NP hardness reductions

Lecture 14

Recap

- **P** = YES/NO questions that can be answered in polynomial time in input size (algorithm)
- **NP**= YES/No problems where YES instance can be verified in polynomial time
- X is **NP-hard**: X in P implies P=NP
- Cook-Levin: CircuitSAT NP hard

How to prove NP hardness

- To prove X is NP-hard:

- **Step 1**: Pick a known NP-hard problem Y
- **Step 2:** Assume for the sake of argument, a polynomial time algorithm for X.
- **Step 3**: Derive a polynomial time algorithm for Y, using algorithm for X as subroutine.
- **Step 4**: Contradiction

Reduce Y to X

Reduce FROM the problem
I know about
TO the problem
I am curious about

NP hardness

- Library of NP-hard problems

CircuitSAT

SAT

?

Let's assume the problem is easy
and see what ridiculous consequences follow

3SAT

- Look at boolean formulas in CNF

$$\overbrace{(a \vee b \vee c \vee d)}^{\text{clause}} \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b})$$

3SAT

- Look at boolean formulas in CNF

$$\overbrace{(a \vee b \vee c \vee d)}^{\text{clause}} \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b})$$

3SAT: exactly three
literals per clause!
every literal is a variable or
the negation of a variable

3SAT

- Look at boolean formulas in CNF

$$\overbrace{(a \vee b \vee c \vee d)}^{\text{clause}} \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b})$$

3SAT: exactly three literals per clause!
every literal is a variable or the negation of a variable

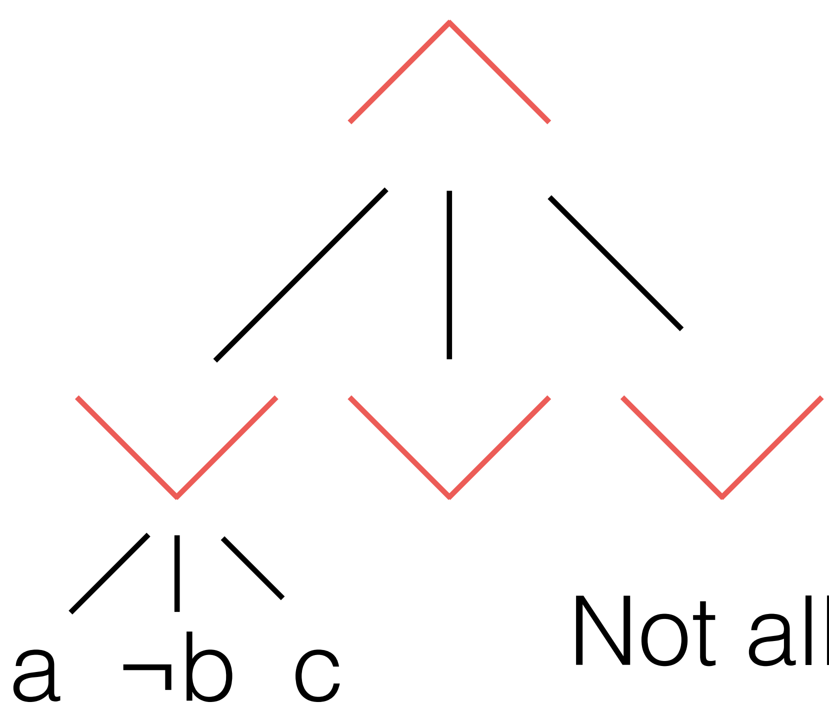
Not all boolean functions can be in the form aVbVcVd

3SAT

- Look at boolean formulas in CNF

$$\overbrace{(a \vee b \vee c \vee d)}^{\text{clause}} \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b})$$

Parse tree:



3SAT: exactly three literals per clause!
every literal is a variable or the negation of a variable

