NP-hardness: Motivation

Erickson, Chapter 12

This course so far

- Design patterns for **efficient algorithms**:
  - Divide-and-Conquer
  - Dynamic Programming
  - Greedy Algorithms
  - Maximum Flow
  - etc.

Which problems have efficient algorithms?

Which problems **don't** have efficient algorithms?

Goal

Classify problems according to their inherent complexity

Complexity Classification

(efficient = polynomial-time computable)

efficient algorithms exist	probably no efficient algorithms
shortest path	longest path
minimum cut	maximum cut
2SAT	3SAT
planar 4-colorability	planar 3-colorability
minimum bipartite vertex-cover	minimum vertex-cover
maximum matching	maximum 3d-matching
linear programming	integer linear programming
primality testing	factoring

(Note: Factoring is not known to be NP-hard!)

Why do we care?

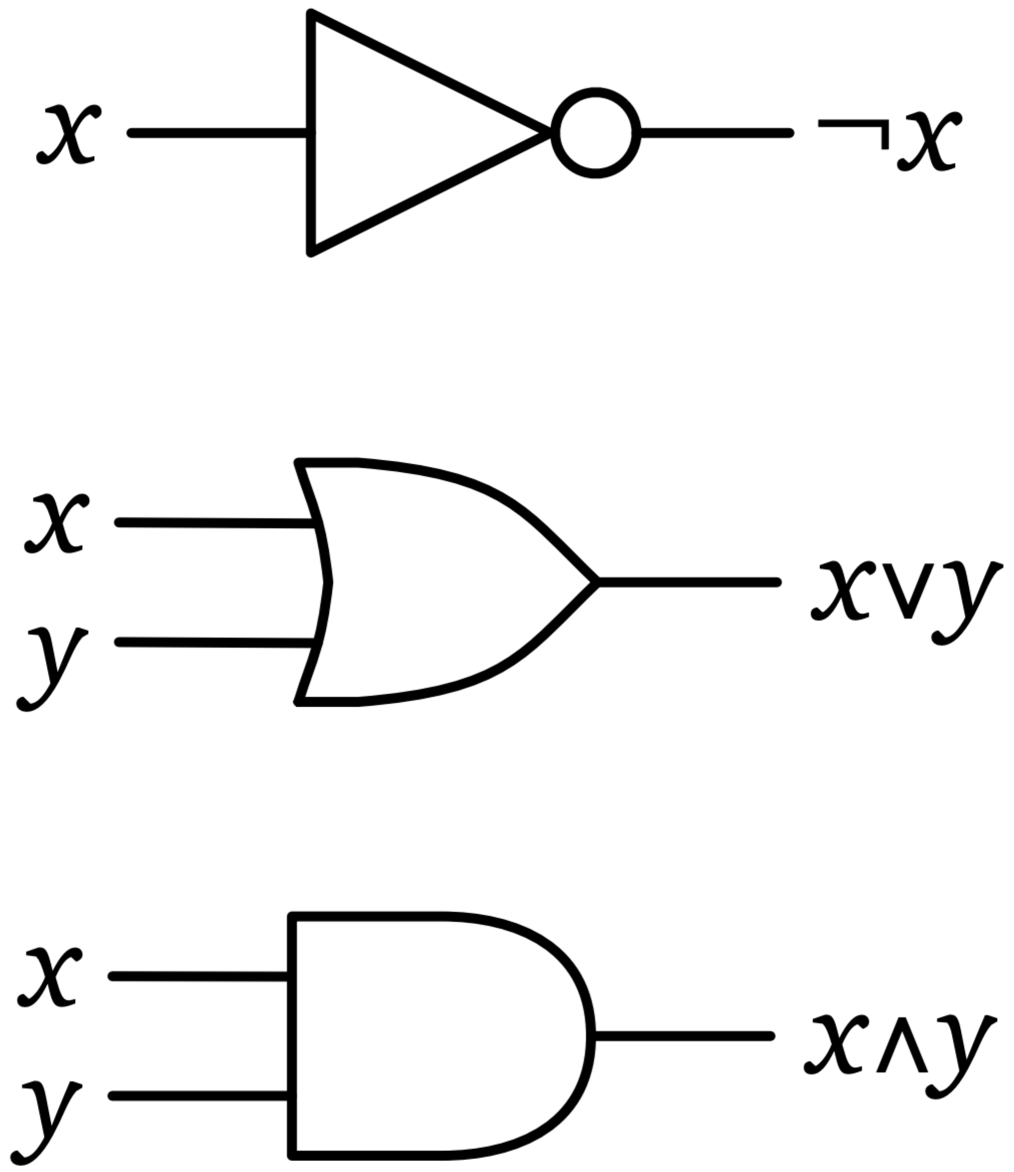
- Know when to change your goals:
  - use heuristics (e.g. SAT-solvers, ILP-solvers, etc.)
  - narrow down your problem
  - use approximation algorithms or fixed-parameter tractable algorithms
- Explain to your employer why neither you nor anyone else can find an efficient algorithm

