# CSE331 Introduction to Algorithms
# Lecture 27
# Introduction to Computational Complexity III

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

July 23, 2021

# Introduction

- This is the the third part of our 3-lectures introduction to computational complexity.
- I will present several **NP**-complete problems. For some of them, I will prove that they are **NP**-hard through a reduction.
- **Reference**: Chapter 34 of the textbook

  Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein.

# Clique

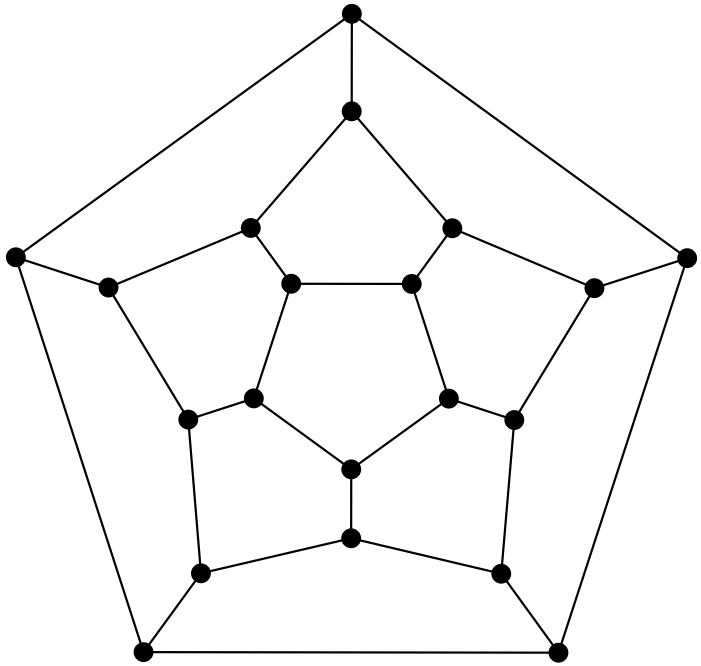
- In the previous two lectures, we saw that 3-SAT is **NP**-hard, 3-SAT$\leqslant_p$Vertex-cover and Vertex-cover$\leqslant_p$Clique.
- It follows that Clique is **NP**-hard.
- It is easy to see that Clique is in **NP**: We can check in polynomial time whether a subset of vertices of the input graph forms a clique.
- Therefore:

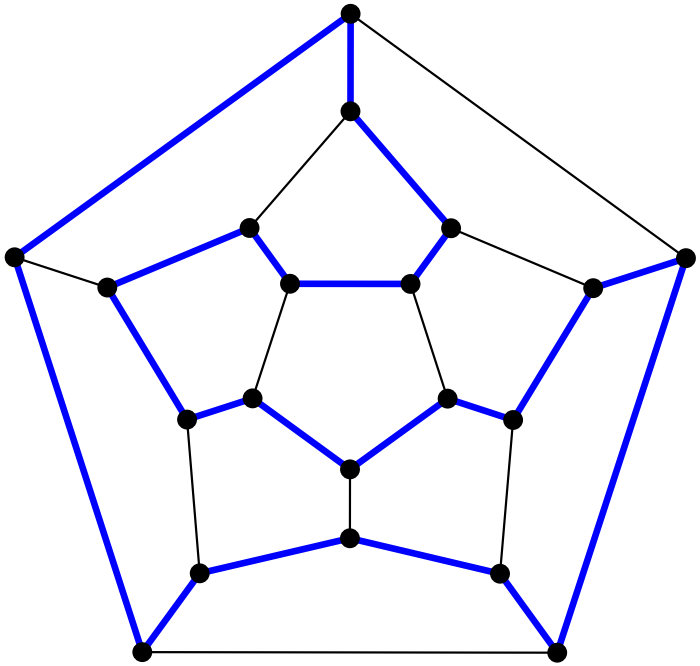
**Theorem**

Clique *is* **NP***-complete.*

# Hamiltonian Cycle

- Origin: A puzzle by Hamilton where the goal is to find a path that visits each city exactly once, and returns to the starting position, in the map below:





# Hamiltonian Cycle

## Definition

A *Hamiltonian cycle* of a graph is a cycle that visits each vertex exactly once.

