

Lecture 24: NP-Completeness II

Michael Dinitz

November 20, 2025
601.433/633 Introduction to Algorithms

Introduction

Last time: Definition of P , NP , reductions, NP -completeness. Proof that Circuit-SAT is NP -complete.

Today: more NP-complete problems.

Definition

A decision problem Q is in NP (*nondeterministic polynomial time*) if there exists a polynomial time algorithm $V(I, X)$ (called the *verifier*) such that

1. If I is a YES-instance of Q , then there is some X (usually called the *witness*, *proof*, or *solution*) with size polynomial in $|I|$ so that $V(I, X) = \text{YES}$.
2. If I is a NO-instance of Q , then $V(I, X) = \text{NO}$ for all X .

Reductions

(P) \leq_p Q
NP-hard

Definition

A *Many-one* or *Karp* reduction from \mathbf{A} to \mathbf{B} is a function f which takes arbitrary instances of \mathbf{A} and transforms them into instances of \mathbf{B} so that

1. If x is a YES-instance of \mathbf{A} then $f(x)$ is a YES-instance of \mathbf{B} .
2. If x is a NO-instance of \mathbf{A} then $f(x)$ is a NO-instance \mathbf{B} .
3. f can be computed in polynomial time.

Definition

Problem Q is **NP-hard** if $Q' \leq_p Q$ for all problems Q' in **NP**. Problem Q is **NP-complete** if it is **NP-hard** and in **NP**.

Circuit-SAT

Definition

Circuit-SAT: Given a boolean circuit of AND, OR, and NOT gates, with a single output and no loops (some inputs might be hardwired), is there a way of setting the inputs so that the output of the circuit is 1?

Theorem

Circuit-SAT is NP-complete.

3-SAT

Boolean formula:

- ▶ Boolean variables x_1, \dots, x_n
- ▶ Literal: variable x_i or negation \bar{x}_i
- ▶ AND: \wedge OR: \vee
- ▶ $x_1 \vee (\bar{x}_5 \wedge x_7) \wedge (\bar{x}_2 \vee (x_6 \wedge \bar{x}_3)) \dots$

Conjunctive normal form (CNF): AND of ORs (clauses)

- ▶ $(x_1 \vee \bar{x}_2 \vee \bar{x}_4) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee \bar{x}_6) \dots$

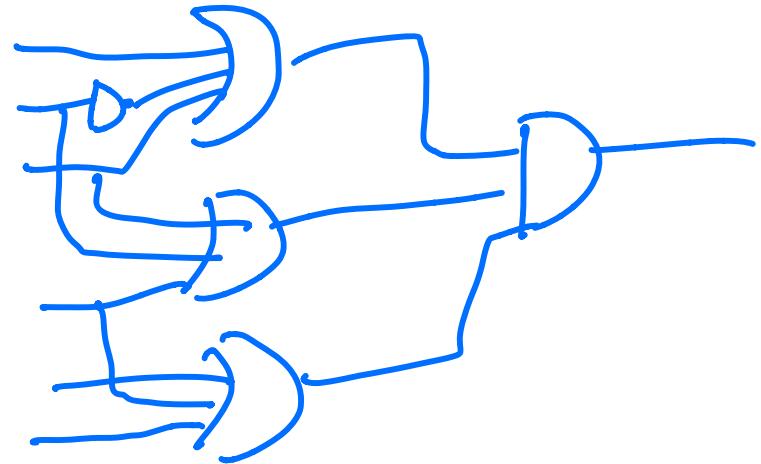
3-SAT

Boolean formula:

- ▶ Boolean variables x_1, \dots, x_n
- ▶ Literal: variable x_i or negation \bar{x}_i
- ▶ AND: \wedge OR: \vee
- ▶ $x_1 \vee (\bar{x}_5 \wedge x_7) \wedge (\bar{x}_2 \vee (x_6 \wedge \bar{x}_3)) \dots$

Conjunctive normal form (CNF): AND of ORs (clauses)

- ▶ $(x_1 \vee \bar{x}_2 \vee \bar{x}_4) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee \bar{x}_6) \dots$



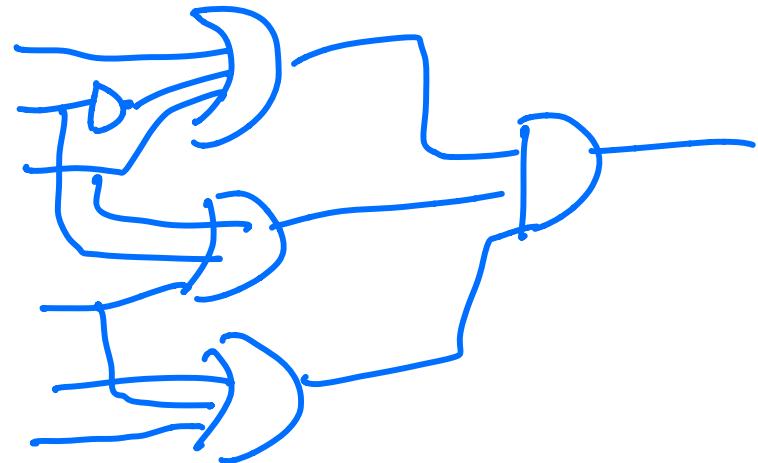
3-SAT

Boolean formula:

- ▶ Boolean variables x_1, \dots, x_n
- ▶ Literal: variable x_i or negation \bar{x}_i
- ▶ AND: \wedge OR: \vee
- ▶ $x_1 \vee (\bar{x}_5 \wedge x_7) \wedge (\bar{x}_2 \vee (x_6 \wedge \bar{x}_3)) \dots$

Conjunctive normal form (CNF): AND of ORs (clauses)

- ▶ $(x_1 \vee \bar{x}_2 \vee \bar{x}_4) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee \bar{x}_6) \dots$



Definition

3-SAT: Instance is 3CNF formula ϕ (every clause has ≤ 3 literals). YES if there is assignment where ϕ evaluates to True (satisfying assignment), NO otherwise.

3-SAT

Theorem

3-SAT is NP-complete.

3-SAT

Theorem

3-SAT is **NP**-complete.

3-SAT in **NP**:

3-SAT

Theorem

*3-SAT is **NP**-complete.*

3-SAT in **NP**: witness is assignment, verifier checks that formula evaluates to True on assignment.

3-SAT

Theorem

*3-SAT is **NP**-complete.*

3-SAT in **NP**: witness is assignment, verifier checks that formula evaluates to True on assignment.

3-SAT is **NP**-hard:

3-SAT

Theorem

3-SAT is **NP**-complete.

3-SAT in **NP**: witness is assignment, verifier checks that formula evaluates to True on assignment.

3-SAT is **NP**-hard: Show Circuit-SAT \leq_p 3-SAT.



3-SAT

Theorem

3-SAT is **NP**-complete.

3-SAT in **NP**: witness is assignment, verifier checks that formula evaluates to True on assignment.

3-SAT is **NP**-hard: Show Circuit-SAT \leq_p 3-SAT.

- ▶ Don't need to show that $A \leq_p$ 3-SAT for arbitrary $A \in NP$: already know that $A \leq_p$ Circuit-SAT!

3-SAT

Theorem

3-SAT is **NP**-complete.

3-SAT in **NP**: witness is assignment, verifier checks that formula evaluates to True on assignment.

3-SAT is **NP**-hard: Show Circuit-SAT \leq_p 3-SAT.

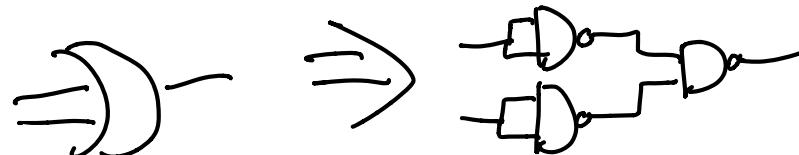
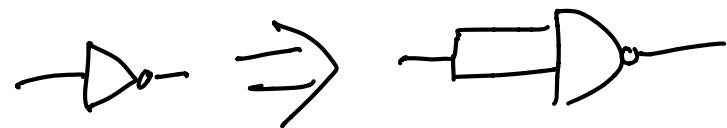
- ▶ Don't need to show that $A \leq_p$ 3-SAT for arbitrary $A \in NP$: already know that $A \leq_p$ Circuit-SAT!

So start with circuit. Want to transform to 3-CNF formula.

Transformation to NANDs

For simplicity, transform into a circuit with one type of gate: NAND (NOT AND)

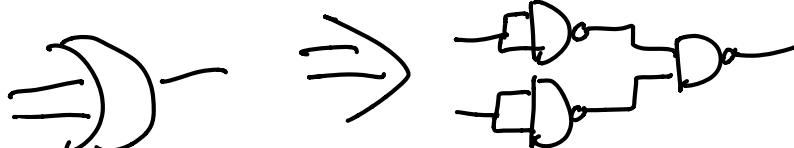
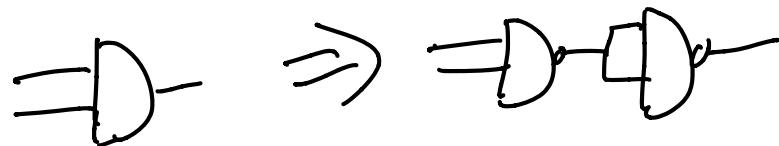
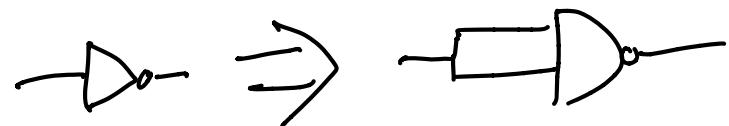
- ▶ AND/OR/NOT universal, but so is just NAND!



Transformation to NANDs

For simplicity, transform into a circuit with one type of gate: NAND (NOT AND)

- ▶ AND/OR/NOT universal, but so is just NAND!