The Circuit Satisfiability Problem

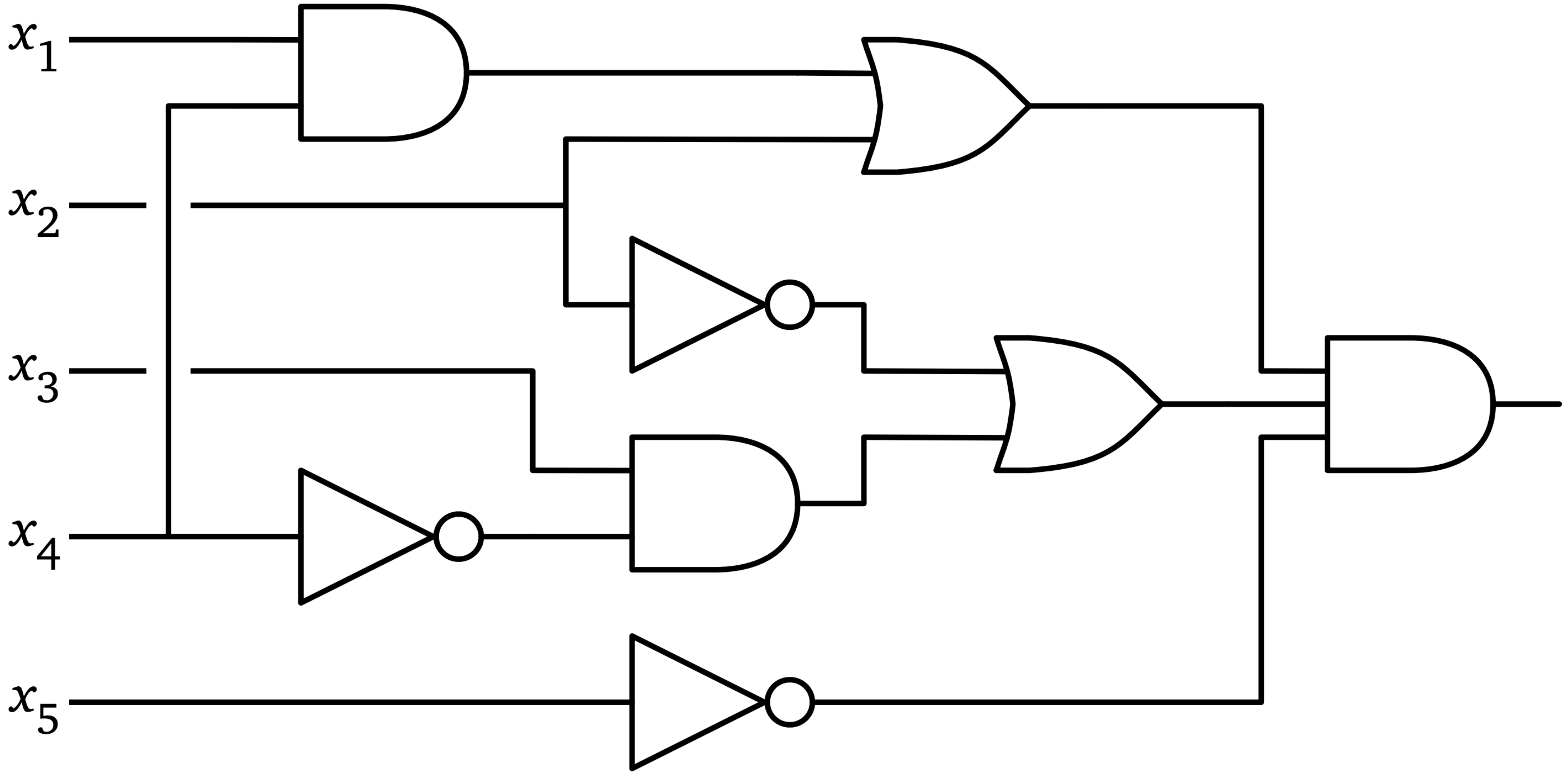
Erickson, Section 12.1

Boolean Circuits

Logical Gates

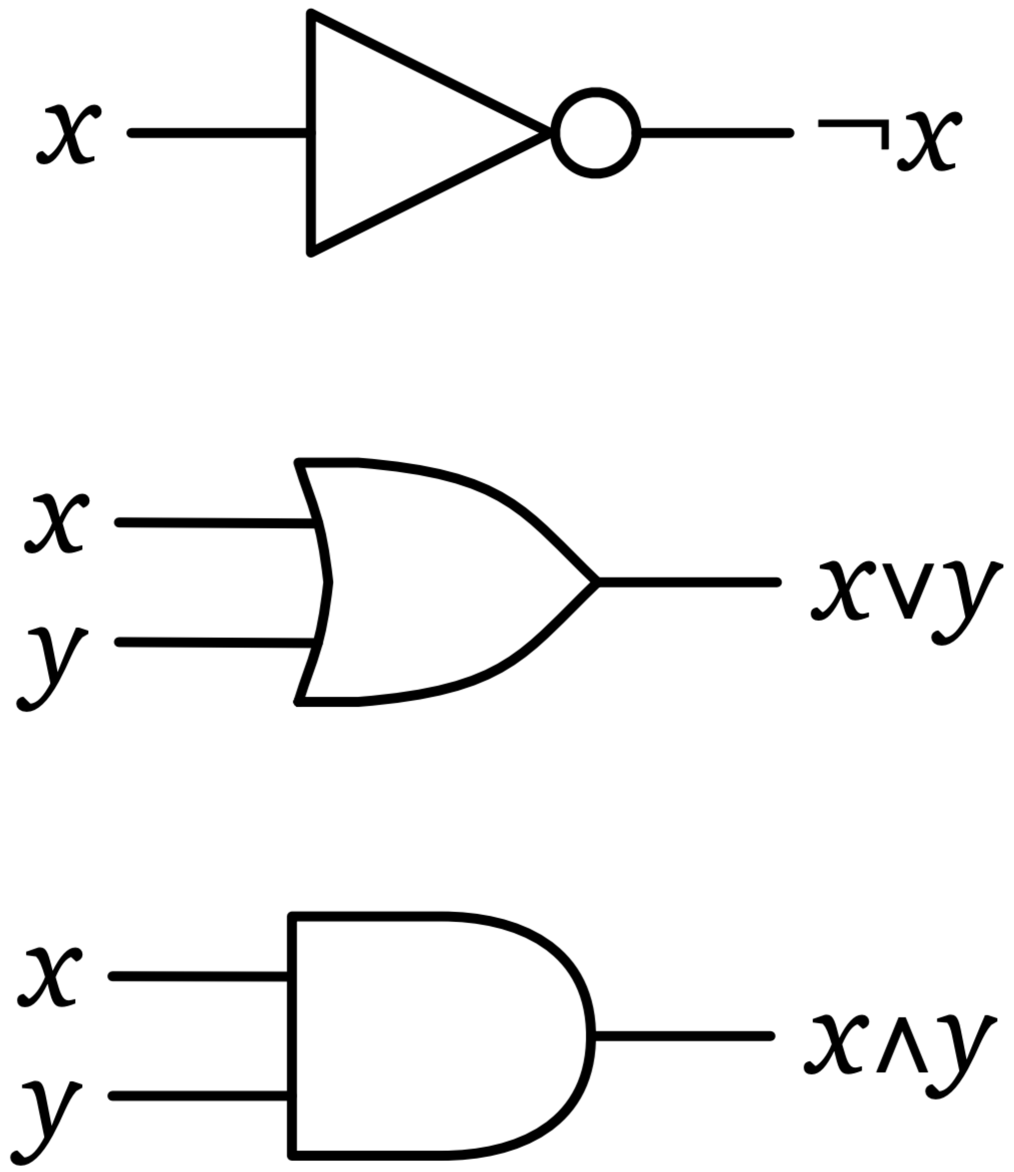


Circuit

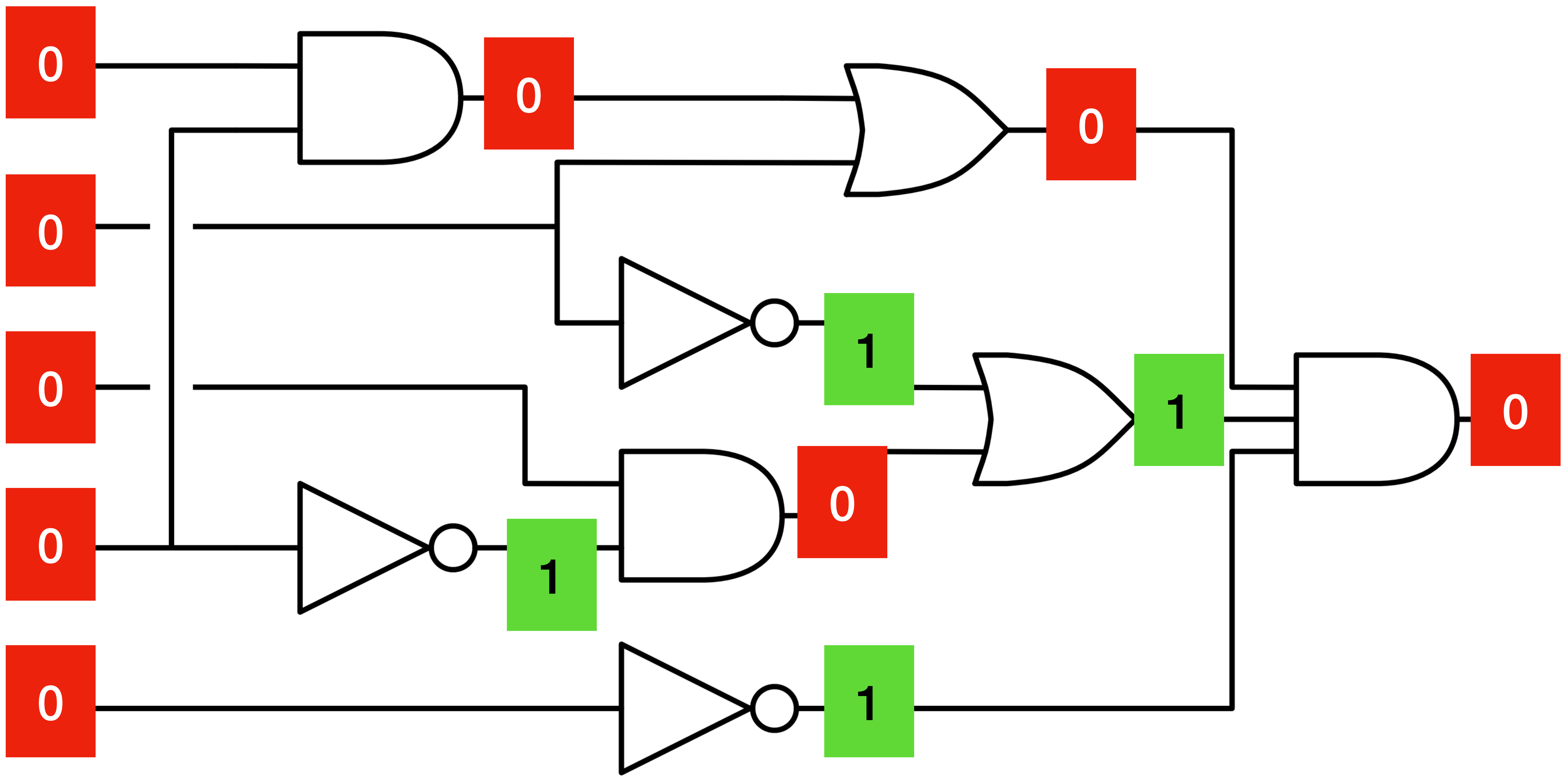


Example

Logical Gates



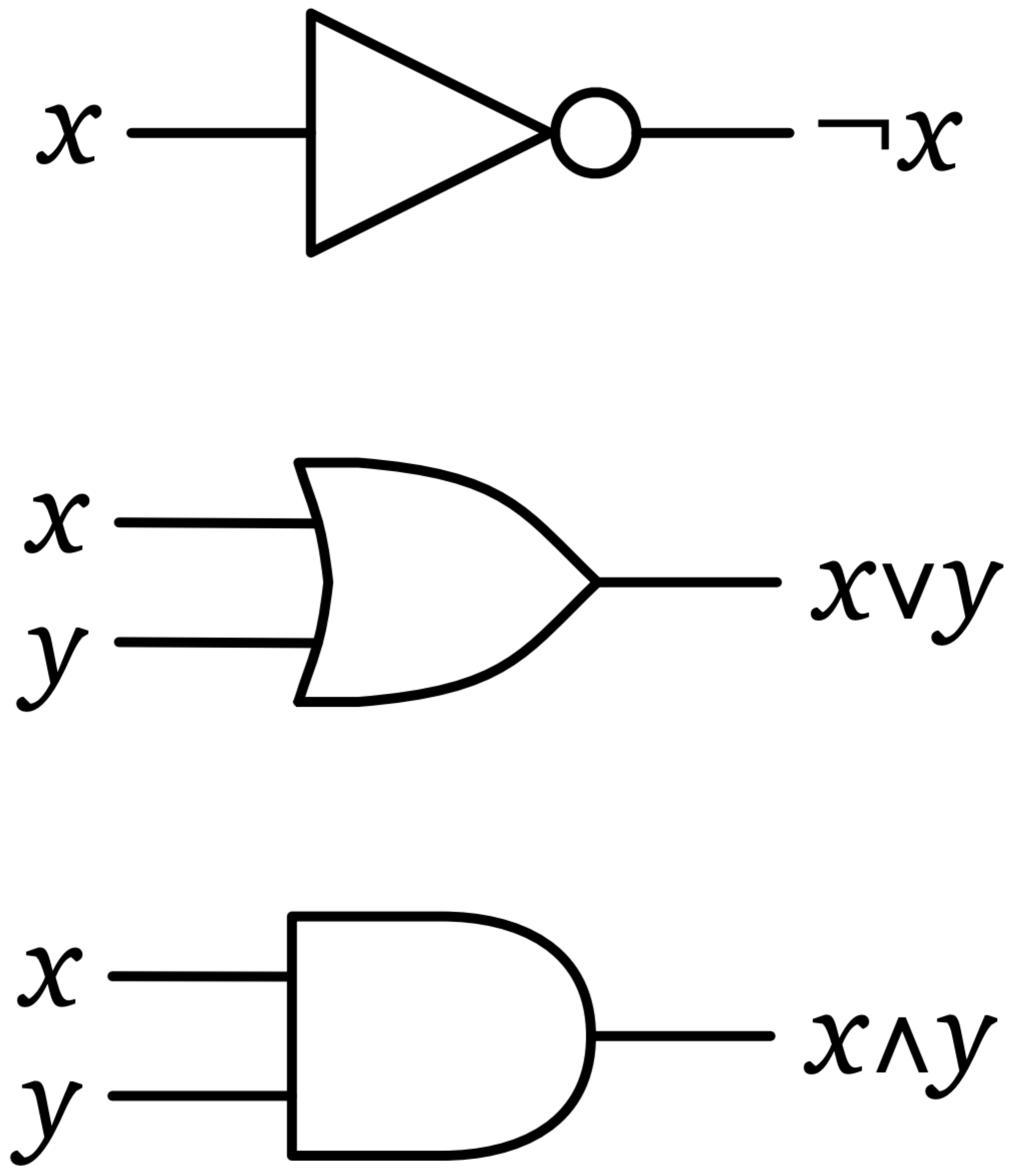