## Problem (Ham-cycle)

*Given an input graph $G$, the Hamiltonian cycle problem is to decide whether $G$ has a Hamiltonian cycle.*

## Theorem

*The Hamiltonian cycle problem is* **NP***-complete.*

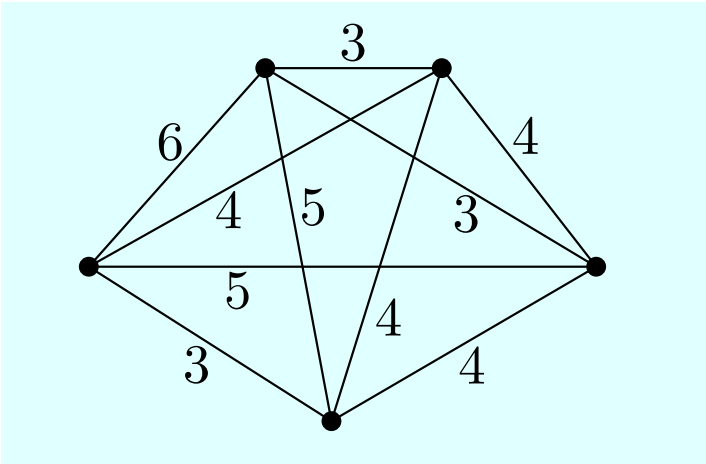
- The proof is not covered in this course. The textbook proves it by a reduction from vertex cover, i.e.

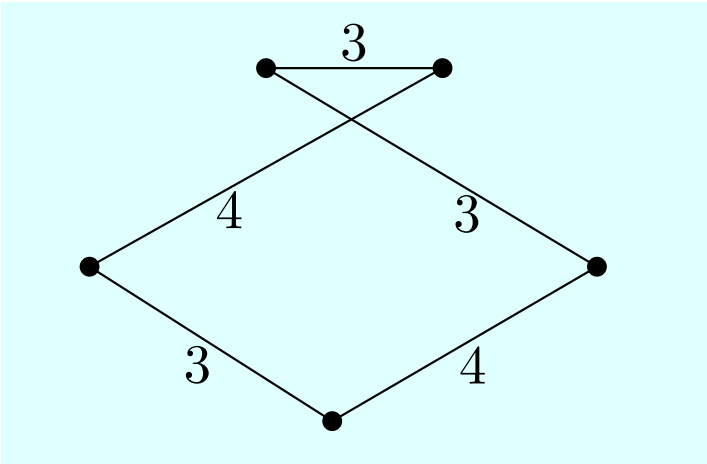
$$\textsc{Vertex-cover} \leqslant_p \textsc{Ham-cycle}.$$

# The Traveling Salesman Problem (TSP)

## Problem (TSP)

*Given a set of cities, and the distance between each pair of cities, find the shortest tour.*




- The input is given as an $n \times n$ matrix of distances $D = (d_{ij})$ where $d_{ij} = d_{ji} \geqslant 0$ and $d_{ii} = 0$ for all $i, j$.

# The Traveling Salesman Problem (TSP)

- TSP is not in **NP**, because it is an optimization problem.
- Therefore we introduce:

**Problem (DECIDE-TSP)**

*Given a TSP instance, and a number $k$, decide whether there is a tour of length at most $k$.*

**Proposition**

DECIDE-TSP *is* **NP***-complete.*

- Proof: see exercise set.

**Corollary**

*TSP is* **NP***-hard.*

# The Subset-Sum Problem

## Problem (SUBSET-SUM)

*Given a set $S$ of integers and a target number $t$, decide whether there is a subset $S' \subseteq S$ whose elements sum to $t$, that is,*

$$t = \sum_{x \in S'} x.$$

## Example

Suppose that $S = \{1, 2, 7, 8, 16\}$.

- If $t = 11$ then it is a positive instance because $1 + 2 + 8 = 11$.
- If $t = 12$ then it is a negative instance.