So given circuit C , first transform it into NAND-only circuit.

Input:

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m
 - ▶ $g_1 = \text{NAND}(x_1, x_3),$
 $g_2 = \text{NAND}(g_1, x_4), \dots$
- ▶ WLOG, g_m is the “output gate”

Reduction to 3-SAT

So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Reduction to 3-SAT

So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Variables: y_1, y_2, \dots, y_n , $y_{n+1}, y_{n+2}, \dots, y_{n+m}$ (one for each wire)

Reduction to 3-SAT

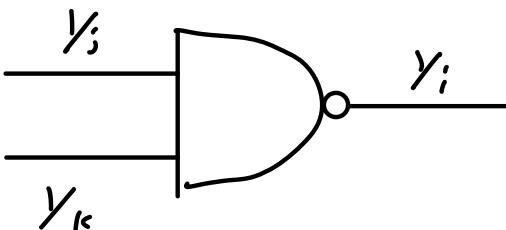
So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Variables: $y_1, y_2, \dots, y_n, y_{n+1}, y_{n+2}, \dots, y_{n+m}$ (one for each wire)

Clauses: For every NAND gate $y_i = \text{NAND}(y_j, y_k)$, create clauses:



Reduction to 3-SAT

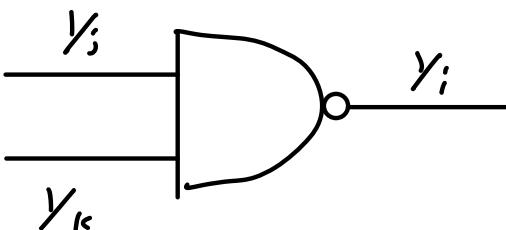
So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Variables: $y_1, y_2, \dots, y_n, y_{n+1}, y_{n+2}, \dots, y_{n+m}$ (one for each wire)

Clauses: For every NAND gate $y_i = \text{NAND}(y_j, y_k)$, create clauses:



- ▶ $y_i \vee y_j \vee y_k$

Reduction to 3-SAT

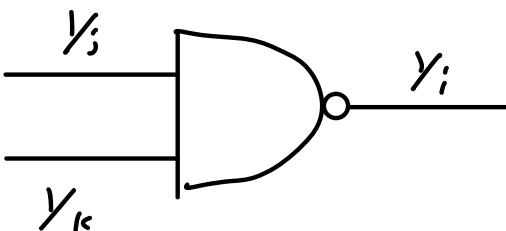
So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Variables: $y_1, y_2, \dots, y_n, y_{n+1}, y_{n+2}, \dots, y_{n+m}$ (one for each wire)

Clauses: For every NAND gate $y_i = \text{NAND}(y_j, y_k)$, create clauses:



- ▶ $y_i \vee y_j \vee y_k$ (if $y_j = 0$ and $y_k = 0$ then $y_i = 1$)

Reduction to 3-SAT

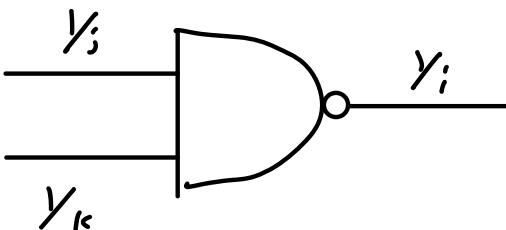
So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Variables: $y_1, y_2, \dots, y_n, y_{n+1}, y_{n+2}, \dots, y_{n+m}$ (one for each wire)

Clauses: For every NAND gate $y_i = \text{NAND}(y_j, y_k)$, create clauses:



- ▶ $y_i \vee y_j \vee y_k$ (if $y_j = 0$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee \bar{y}_j \vee y_k$

Reduction to 3-SAT

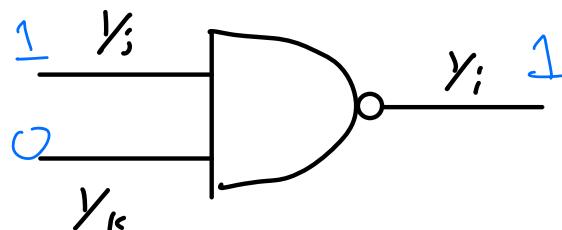
So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Variables: $y_1, y_2, \dots, y_n, y_{n+1}, y_{n+2}, \dots, y_{n+m}$ (one for each wire)

Clauses: For every NAND gate $y_i = \text{NAND}(y_j, y_k)$, create clauses:



- ▶ $y_i \vee y_j \vee y_k$ (if $y_j = 0$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee \bar{y}_j \vee y_k$ (if $y_j = 1$ and $y_k = 0$ then $y_i = 1$)

Reduction to 3-SAT

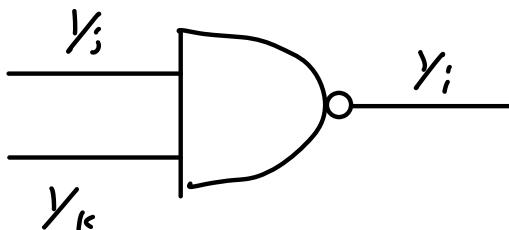
So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Variables: $y_1, y_2, \dots, y_n, y_{n+1}, y_{n+2}, \dots, y_{n+m}$ (one for each wire)