Circuit



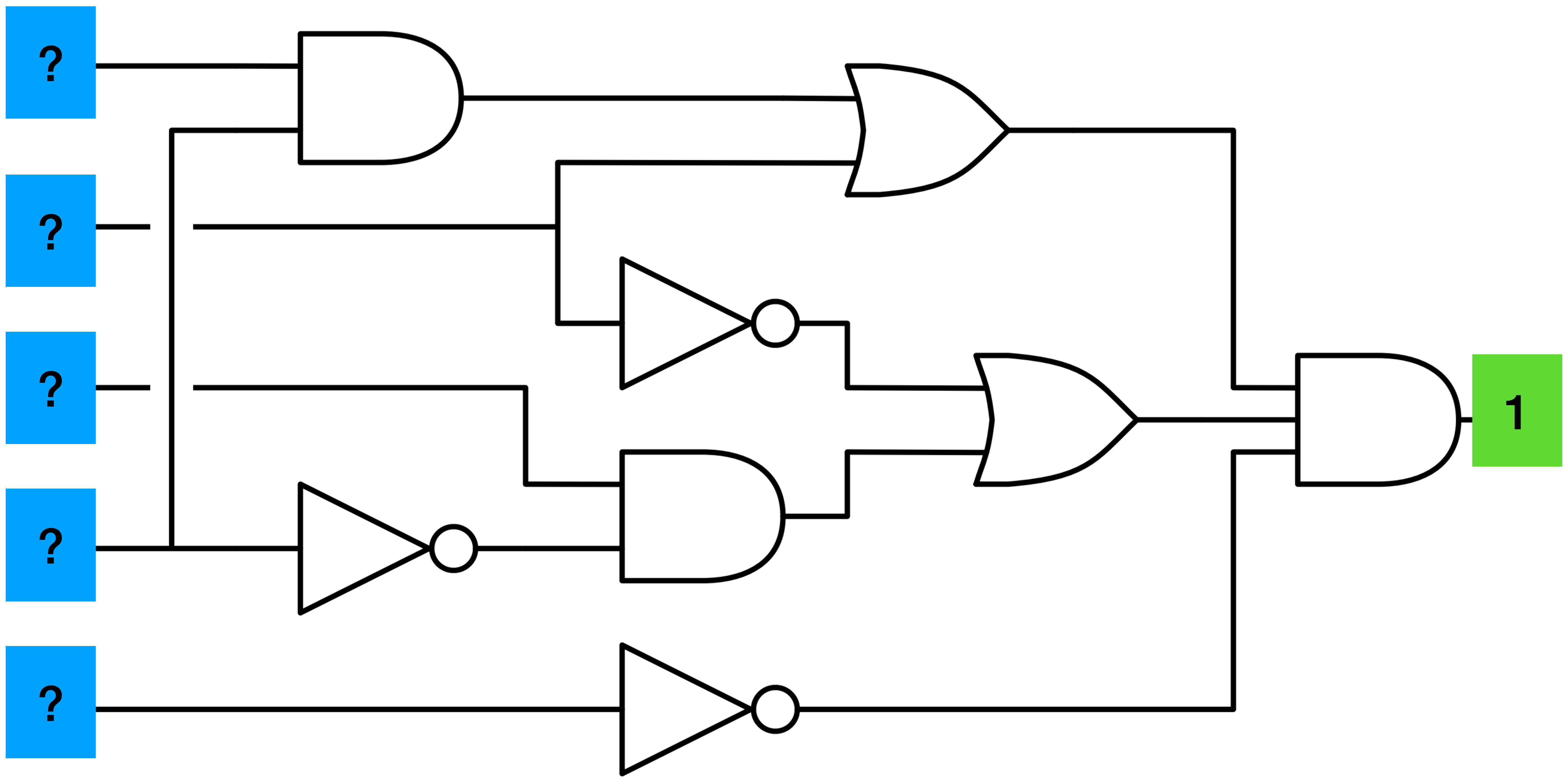
Problem 1: Circuit satisfiability

Is there an assignment so that the circuit outputs 1?

Logical Gates



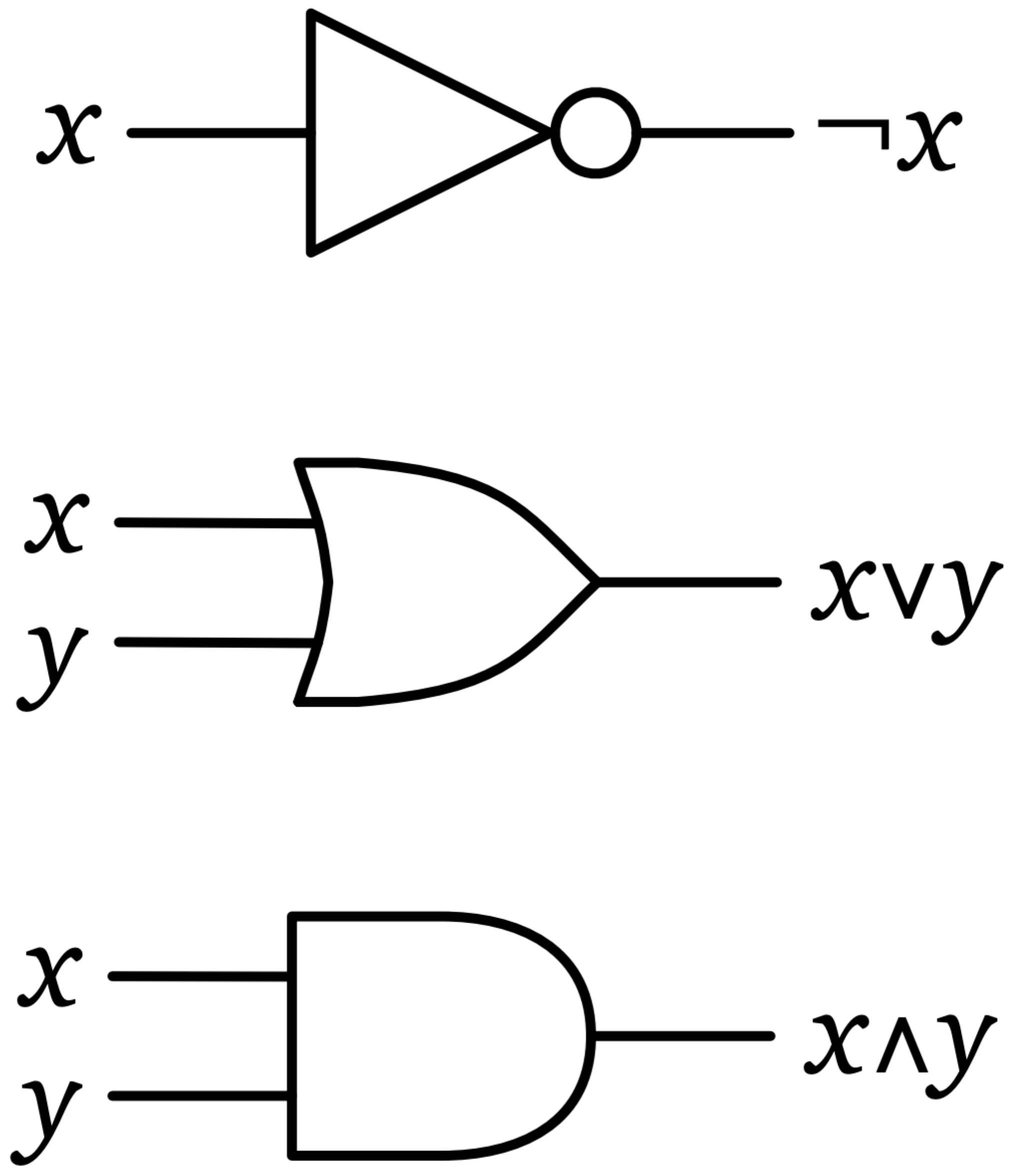
Circuit



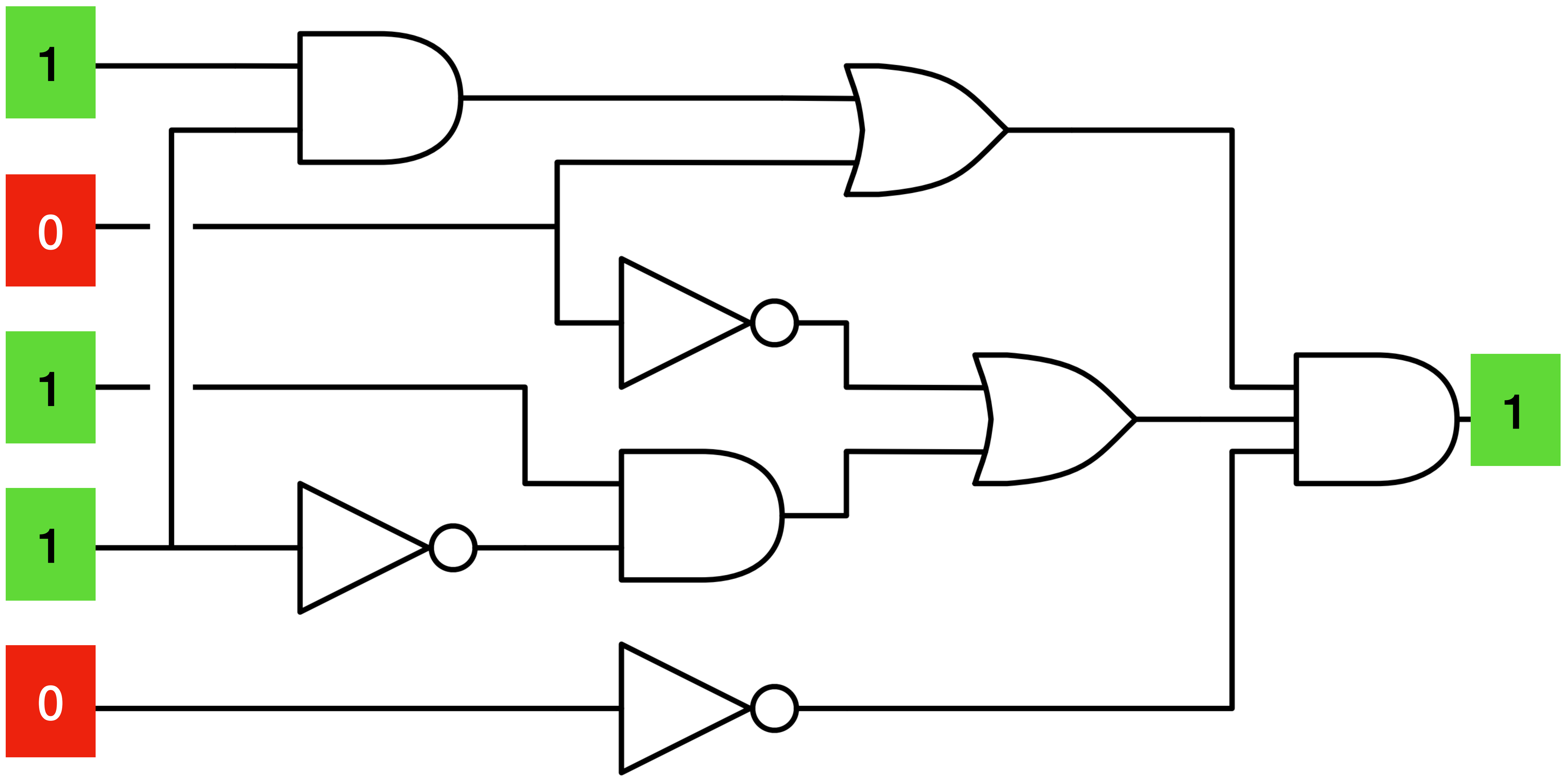
Problem 2: Verification of circuit satisfiability