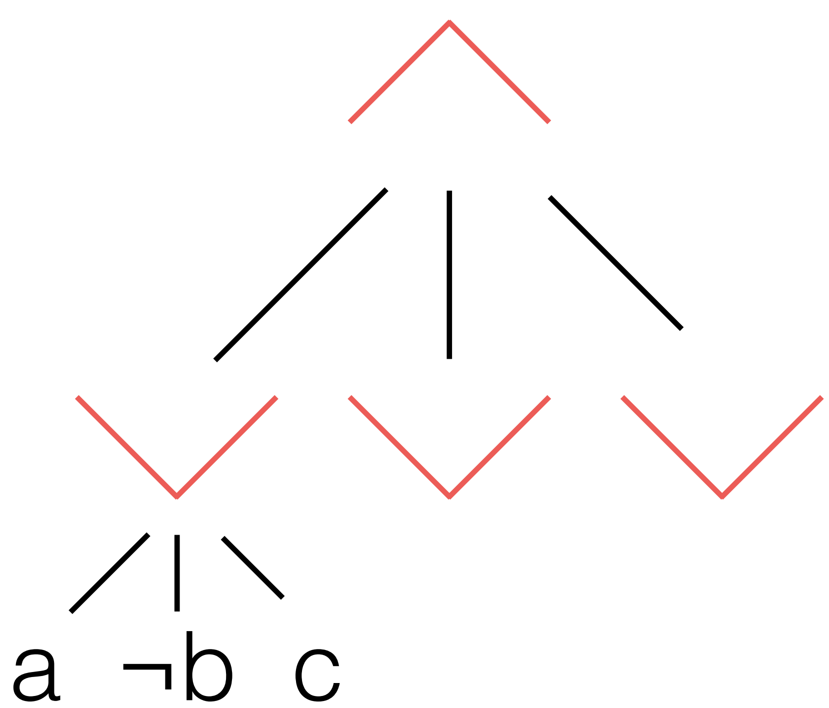
Not all boolean functions can be in the form aVbVcVd

3SAT

- Look at boolean formulas in CNF

$$\overbrace{(a \vee b \vee c \vee d)}^{\text{clause}} \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b})$$

Parse tree:



3SAT special case of SAT.
unlike when we are thinking
about special cases in algorithms

2SAT there is algorithm!

NP hardness

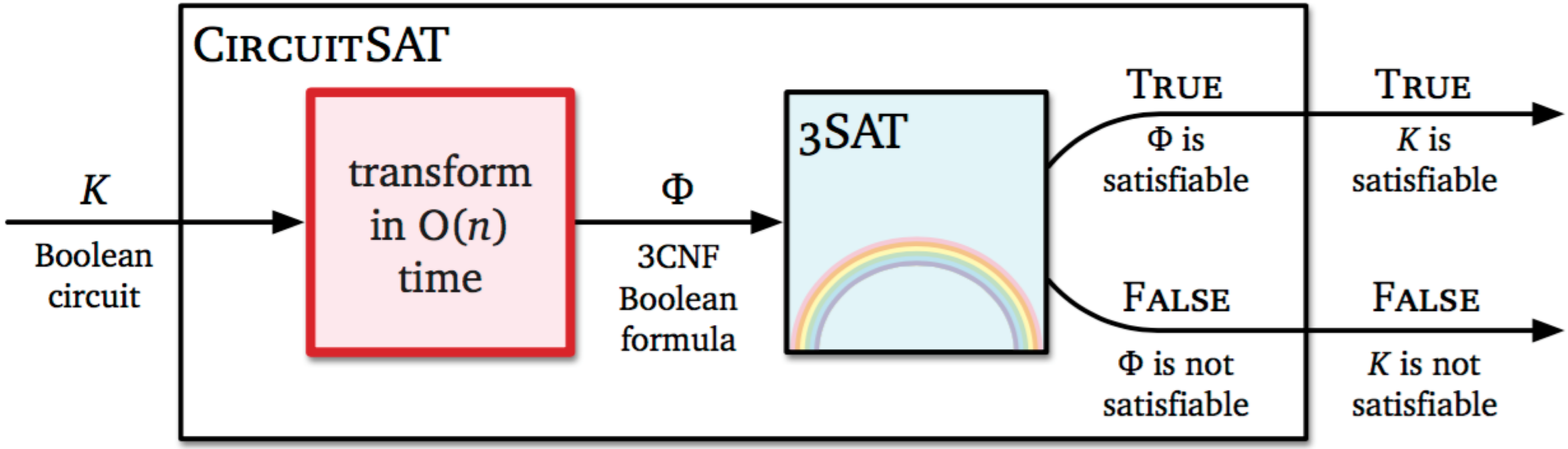
- Poly time reduction from CircuitSAT.