Clauses: For every NAND gate $y_i = \text{NAND}(y_j, y_k)$, create clauses:



- ▶ $y_i \vee y_j \vee y_k$ (if $y_j = 0$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee \bar{y}_j \vee y_k$ (if $y_j = 1$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee y_j \vee \bar{y}_k$

Reduction to 3-SAT

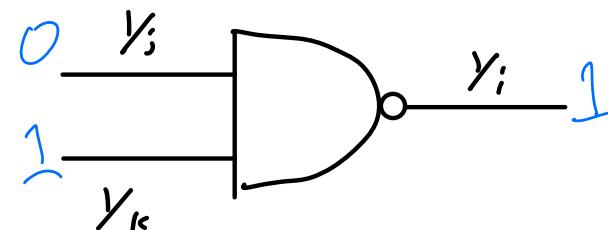
So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Variables: $y_1, y_2, \dots, y_n, y_{n+1}, y_{n+2}, \dots, y_{n+m}$ (one for each wire)

Clauses: For every NAND gate $y_i = \text{NAND}(y_j, y_k)$, create clauses:



- ▶ $y_i \vee y_j \vee y_k$ (if $y_j = 0$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee \bar{y}_j \vee y_k$ (if $y_j = 1$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee y_j \vee \bar{y}_k$ (if $y_j = 0$ and $y_k = 1$ then $y_i = 1$)

Reduction to 3-SAT

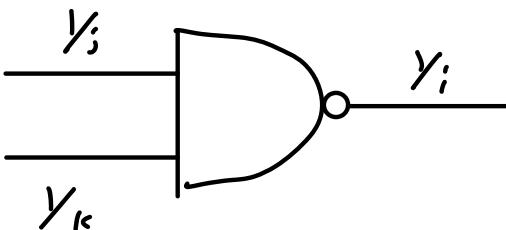
So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Variables: $y_1, y_2, \dots, y_n, y_{n+1}, y_{n+2}, \dots, y_{n+m}$ (one for each wire)

Clauses: For every NAND gate $y_i = \text{NAND}(y_j, y_k)$, create clauses:



- ▶ $y_i \vee y_j \vee y_k$ (if $y_j = 0$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee \bar{y}_j \vee y_k$ (if $y_j = 1$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee y_j \vee \bar{y}_k$ (if $y_j = 0$ and $y_k = 1$ then $y_i = 1$)
- ▶ $\bar{y}_i \vee \bar{y}_j \vee \bar{y}_k$

Reduction to 3-SAT

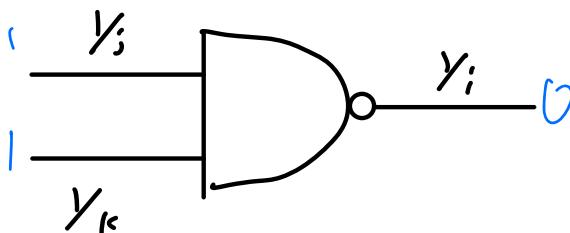
So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Variables: $y_1, y_2, \dots, y_n, y_{n+1}, y_{n+2}, \dots, y_{n+m}$ (one for each wire)

Clauses: For every NAND gate $y_i = \text{NAND}(y_j, y_k)$, create clauses:



- ▶ $y_i \vee y_j \vee y_k$ (if $y_j = 0$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee \bar{y}_j \vee y_k$ (if $y_j = 1$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee y_j \vee \bar{y}_k$ (if $y_j = 0$ and $y_k = 1$ then $y_i = 1$)
- ▶ $\bar{y}_i \vee \bar{y}_j \vee \bar{y}_k$ (if $y_j = 1$ and $y_k = 1$ then $y_i = 0$)

Reduction to 3-SAT

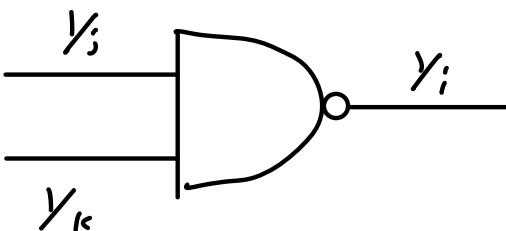
So given as input a circuit C :

- ▶ n “input wires” x_1, x_2, \dots, x_n
- ▶ m NAND gates: g_1, \dots, g_m . Output gate g_m

Need to construct many-one reduction f to 3-SAT: in polynomial time, construct 3-CNF formula $f(C)$ such that $f(C)$ has a satisfying assignment if and only if C has an input where it outputs 1.