# The Subset-Sum Problem

**Theorem**

SUBSET-SUM *is* **NP***-complete.*

**Proof.**

- SUBSET-SUM $\in$ **NP** because we can verify in polynomial time whether a subset sums to $t$. (In fact it is trivially done in *linear* time.)
- SUBSET-SUM is **NP**-hard because 3-SAT $\leqslant_p$ SUBSET-SUM. The reduction is given in next slides: For any 3-SAT instance we construct in polynomial time an equivalent instance of SUBSET-SUM.

*Detailed proofs are in the textbook.* □

# The Subset-Sum Problem

| | | $x_1$ | $x_2$ | $x_3$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ |
|---|---|---|---|---|---|---|---|---|
| $\nu_1$ | = | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| $\nu'_1$ | = | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| $\nu_2$ | = | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| $\nu'_2$ | = | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| $\nu_3$ | = | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| $\nu'_3$ | = | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| $s_1$ | = | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $s'_1$ | = | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| $s_2$ | = | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $s'_2$ | = | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| $s_3$ | = | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $s'_3$ | = | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| $s_4$ | = | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $s'_4$ | = | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| t | = | 1 | 1 | 1 | 4 | 4 | 4 | 4 |

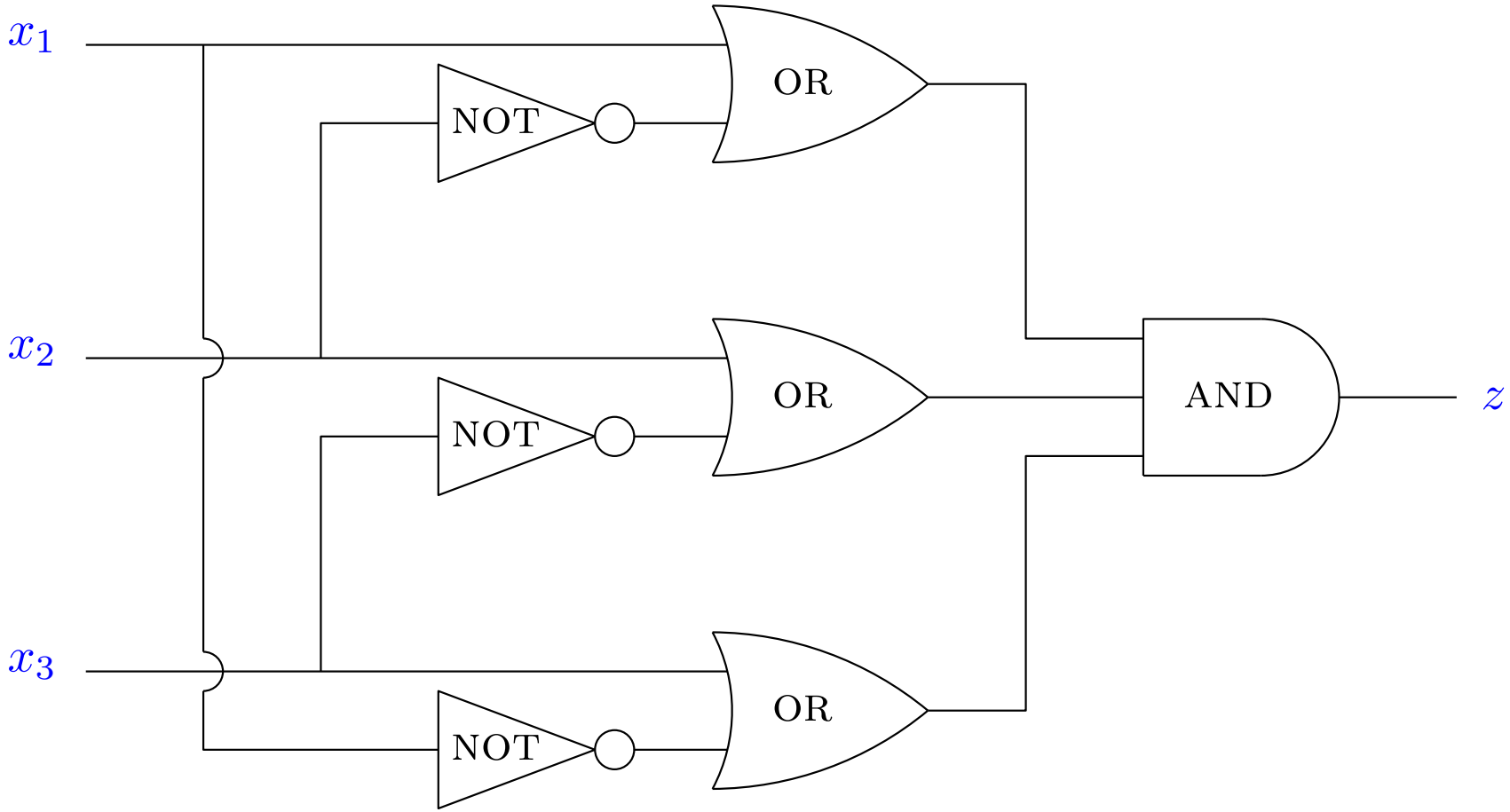
- $C_1 = x_1 \vee \neg x_2 \vee \neg x_3$
- $C_2 = \neg x_1 \vee \neg x_2 \vee \neg x_3$
- $C_3 = \neg x_1 \vee \neg x_2 \vee x_3$
- $C_4 = x_1 \vee x_2 \vee x_3$