Verify that the circuit outputs 1 on the given assignment.

Logical Gates



Circuit



This is a **satisfying assignment** of the circuit.

Circuit satisfiability vs verification

- **Definition.** Circuit C is **satisfiable** if it has a satisfying assignment.

- Problem 1: Given circuit C, decide whether C is satisfiable.
- Problem 2: Given circuit C and assignment x, decide whether x satisfies C.

- **Exercise:** Do you think Problem 1 is polynomial-time computable? Do you think Problem 2 polynomial-time computable? Why / Why not?

P versus NP

Erickson, Section 12.2

Decision problems

- **Definition.**
  - finite alphabet Σ, typically $\Sigma = \{0,1\}$
  - decision problem $L \subseteq \Sigma^*$ (also called "language")
- **Example.**
  - CircuitSAT = { Circuit C | C is satisfiable }

Exercise. How do you encode a circuit C as a string in Σ^*?

Algorithm for decision problem

- An algorithm A **solves** L if, for all possible input strings $x \in \Sigma*$, we have:
  - if $x \in L$, then $A(x) = 1$
  - if $x \notin L$, then $A(x) = 0$

- **Example.** An algorithm A solves CircuitSAT if, given any circuit C as input,
  - if C is satisfiable, then A(C)=1
  - if C is not satisfiable, then A(C)=0

Verifier for decision problem

- A **verifier** V for L is an algorithm that is given $x \in \Sigma^*$ as input, such that
  - if $x \in L$, then there exists some $y \in \Sigma^*$ such that $V(x, y) = 1$
  - if $x \notin L$, then, for all $y \in \Sigma^*$, we have $V(x, y) = 0$

- **Exercise.** Write the pseudocode of a polynomial-time verifier V(C, y) for CircuitSAT, that is, an algorithm V that is given a circuit C and an assignment y for C as input.

P versus NP

- **P** = $\left\{ L \subseteq \Sigma^* \mid L \text{ has a polynomial-time algorithm} \right\}$
- **NP** = $\left\{ L \subseteq \Sigma^* \mid L \text{ has a polynomial-time verifier} \right\}$

- **Exercise.** Prove that CircuitSAT is contained in **NP**
- **Open research problem.** Prove that CircuitSAT is not contained in **P**

$$P \overset{?}{\neq} NP$$

NP-hardness, NP-completeness

Erickson, Section 12.3, 12.4

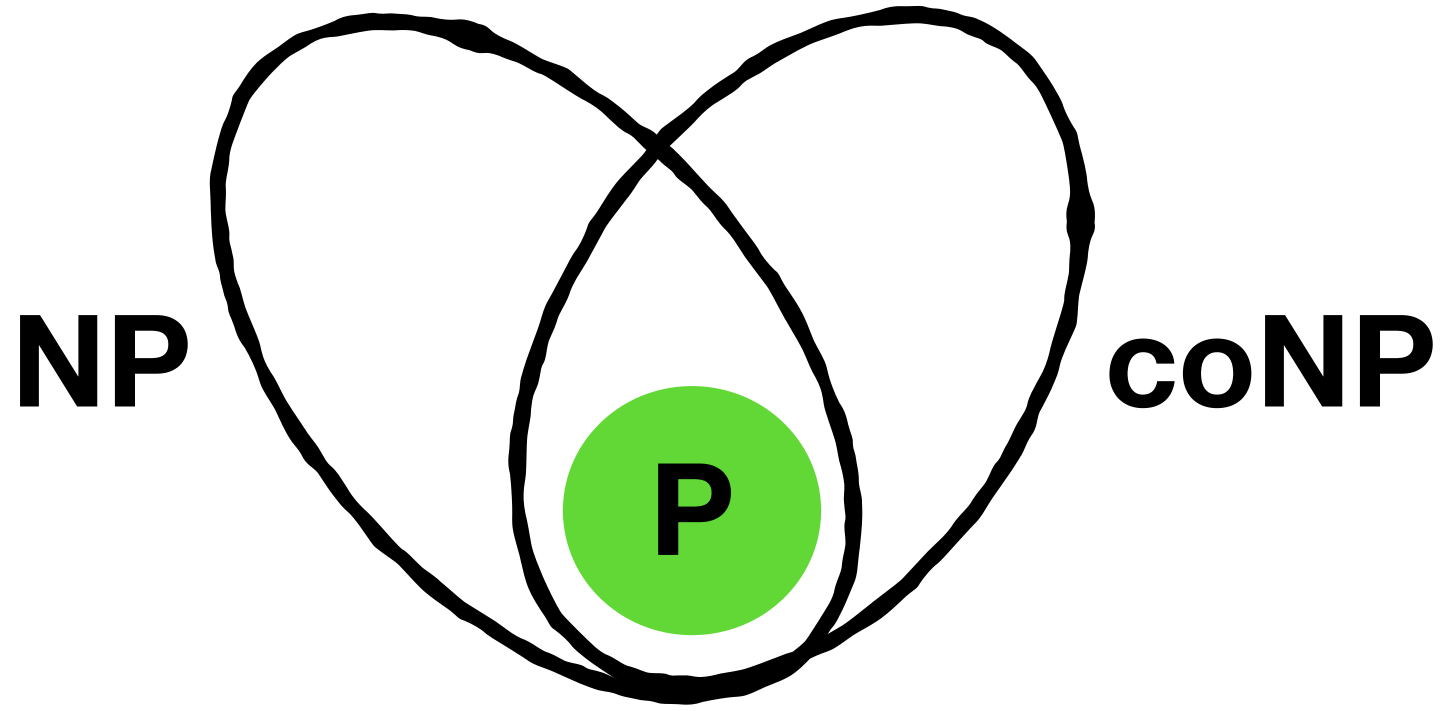
Definition of NP-hardness/NP-completeness

Erickson, Section 12.3

Let $L \subseteq \Sigma^*$ be any decision problem.

- The problem L is **NP-hard** if, for every $L' \in$ **NP**, there is a *polynomial-time reduction* from L' to L.

- The problem L is **NP-complete** if L is **NP**-hard and $L \in$ **NP**



Polynomial-time reduction from L' to L

Erickson, Section 12.4

Suppose we have a magical algorithm **A** that solves L.

Then a **polynomial-time reduction** from L' to L is an algorithm A' that

- takes an input $x' \in \Sigma^*$ for the problem L'
- transforms this input in polynomial time to an input x for the problem L
- executes the magical algorithm **A**(x)
- outputs YES or NO depending on the output of **A**(x).

Cook-Levin Theorem

CircuitSAT is NP-hard

- (We do not prove this theorem here.)

- Here is an important lemma:

If L is **NP**-hard and **P** $\neq$ **NP**, then L $\notin$ **P**.

- **Exercise.** Assuming **P** $\neq$ **NP**, what do you now know about CircuitSAT?

Reductions and SAT

Erickson, Section 12.5

Formula Satisfiability

Exercise.
Which of these formulas is satisfiable?

$\Phi_1 = (x_1 \wedge \overline{x_2}) \vee (x_3 \wedge (x_4 \vee \overline{x_1}))$

$\Phi_2 = (x_1 \wedge \overline{x_2} \wedge (\overline{x_1} \vee x_2))$

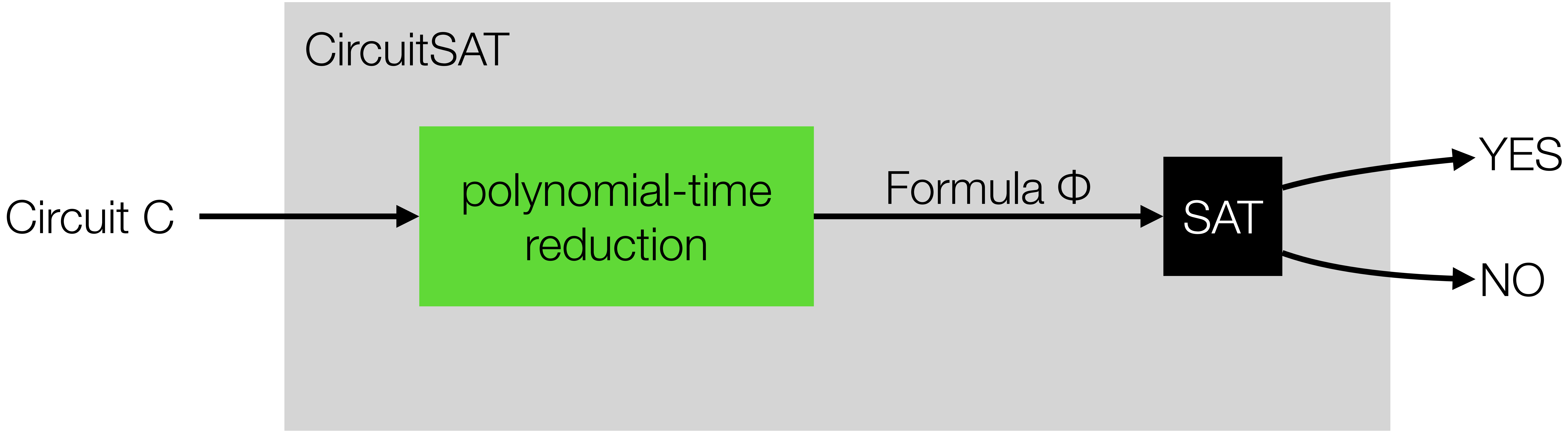
Definition. SAT is the following decision problem:

- Input: Boolean formula Φ
- Question: Is Φ satisfiable?