- If there is a poly time algorithm to solve 3SAT, then there is poly time algorithm to solve CircuitSAT

NP hardness

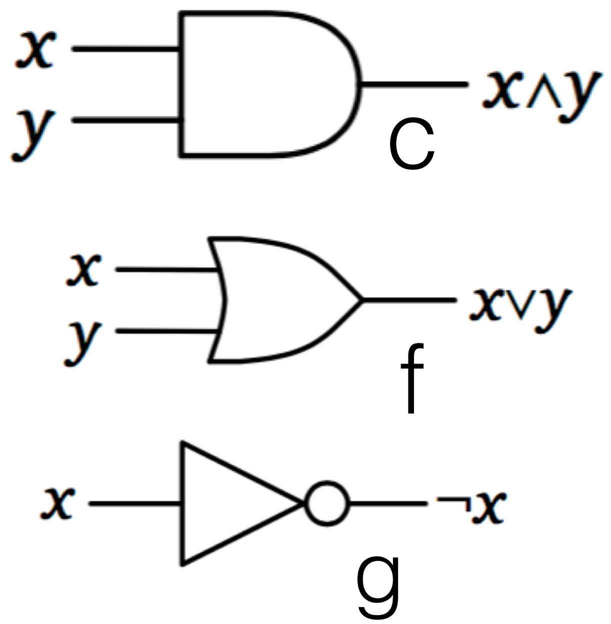
- Poly time reduction from CircuitSAT.

- If there is a poly time algorithm to solve 3SAT, then there is poly time algorithm to solve CircuitSAT



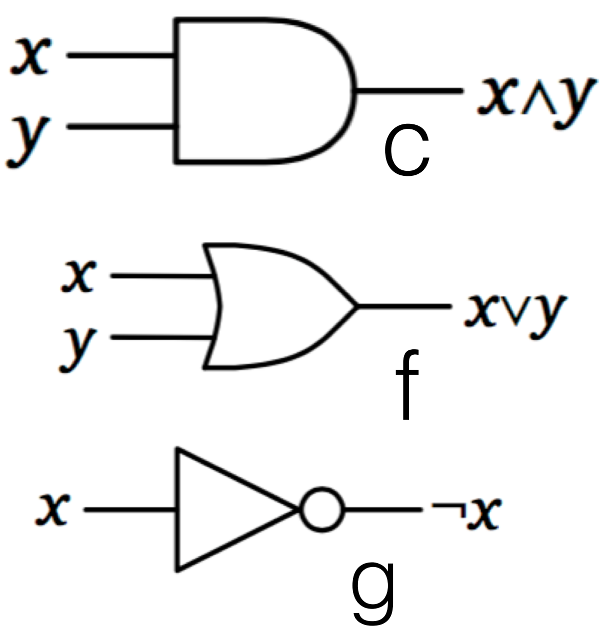
Reduction CircuitSAT to 3SAT

- **Step 1**: Make gates binary (blows up size by at most 2x wires, if there were x wires). Poly time.
- **Step 2:** Transcribe