Variables: $y_1, y_2, \dots, y_n, y_{n+1}, y_{n+2}, \dots, y_{n+m}$ (one for each wire)

Clauses: For every NAND gate $y_i = \text{NAND}(y_j, y_k)$, create clauses:



- ▶ $y_i \vee y_j \vee y_k$ (if $y_j = 0$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee \bar{y}_j \vee y_k$ (if $y_j = 1$ and $y_k = 0$ then $y_i = 1$)
- ▶ $y_i \vee y_j \vee \bar{y}_k$ (if $y_j = 0$ and $y_k = 1$ then $y_i = 1$)
- ▶ $\bar{y}_i \vee \bar{y}_j \vee \bar{y}_k$ (if $y_j = 1$ and $y_k = 1$ then $y_i = 0$)

Also add clause (y_{m+n}) (want output gate to be 1)

Analysis

Theorem

This is a many-one reduction from Circuit-SAT to 3-SAT.

Analysis

Theorem

This is a many-one reduction from Circuit-SAT to 3-SAT.

Polytime: ✓

Analysis

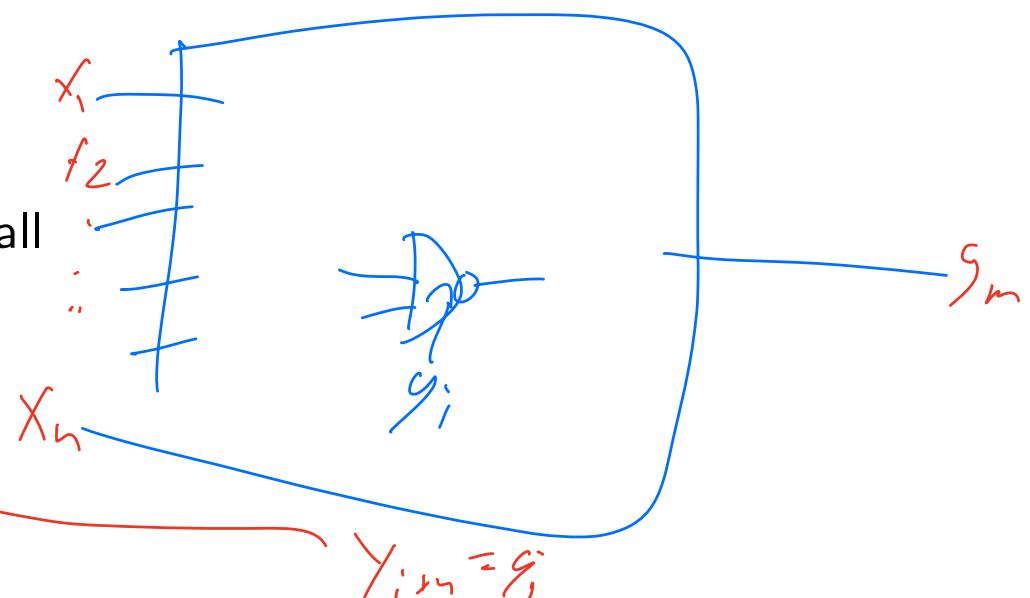
Theorem

This is a many-one reduction from Circuit-SAT to 3-SAT.

Polytime: ✓

Suppose C YES of Circuit-SAT

- ⇒ ∃ setting x of input wires so $g_m = 1$
- ⇒ ∃ assignment of y_1, \dots, y_{m+n} so that all clauses are satisfied:
 - ▶ $y_i = x_i$ if $i \leq n$
 - ▶ $y_i = g_{i-n}$ if $i > n$
- ⇒ $f(C)$ YES of 3-SAT



Analysis

Theorem

This is a many-one reduction from Circuit-SAT to 3-SAT.

Polytime: ✓

Suppose \mathbf{C} YES of Circuit-SAT

- ⇒ \exists setting \mathbf{x} of input wires so $\mathbf{g}_m = \mathbf{1}$
- ⇒ \exists assignment of $\mathbf{y}_1, \dots, \mathbf{y}_{m+n}$ so that all clauses are satisfied:
 - ▶ $\mathbf{y}_i = \mathbf{x}_i$ if $i \leq n$
 - ▶ $\mathbf{y}_i = \mathbf{g}_{i-n}$ if $i > n$
- ⇒ $f(\mathbf{C})$ YES of 3-SAT

Suppose $f(\mathbf{C})$ YES of 3-SAT

- ⇒ \exists assignment \mathbf{y} to variables so that all clauses satisfied
- ⇒ \exists setting \mathbf{x} of input wires so $\mathbf{g}_m = \mathbf{1}$:
 - ▶ $\mathbf{x}_i = \mathbf{y}_i$
 - ▶ Output of gate $\mathbf{g}_i = \mathbf{y}_{i+n}$ (by construction)
 - ▶ So $\mathbf{g}_m = \mathbf{1}$ (since (\mathbf{y}_{m+n}) is a clause)
- ⇒ \mathbf{C} a YES instance of Circuit-SAT

General Methodology to Prove Q NP-Complete

1. Show Q is in NP

- ▶ Can verify witness for YES
- ▶ Can catch false witness for NO (or contrapositive: if witness is verified, then a YES instance)

