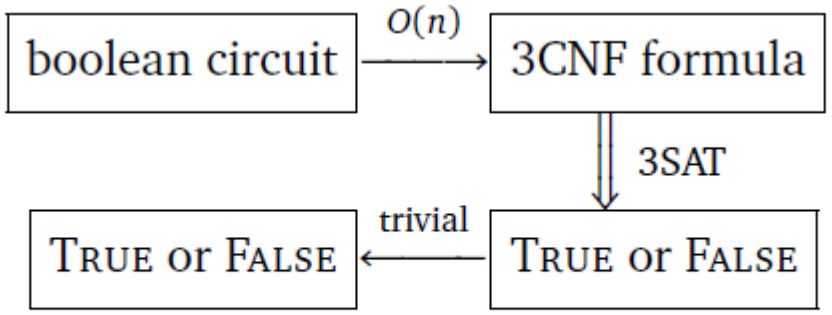
Reduction from circuit satisfiability to 3-SAT



$$(y_1 \vee \overline{x_1} \vee \overline{x_4}) \wedge (\overline{y_1} \vee x_1 \vee z_1) \wedge (\overline{y_1} \vee x_1 \vee \overline{z_1}) \wedge (\overline{y_1} \vee x_4 \vee z_2) \wedge (\overline{y_1} \vee x_4 \vee \overline{z_2})$$
$$\wedge (y_2 \vee x_4 \vee z_3) \wedge (y_2 \vee x_4 \vee \overline{z_3}) \wedge (\overline{y_2} \vee \overline{x_4} \vee z_4) \wedge (\overline{y_2} \vee \overline{x_4} \vee \overline{z_4})$$
$$\wedge (y_3 \vee \overline{x_3} \vee \overline{y_2}) \wedge (\overline{y_3} \vee x_3 \vee z_5) \wedge (\overline{y_3} \vee x_3 \vee \overline{z_5}) \wedge (\overline{y_3} \vee y_2 \vee z_6) \wedge (\overline{y_3} \vee y_2 \vee \overline{z_6})$$
$$\wedge (\overline{y_4} \vee y_1 \vee x_2) \wedge (y_4 \vee \overline{x_2} \vee z_7) \wedge (y_4 \vee \overline{x_2} \vee \overline{z_7}) \wedge (y_4 \vee \overline{y_1} \vee z_8) \wedge (y_4 \vee \overline{y_1} \vee \overline{z_8})$$
$$\wedge (y_5 \vee x_2 \vee z_9) \wedge (y_5 \vee x_2 \vee \overline{z_9}) \wedge (\overline{y_5} \vee \overline{x_2} \vee z_{10}) \wedge (\overline{y_5} \vee \overline{x_2} \vee \overline{z_{10}})$$
$$\wedge (y_6 \vee x_5 \vee z_{11}) \wedge (y_6 \vee x_5 \vee \overline{z_{11}}) \wedge (\overline{y_6} \vee \overline{x_5} \vee z_{12}) \wedge (\overline{y_6} \vee \overline{x_5} \vee \overline{z_{12}})$$
$$\wedge (\overline{y_7} \vee y_3 \vee y_5) \wedge (y_7 \vee \overline{y_3} \vee z_{13}) \wedge (y_7 \vee \overline{y_3} \vee \overline{z_{13}}) \wedge (y_7 \vee \overline{y_5} \vee z_{14}) \wedge (y_7 \vee \overline{y_5} \vee \overline{z_{14}})$$
$$\wedge (y_8 \vee \overline{y_4} \vee \overline{y_7}) \wedge (\overline{y_8} \vee y_4 \vee z_{15}) \wedge (\overline{y_8} \vee y_4 \vee \overline{z_{15}}) \wedge (\overline{y_8} \vee y_7 \vee z_{16}) \wedge (\overline{y_8} \vee y_7 \vee \overline{z_{16}})$$
$$\wedge (y_9 \vee \overline{y_8} \vee \overline{y_6}) \wedge (\overline{y_9} \vee y_8 \vee z_{17}) \wedge (\overline{y_9} \vee y_8 \vee \overline{z_{17}}) \wedge (\overline{y_9} \vee y_6 \vee z_{18}) \wedge (\overline{y_9} \vee y_6 \vee \overline{z_{18}})$$
$$\wedge (y_9 \vee z_{19} \vee z_{20}) \wedge (y_9 \vee \overline{z_{19}} \vee z_{20}) \wedge (y_9 \vee z_{19} \vee \overline{z_{20}}) \wedge (y_9 \vee \overline{z_{19}} \vee \overline{z_{20}})$$

Reduction from circuit satisfiability to 3-SAT

- This process transforms the circuit into an equivalent 3-CNF formula : The output formula is satisfiable if and only if the input circuit is satisfiable.
- The size of the output formula is only O(size of input circuit) and the reduction can be done in polynomial time.
- Thus, we have a polynomial-time reduction from circuit satisfiability to 3-SAT:



$$T_{\text{CSAT}}(n) \leq O(n) + T_{\text{3SAT}}(O(n)) \quad \Longrightarrow \quad T_{\text{3SAT}}(n) \geq T_{\text{CSAT}}(\Omega(n)) - O(n)$$

3-SAT

- Thus, 3SAT is NP-hard.
- And, it is also NP. (a satisfying assignment of the formula is the certificate that we can check in polynomial time.)
- So, 3-SAT is NP-complete.