# The Subset-Sum Problem

| | | $x_1$ | $x_2$ | $x_3$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ |
|---|---|---|---|---|---|---|---|---|
| $\nu_1$ | = | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| $\nu'_1$ | = | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| $\nu_2$ | = | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| $\nu'_2$ | = | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| $\nu_3$ | = | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| $\nu'_3$ | = | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| $s_1$ | = | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $s'_1$ | = | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| $s_2$ | = | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $s'_2$ | = | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| $s_3$ | = | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $s'_3$ | = | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| $s_4$ | = | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $s'_4$ | = | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| t | = | 1 | 1 | 1 | 4 | 4 | 4 | 4 |

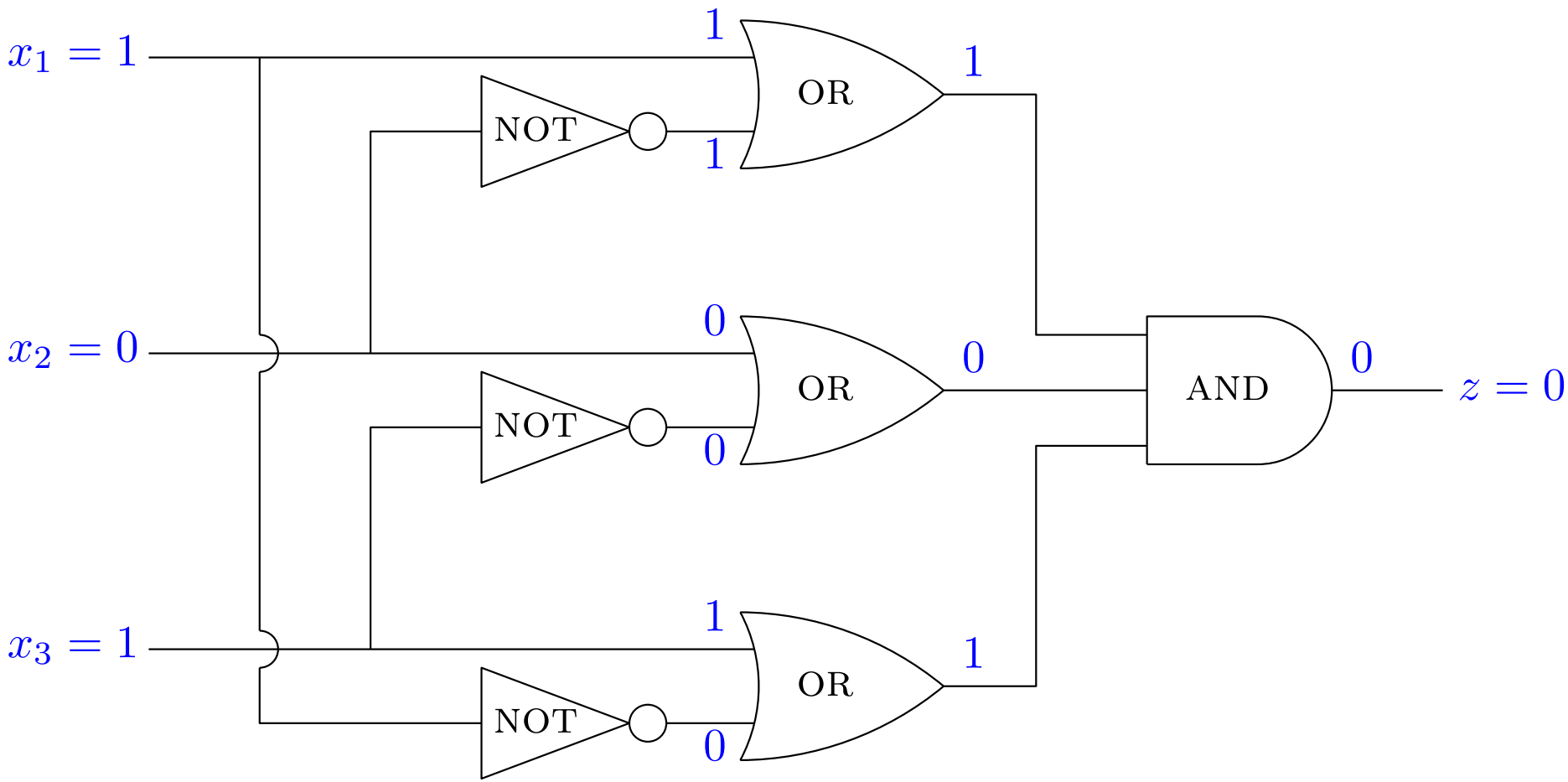
- $C_1 = \quad x_1 \vee \neg x_2 \vee \neg x_3$
- $C_2 = \neg x_1 \vee \neg x_2 \vee \neg x_3$
- $C_3 = \neg x_1 \vee \neg x_2 \vee \quad x_3$
- $C_4 = \quad x_1 \vee \quad x_2 \vee \quad x_3$
- Satisfying assignment $x_1 = 0$, $x_2 = 0$, $x_3 = 1$
- The shaded rows, regarded as numbers in base 10, sum to $t$.

# Circuit Satisfiability




- Circuit with input $x_1$, $x_2$, $x_3$ and output $z$.

# Circuit Satisfiability




- On input $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, output is $z = 0$.