2. Find some NP -hard problem A . Reduce *from A to Q*:

- ▶ Given instance I of A , turn into $f(I)$ of Q (in time polynomial in $|I|$)
- ▶ I YES of A if and only if $f(I)$ YES of Q

General Methodology to Prove Q NP-Complete

1. Show Q is in NP

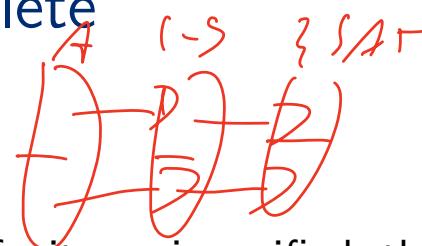
- ▶ Can verify witness for YES
- ▶ Can catch false witness for NO (or contrapositive: if witness is verified, then a YES instance)

2. Find some NP -hard problem A . Reduce *from A to Q*:

- ▶ Given instance I of A , turn into $f(I)$ of Q (in time polynomial in $|I|$)
- ▶ I YES of A if and only if $f(I)$ YES of Q

Notes:

- ▶ Careful about direction of reduction!!!!
- ▶ Need to handle *arbitrary* instances of A , but can turn into very structured instances of Q
- ▶ Often easiest to prove NO direction via contrapositive, to turn into statement about YES:
 - ▶ I YES of $A \implies f(I)$ YES of Q
 - ▶ $f(I)$ YES of $Q \implies I$ YES of A *NCT* $\neg I$ YES of $A \implies \neg f(I)$ YES of Q
 - ▶ So proving “both directions”, but reduction only in one direction.



CLIQUE

Definition: A *clique* in an undirected graph $G = (V, E)$ is a set $S \subseteq V$ such that $\{u, v\} \in E$ for all $u, v \in S$

Definition (CLIQUE)

Instance is a graph $G = (V, E)$ and an integer k . YES if G contains a clique of size at least k , NO otherwise.

CLIQUE

Definition: A *clique* in an undirected graph $G = (V, E)$ is a set $S \subseteq V$ such that $\{u, v\} \in E$ for all $u, v \in S$

Definition (CLIQUE)

Instance is a graph $G = (V, E)$ and an integer k . YES if G contains a clique of size at least k , NO otherwise.

Theorem

CLIQUE is **NP**-complete.

CLIQUE

Definition: A *clique* in an undirected graph $G = (V, E)$ is a set $S \subseteq V$ such that $\{u, v\} \in E$ for all $u, v \in S$

Definition (CLIQUE)

Instance is a graph $G = (V, E)$ and an integer k . YES if G contains a clique of size at least k , NO otherwise.

Theorem

CLIQUE is **NP**-complete.

In **NP**:

CLIQUE

Definition: A *clique* in an undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ is a set $S \subseteq V$ such that $\{u, v\} \in E$ for all $u, v \in S$

Definition (CLIQUE)

Instance is a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and an integer k . YES if \mathbf{G} contains a clique of size at least k , NO otherwise.

Theorem

CLIQUE is **NP**-complete.

In NP:

- ▶ Witness: $S \subseteq V$
- ▶ Verifier: Checks if S is a clique and $|S| \geq k$
 - ▶ If (G, k) a YES instance: there is a clique S of size $\geq k$ on which verifier returns YES
 - ▶ If (G, k) a NO instance: S cannot be clique of size $\geq k$, so verifier always returns NO

CLIQUE is NP -hard

Prove by reducing 3-SAT to CLIQUE

- ▶ For arbitrary $A \in NP$, would have $A \leq_p \text{Circuit-SAT} \leq_p \text{3-SAT} \leq_p \text{CLIQUE}$

CLIQUE is *NP*-hard

Prove by reducing 3-SAT to CLIQUE

- ▶ For arbitrary $A \in NP$, would have $A \leq_p \text{Circuit-SAT} \leq_p \text{3-SAT} \leq_p \text{CLIQUE}$

Given 3-SAT formula F (with n variables and m clauses), set $k = m$ and create graph

$G = (V, E)$:

- ▶ For every clause of F , for every satisfying assignment to the clause, create vertex
- ▶ Add an edge between consistent assignments