Is SAT **NP**-hard?

Reductions

- To show that SAT is **NP**-hard,
  we construct a **polynomial-time reduction** from CircuitSAT to SAT:



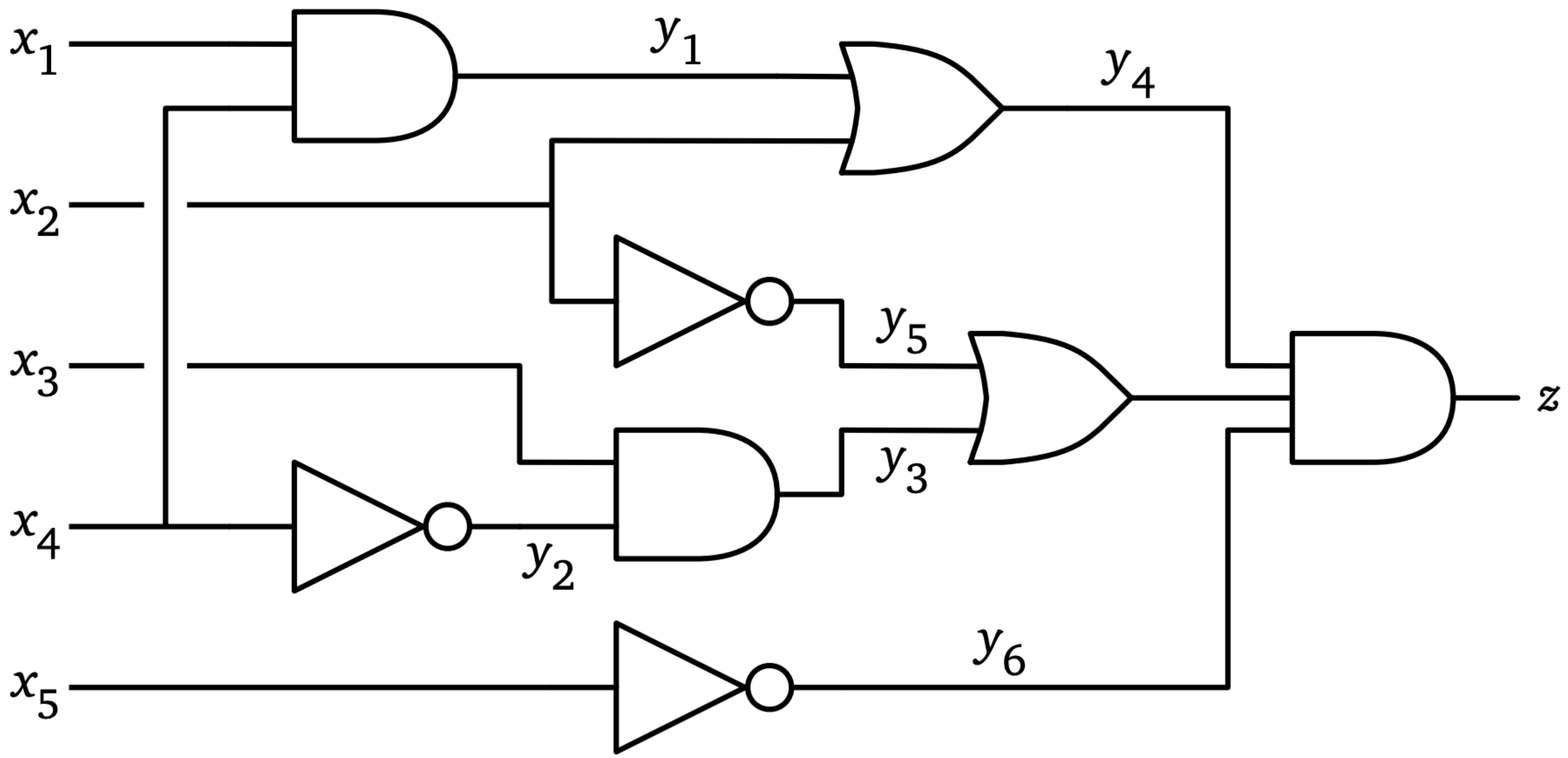
- Any potential algorithm for SAT yields an algorithm for CircuitSAT

Reduction from CircuitSAT to SAT

- **Goal.** Given a circuit C, compute a formula Φ such that C is satisfiable if and only if Φ is satisfiable.
- **Idea.** Introduce a new variable for each wire, then replace each gate with a formula that verifies the computation of the gate.

Reduction from CircuitSAT to SAT

Example



$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge$$

$$(y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

3SAT is NP-hard

Erickson, Section 12.6

3CNF formulas

Erickson, Section 12.6

- Conjunctive normal form (CNF):

$$(a \vee b \vee c \vee d) \wedge \underbrace{(b \vee \overline{c} \vee \overline{d})}_{\text{clause}} \wedge (\overline{a} \vee c \vee d) \wedge (a \vee \overline{b})$$

- 3CNF formulas: Every clause has width 3.

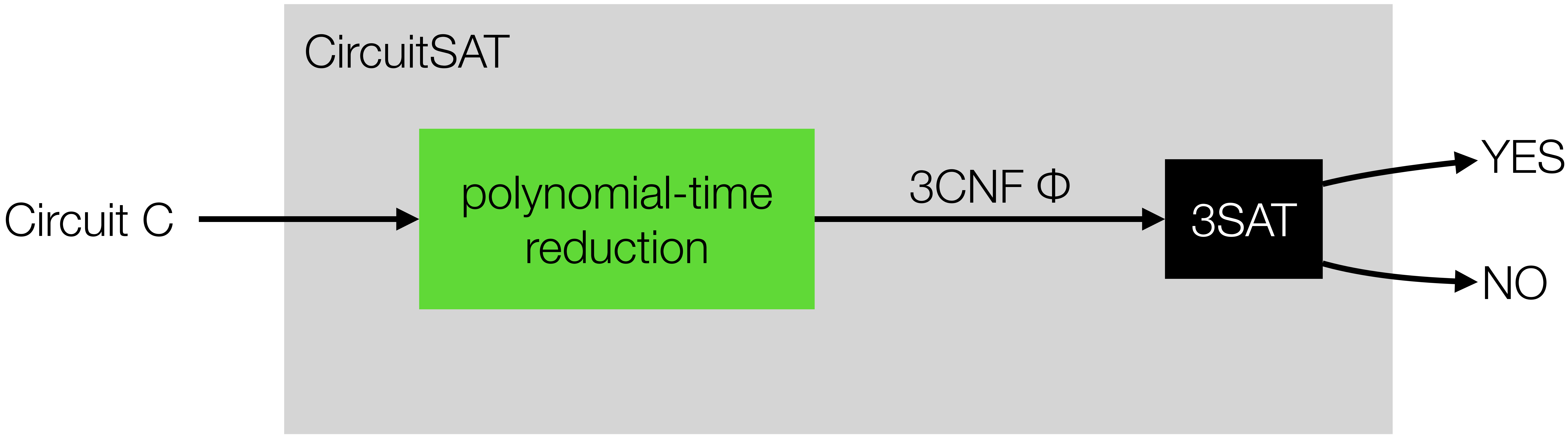
3SAT

- 3SAT is the decision problem:
  - **Input.** 3CNF formula Φ
  - **Question.** Is Φ satisfiable?

- We want to show that 3SAT is NP-hard.