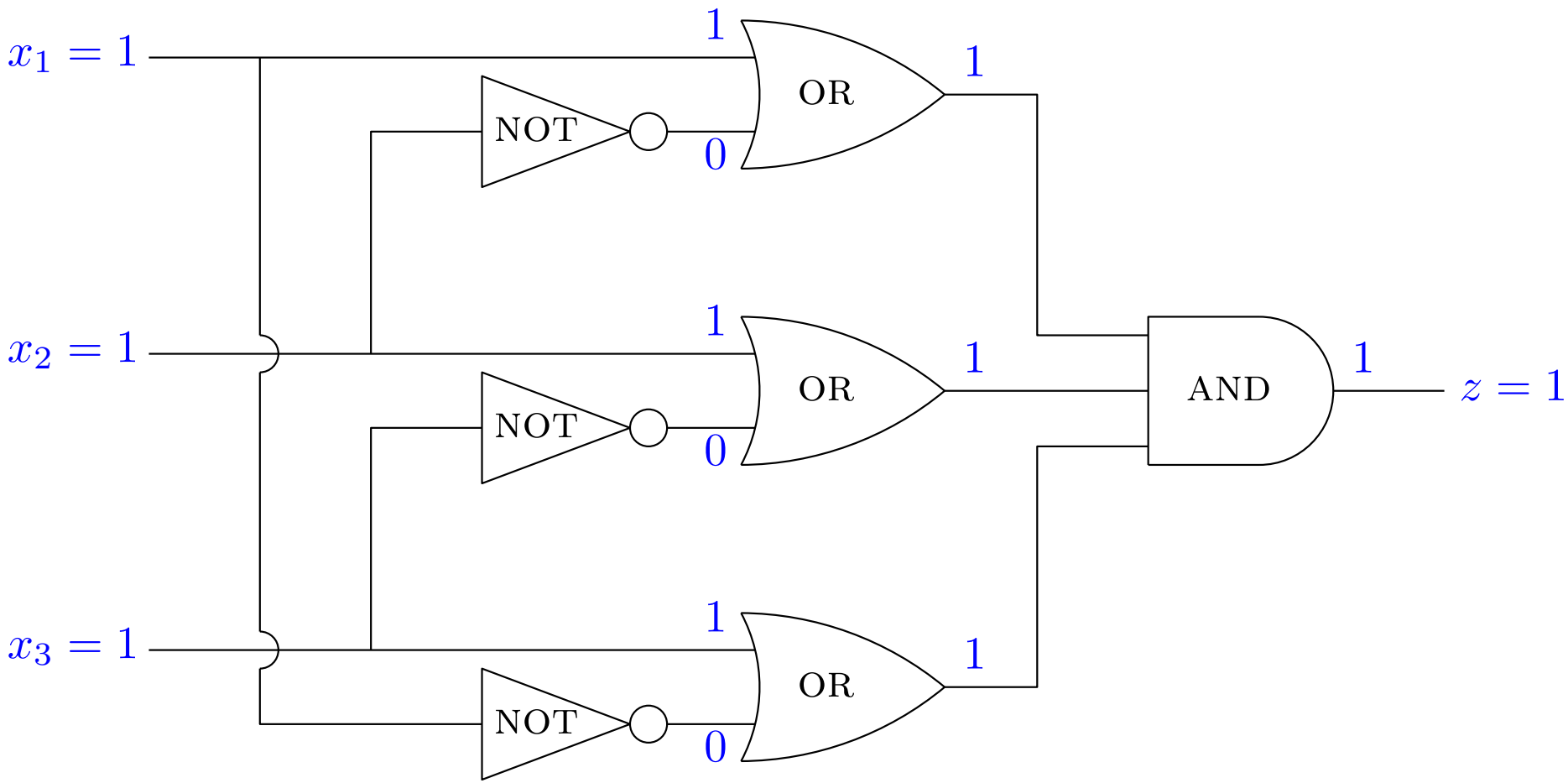
# Circuit Satisfiability

**Problem (Circuit-satisfiability)**

*Given a Boolean circuit, decide whether there is an input such that the output is 1.*

- Is the circuit above satisfiable?
- Yes: See next slide.

# Circuit Satisfiability




- A satisfying assignment $x_1 = 1$, $x_2 = 1$, $x_3 = 1$

# Circuit Satisfiability

**Theorem**

CIRCUIT-SATISFIABILITY *is* **NP***-complete.*

**Proof.**

(Sketch) We use a reduction from 3-SAT. Any 3-SAT instance can be converted into an equivalent circuit in linear time. □

**Example**

The circuit above computes the (2-SAT) formula

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1).$$

# Concluding Remarks on this Lecture

- A large variety of *combinatorial* problems are **NP**-complete. We saw graph algorithms problems, an arithmetic problem (SUBSET-SUM), a logic problem (3-SAT), a circuit problem ...
- In order to prove that a problem is **NP**-hard, we usually prove a reduction from a known **NP**-hard problem.
- These reductions are often simple, but not always.
- 3-SAT is often used in these reductions.

# Concluding Remarks on this Course

- We studied:
  - Algorithms analysis
  - Algorithms design techniques:
    - ★ Incremental algorithms (INSERTION SORT)
    - ★ Divide & conquer (MERGE SORT, Strassen's algorithm ...)
    - ★ Randomization (QUICKSORT, SELECTION)
    - ★ Dynamic programming (LCS, optimal BST)
    - ★ Greedy algorithms (Prim's algorithm)
  - Computational complexity:
    - ★ Lower bound for sorting
    - ★ **P**, **NP**, **NP**-completeness
- Related courses:
  - CSE332: Theory of computation. Fall 2021 (?)
  - CSE515: Advanced algorithms. Spring 2021 (?)