CLIQUE is *NP*-hard

Prove by reducing 3-SAT to CLIQUE

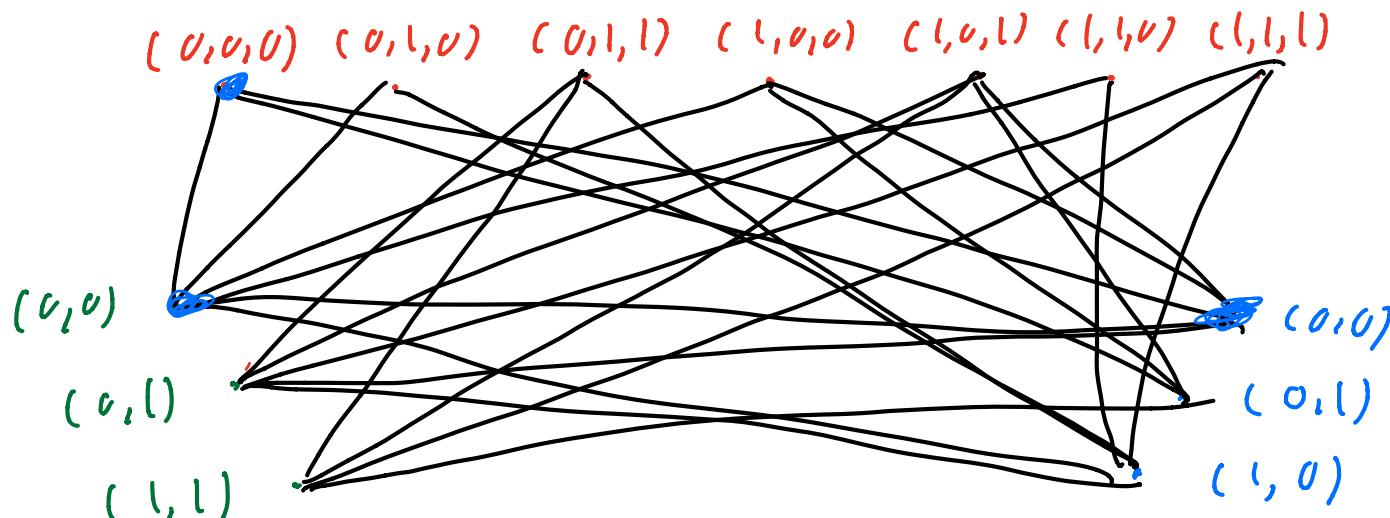
- ▶ For arbitrary $A \in NP$, would have $A \leq_p$ Circuit-SAT \leq_p 3-SAT \leq_p CLIQUE

Given 3-SAT formula F (with n variables and m clauses), set $k = m$ and create graph

$G = (V, E)$:

- ▶ For every clause of F , for every satisfying assignment to the clause, create vertex
- ▶ Add an edge between consistent assignments

Example: $F = (x_1 \vee x_2 \vee \bar{x}_4) \wedge (\bar{x}_3 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_3)$



3-SAT to CLIQUE reduction analysis

Polytime: ✓

3-SAT to CLIQUE reduction analysis

Polytime: ✓

If F YES of 3-SAT:

- ▶ There is some satisfying assignment x
- ▶ For every clause, choose vertex corresponding to x . Let S be chosen vertices
- ▶ $|S| = m = k$, and clique since all consistent (since all from x)

⇒ (G, k) YES of CLIQUE

3-SAT to CLIQUE reduction analysis

Polytime: ✓

If F YES of 3-SAT:

- ▶ There is some satisfying assignment x
- ▶ For every clause, choose vertex corresponding to x . Let S be chosen vertices
- ▶ $|S| = m = k$, and clique since all consistent (since all from x)

⇒ (G, k) YES of CLIQUE

If (G, k) YES of CLIQUE:

- ▶ There is some clique S of size $k = m$
- ▶ Must contain exactly one vertex from each clause (since clique of size m)
- ▶ Since clique, all assignments consistent ⇒ there is an assignment that satisfies all clauses

⇒ F YES of 3-SAT

INDEPENDENT SET

Definition: $S \subseteq V$ is an *independent set* in $G = (V, E)$ if $\{u, v\} \notin E$ for all $u, v \in S$

Definition (INDEPENDENT SET)

Instance is graph $G = (V, E)$ and integer k . YES if G has an independent set of size $\geq k$, NO otherwise.

INDEPENDENT SET

Definition: $S \subseteq V$ is an *independent set* in $G = (V, E)$ if $\{u, v\} \notin E$ for all $u, v \in S$

Definition (INDEPENDENT SET)

Instance is graph $G = (V, E)$ and integer k . YES if G has an independent set of size $\geq k$, NO otherwise.

Theorem

INDEPENDENT SET is **NP**-complete.

INDEPENDENT SET

Definition: $S \subseteq V$ is an *independent set* in $G = (V, E)$ if $\{u, v\} \notin E$ for all $u, v \in S$

Definition (INDEPENDENT SET)

Instance is graph $G = (V, E)$ and integer k . YES if G has an independent set of size $\geq k$, NO otherwise.

Theorem

INDEPENDENT SET is **NP**-complete.

In **NP**:

INDEPENDENT SET

Definition: $S \subseteq V$ is an *independent set* in $G = (V, E)$ if $\{u, v\} \notin E$ for all $u, v \in S$

Definition (INDEPENDENT SET)

Instance is graph $G = (V, E)$ and integer k . YES if G has an independent set of size $\geq k$, NO otherwise.

Theorem

INDEPENDENT SET is **NP**-complete.

In **NP**:

- ▶ Witness is $S \subseteq V$. Verifier checks that $|S| \geq k$ and no edges in S
- ▶ If (G, k) a YES instance then such an S exists \implies verifier returns YES on it.
- ▶ If (G, k) a NO then verifier will return NO on every S .