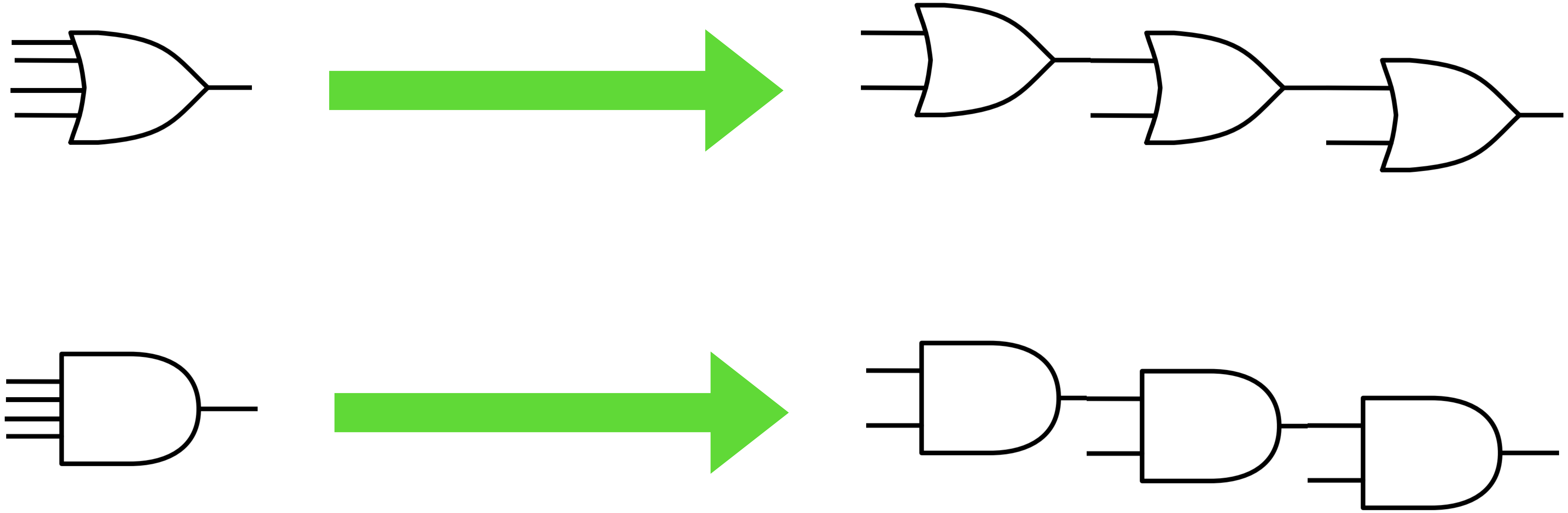
Reduction from CircuitSAT to 3SAT

Overview



Reduction from CircuitSAT to 3SAT

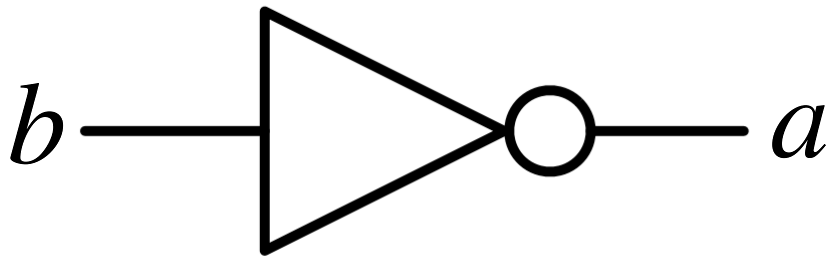
Step 1: Reduce fan-in



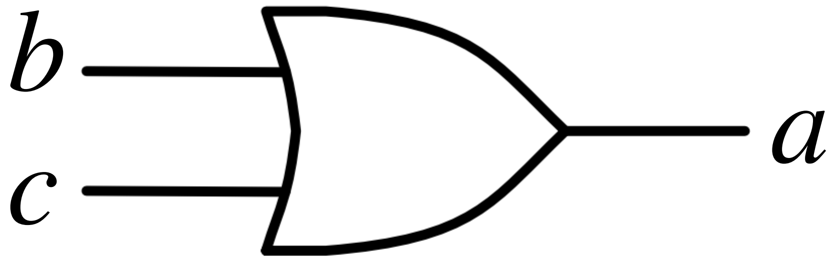
- After this operation, all gates have at most 2 wires feeding into them.

Reduction from CircuitSAT to 3SAT

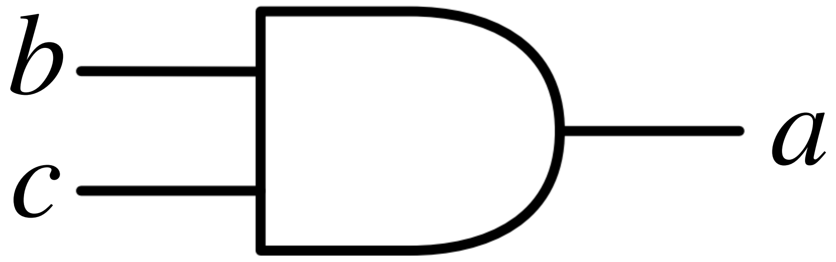
Step 2: Transform gates to formulas to clauses



$a = \overline{b}$ → $(a \vee b) \wedge (\overline{a} \vee \overline{b})$



$a = b \vee c$ → $(\overline{a} \vee b \vee c) \wedge (a \vee \overline{b}) \wedge (a \vee \overline{c})$



$a = b \wedge c$ → $(a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b) \wedge (\overline{a} \vee c)$

- Additionally, add the clause (z), indicating that the output wire of C is set to 1.

Reduction from CircuitSAT to 3SAT

Step 3: Fill up clauses of smaller width to obtain a 3CNF

$(a \vee b) \rightarrow (a \vee b \vee z) \wedge (a \vee b \vee \bar{z})$

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

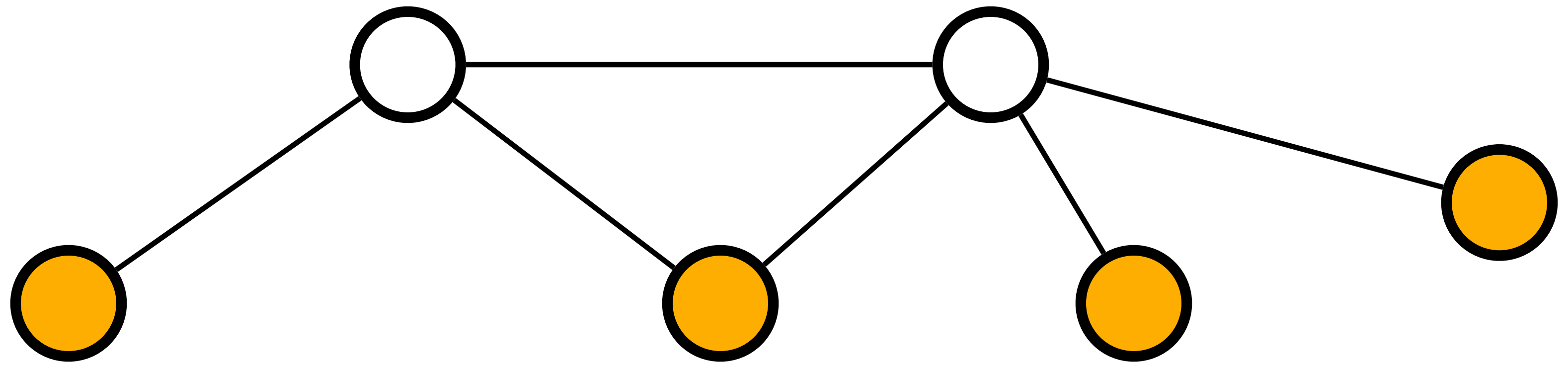
- After this operation, all clauses have exactly 3 literals.
- The entire reduction takes only polynomial time.
- **Result.** 3SAT is NP-hard.

Maximum Independent Set is NP-hard

Erickson, Section 12.7

Maximum Independent Set Problem

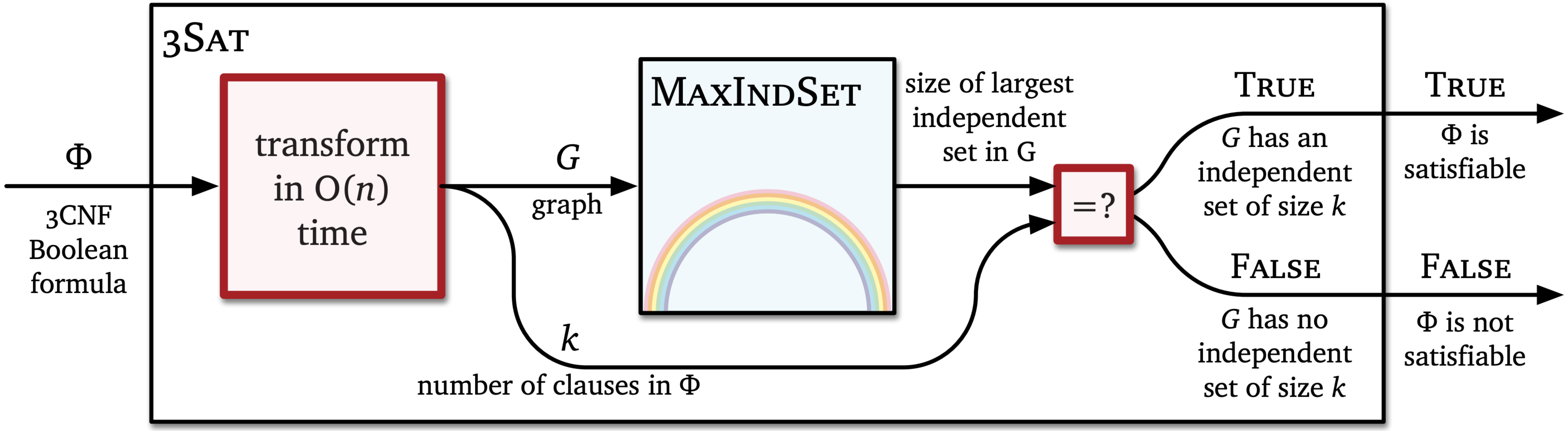
- **Independent Set.** Set S ⊆ V(G) such that no two vertices in S are adjacent



- **Maximum Independent Set Problem.**
  - **Input.** Graph G
  - **Output.** Size |S| of a maximum independent set S.