INDEPENDENT SET is *NP*-hard

Reduce from:

INDEPENDENT SET is *NP*-hard

Reduce from: CLIQUE

INDEPENDENT SET is *NP*-hard

Reduce from: CLIQUE

- ▶ Given instance (G, k) of CLIQUE, create “complement graph” H : same vertex set, with $\{u, v\} \in E(H)$ if and only if $\{u, v\} \notin E(G)$
- ▶ Instance (H, k) of INDEPENDENT SET

INDEPENDENT SET is *NP*-hard

Reduce from: CLIQUE

- ▶ Given instance (G, k) of CLIQUE, create “complement graph” H : same vertex set, with $\{u, v\} \in E(H)$ if and only if $\{u, v\} \notin E(G)$
- ▶ Instance (H, k) of INDEPENDENT SET

If (G, k) YES of CLIQUE:

- ⇒ Clique $S \subseteq V$ of G with $|S| \geq k$
- ⇒ S an independent set in H

INDEPENDENT SET is *NP*-hard

Reduce from: CLIQUE

- ▶ Given instance (G, k) of CLIQUE, create “complement graph” H : same vertex set, with $\{u, v\} \in E(H)$ if and only if $\{u, v\} \notin E(G)$
- ▶ Instance (H, k) of INDEPENDENT SET

If (G, k) YES of CLIQUE:

- ⇒ Clique $S \subseteq V$ of G with $|S| \geq k$
- ⇒ S an independent set in H

If (H, k) YES of INDEPENDENT SET:

- ⇒ Independent set $S \subseteq V$ in H with $|S| \geq k$
- ⇒ S a clique in G

VERTEX COVER

Definition: $S \subseteq V$ is a *vertex cover* of $G = (V, E)$ if $S \cap e \neq \emptyset$ for all $e \in E$

Definition (VERTEX COVER)

Instance is graph $G = (V, E)$, integer k . YES if G has a vertex cover of size $\leq k$, NO otherwise.

VERTEX COVER

Definition: $S \subseteq V$ is a *vertex cover* of $G = (V, E)$ if $S \cap e \neq \emptyset$ for all $e \in E$

Definition (VERTEX COVER)

Instance is graph $G = (V, E)$, integer k . YES if G has a vertex cover of size $\leq k$, NO otherwise.

Theorem

VERTEX COVER is *NP-complete*

VERTEX COVER

Definition: $S \subseteq V$ is a *vertex cover* of $G = (V, E)$ if $S \cap e \neq \emptyset$ for all $e \in E$

Definition (VERTEX COVER)

Instance is graph $G = (V, E)$, integer k . YES if G has a vertex cover of size $\leq k$, NO otherwise.

Theorem

VERTEX COVER is *NP-complete*

In *NP*:

VERTEX COVER

Definition: $S \subseteq V$ is a *vertex cover* of $G = (V, E)$ if $S \cap e \neq \emptyset$ for all $e \in E$

Definition (VERTEX COVER)

Instance is graph $G = (V, E)$, integer k . YES if G has a vertex cover of size $\leq k$, NO otherwise.

Theorem

VERTEX COVER is *NP-complete*

In *NP*:

- ▶ Witness is $S \subseteq V$. Verifier checks that $|S| \leq k$ and every edge has at least one endpoint in S
- ▶ If (G, k) a YES instance then such an S exists \implies verifier returns YES on it.
- ▶ If (G, k) a NO then verifier will return NO on every S .

VERTEX COVER is *NP*-hard

Reduce from INDEPENDENT SET

- ▶ Given instance $(G = (V, E), k)$ of INDEPENDENT SET, create instance $(G, n - k)$ of VERTEX COVER (where $n = |V|$)

VERTEX COVER is *NP*-hard

Reduce from INDEPENDENT SET

- ▶ Given instance $(G = (V, E), k)$ of INDEPENDENT SET, create instance $(G, n - k)$ of VERTEX COVER (where $n = |V|$)

If (G, k) a YES instance of INDEPENDENT SET:

- ⇒ G has an independent set S with $|S| \geq k$
- ⇒ $V \setminus S$ a vertex cover of G of size $\leq n - k$
- ⇒ $(G, n - k)$ a YES instance of VERTEX COVER

VERTEX COVER is *NP*-hard

Reduce from INDEPENDENT SET

- ▶ Given instance $(G = (V, E), k)$ of INDEPENDENT SET, create instance $(G, n - k)$ of VERTEX COVER (where $n = |V|$)

If (G, k) a YES instance of INDEPENDENT SET:

- ⇒ G has an independent set S with $|S| \geq k$
- ⇒ $V \setminus S$ a vertex cover of G of size $\leq n - k$
- ⇒ $(G, n - k)$ a YES instance of VERTEX COVER

If $(G, n - k)$ a YES instance of VERTEX COVER:

- ⇒ G has a vertex cover S of size at most $n - k$
- ⇒ $V \setminus S$ an independent set of G of size at least k
- ⇒ (G, k) a YES instance of INDEPENDENT SET