Reduction from 3SAT to MaxIndSet

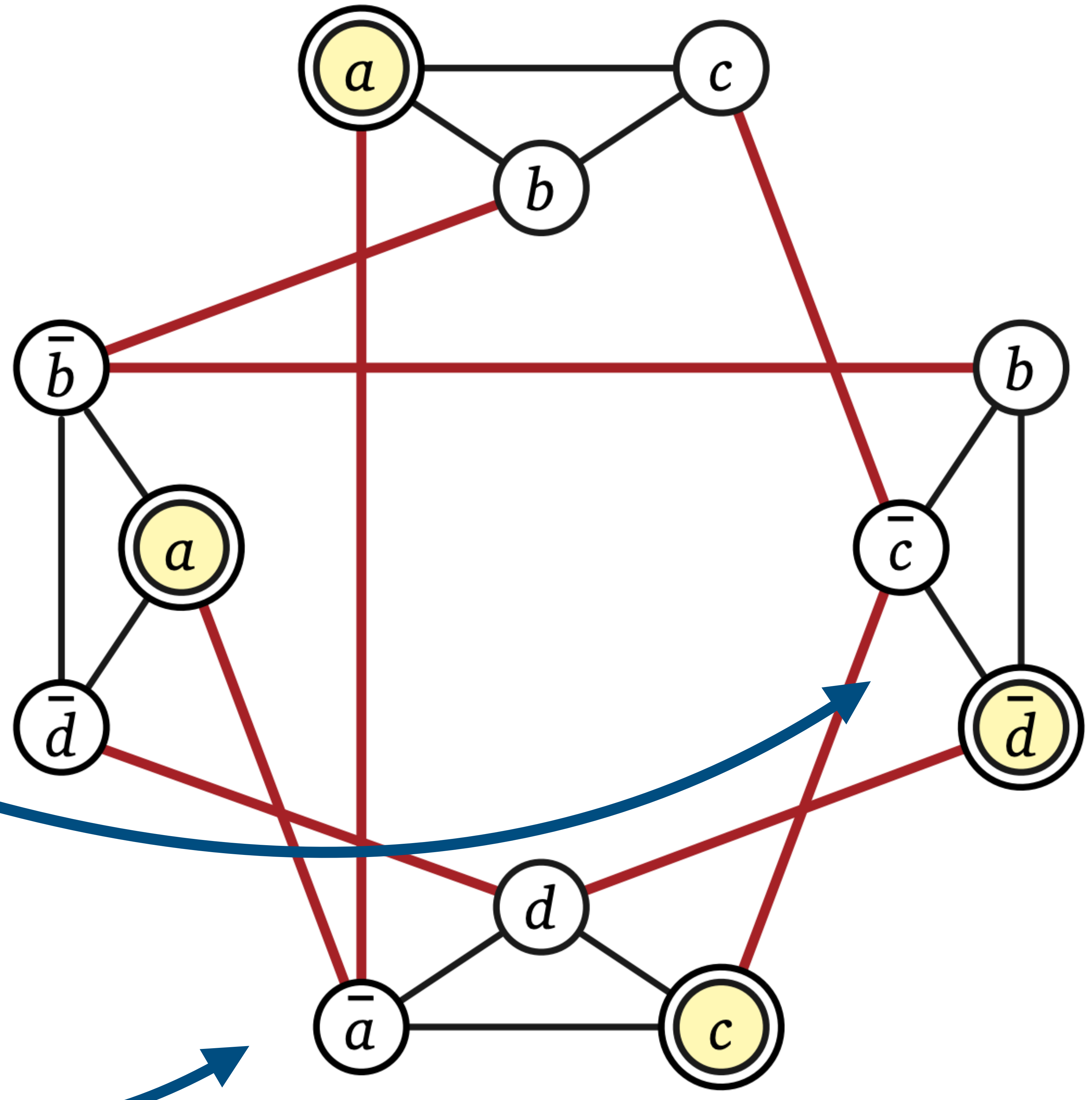
Overview



Reduction from 3SAT to MaxIndSet

Example

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



Given Φ, construct G as follows:

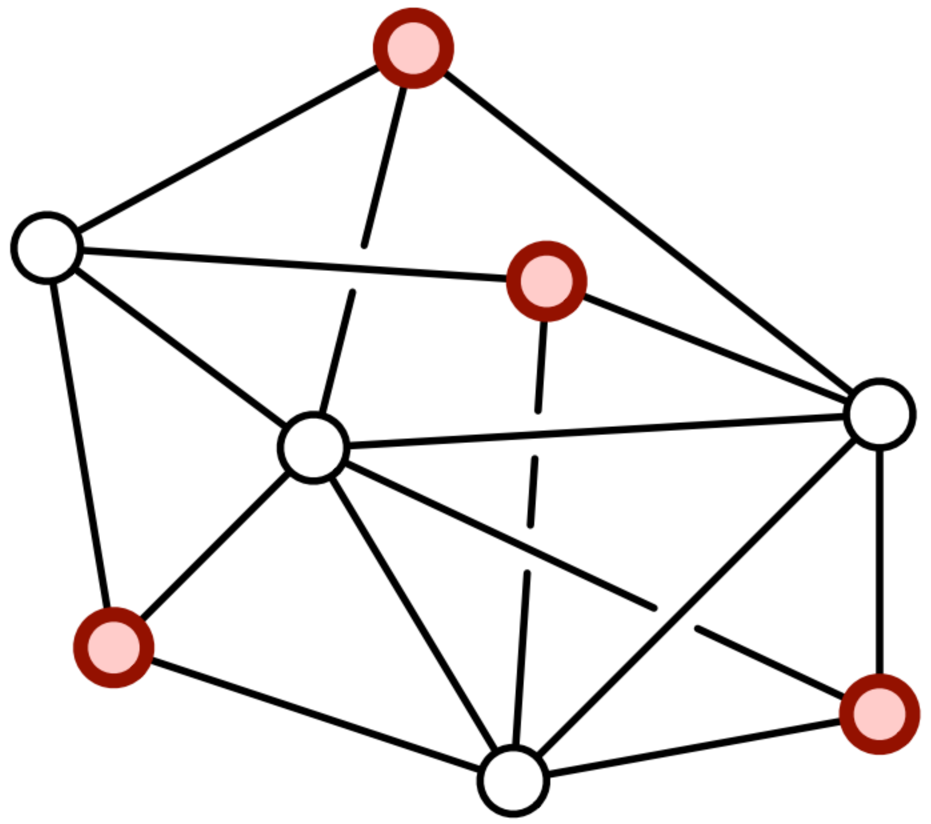
- create a triangle for each clause
- make inconsistent literals adjacent

Exercise. Why is this reduction correct?
Read the proof. [Erickson, Section 12.7]

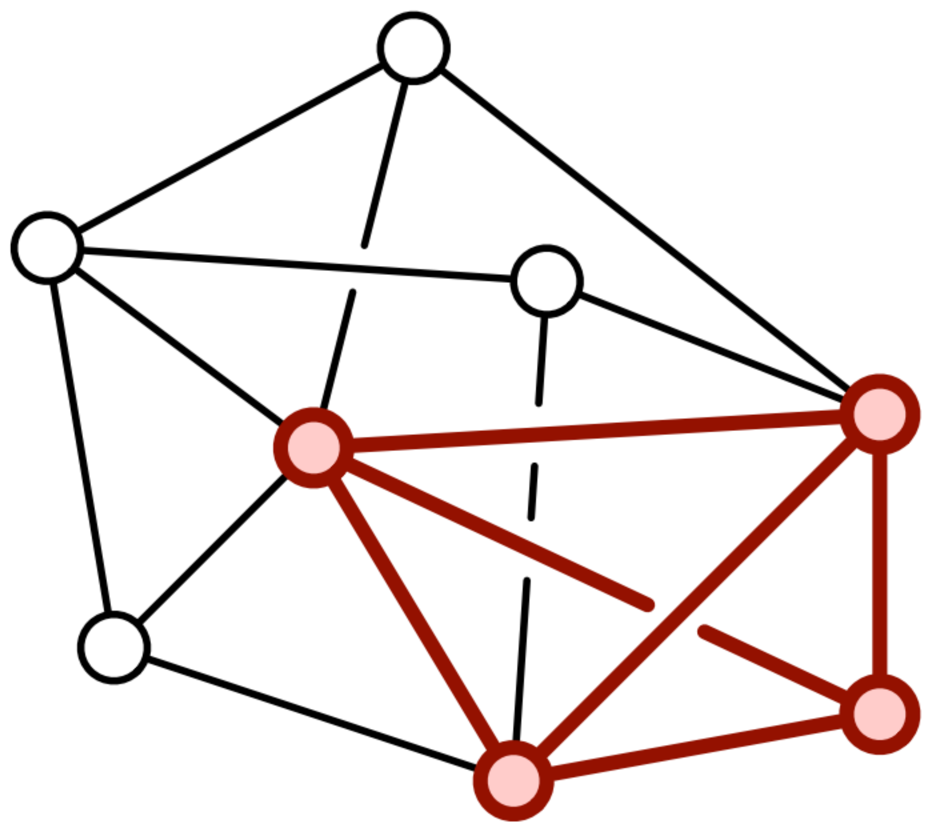
Maximum Clique is NP-hard
Minimum Vertex-Cover is NP-hard

Erickson, Section 12.9

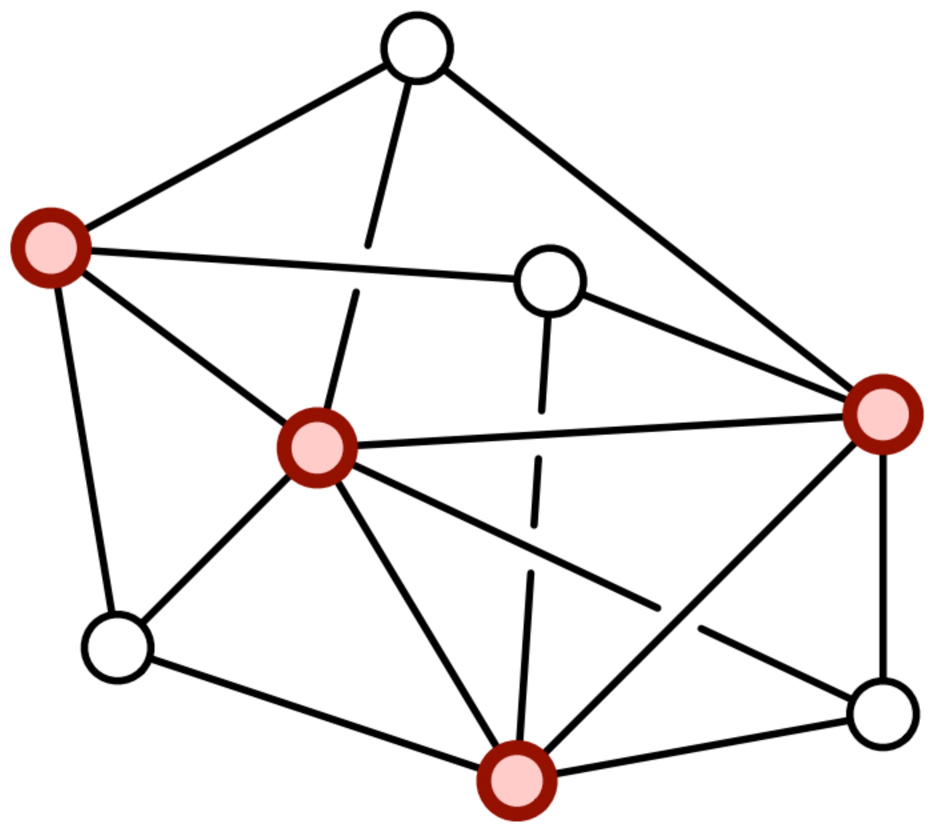
Three related problems



Maximum
Independent Set

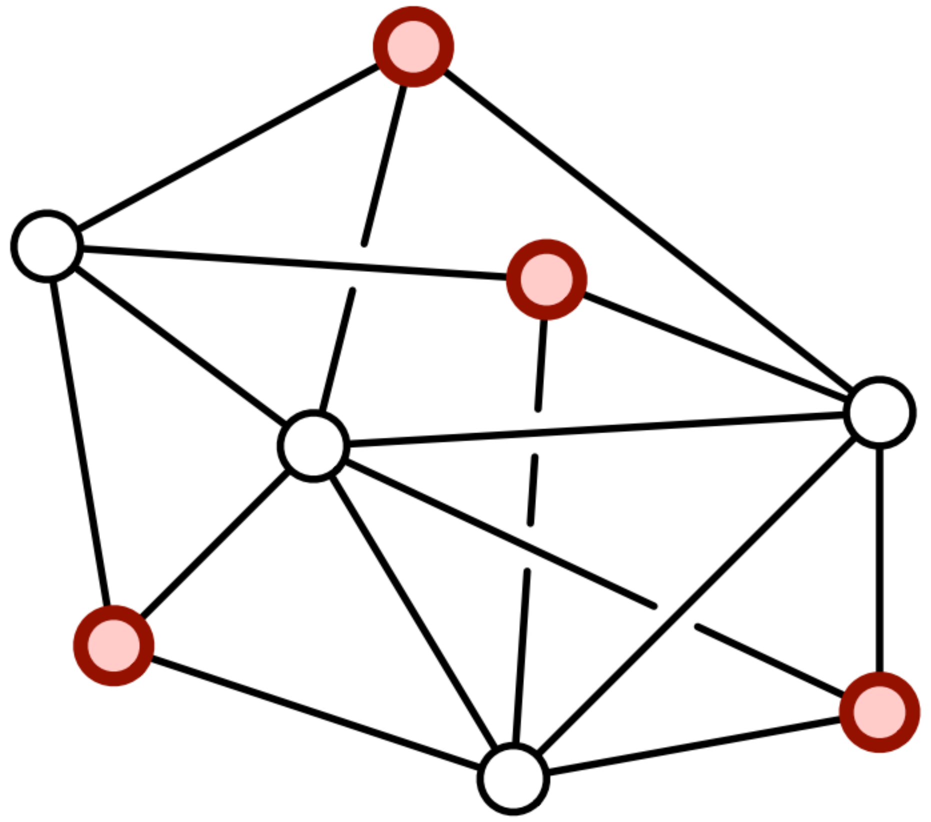


Maximum Clique

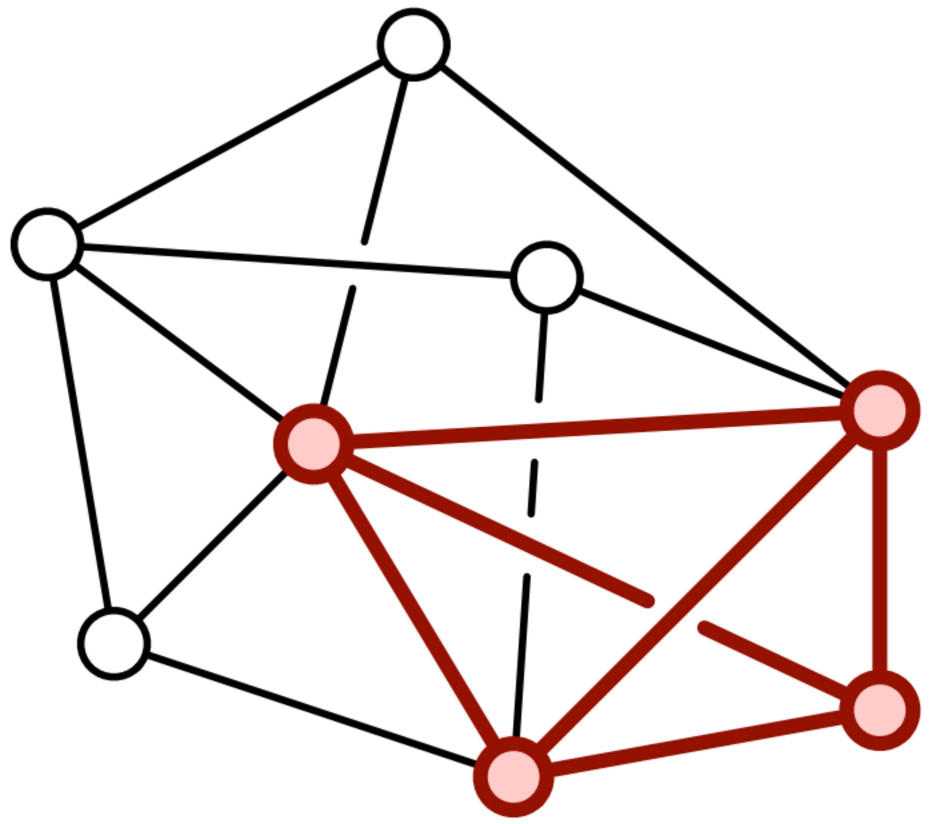


Minimum
Vertex-Cover

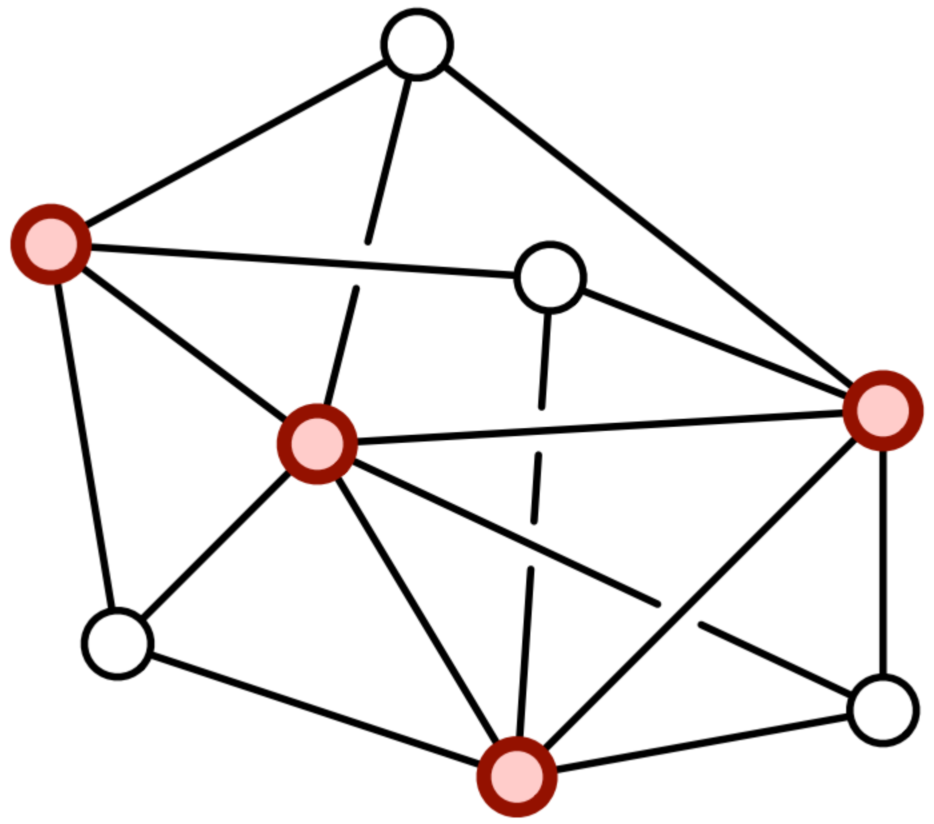
Three related problems



Maximum
Independent Set



Maximum Clique



Minimum
Vertex-Cover

Exercise. How exactly are
these two concepts related?

Independent Set vs Clique

graph G

complement graph $\overline{G}$

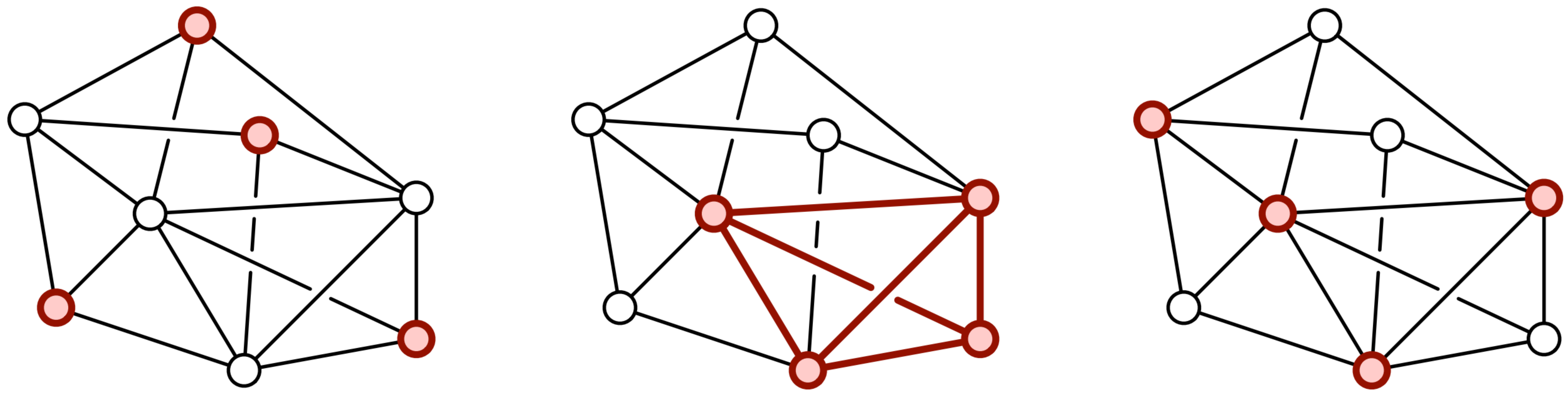
S is an independent sets of G

$\iff$ S contains no edges from G

$\iff$ S contains no non-edges from $\overline{G}$

$\iff$ S is a clique of $\overline{G}$

Three related problems



Maximum Independent Set

Maximum Clique

Minimum Vertex-Cover

Exercise. How exactly are these two concepts related?

Independent Set vs Vertex-Cover

S is an **independent sets** of G

$\Longleftrightarrow$ S contains no edges from G

$\Longleftrightarrow$ all edges of G intersect with V – S

$\Longleftrightarrow$ V – S is a **vertex-cover** of G

Polynomial-time Reductions