Reduction CircuitSAT to 3SAT

- **Step 1**: Make gates binary (blows up size by at most 2x wires, if there were x wires). Poly time.
- **Step 2:** Transcribe



c=x∧y

∧

f=x∨y

∧

g=¬x

Reduction CircuitSAT to 3SAT

- **Step 3:** Make clauses in 3CNF

Reduction CircuitSAT to 3SAT

- **Step 3:** Make clauses in 3CNF

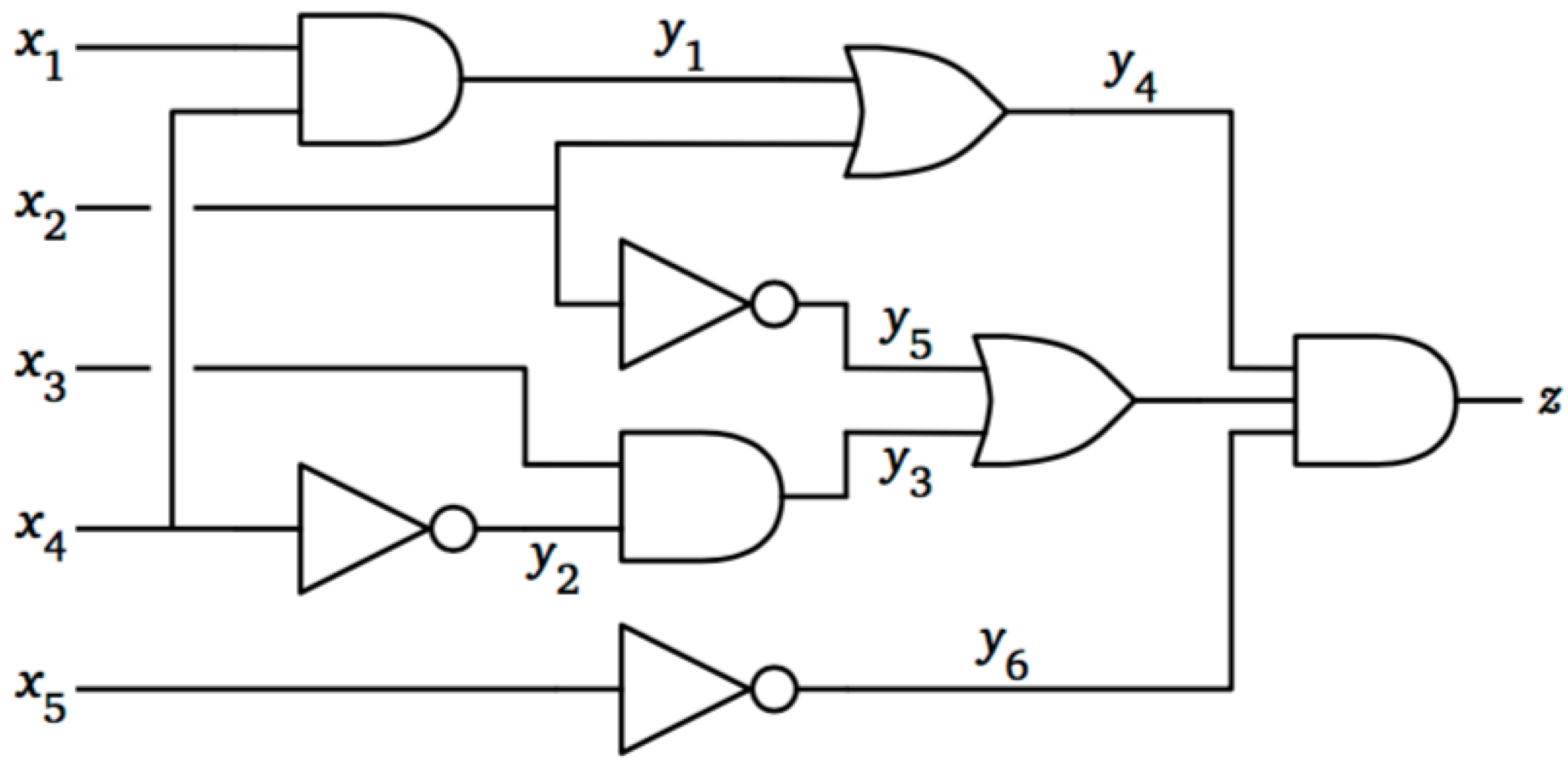
$$a = b \wedge c \quad \longmapsto \quad (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee c)$$

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

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

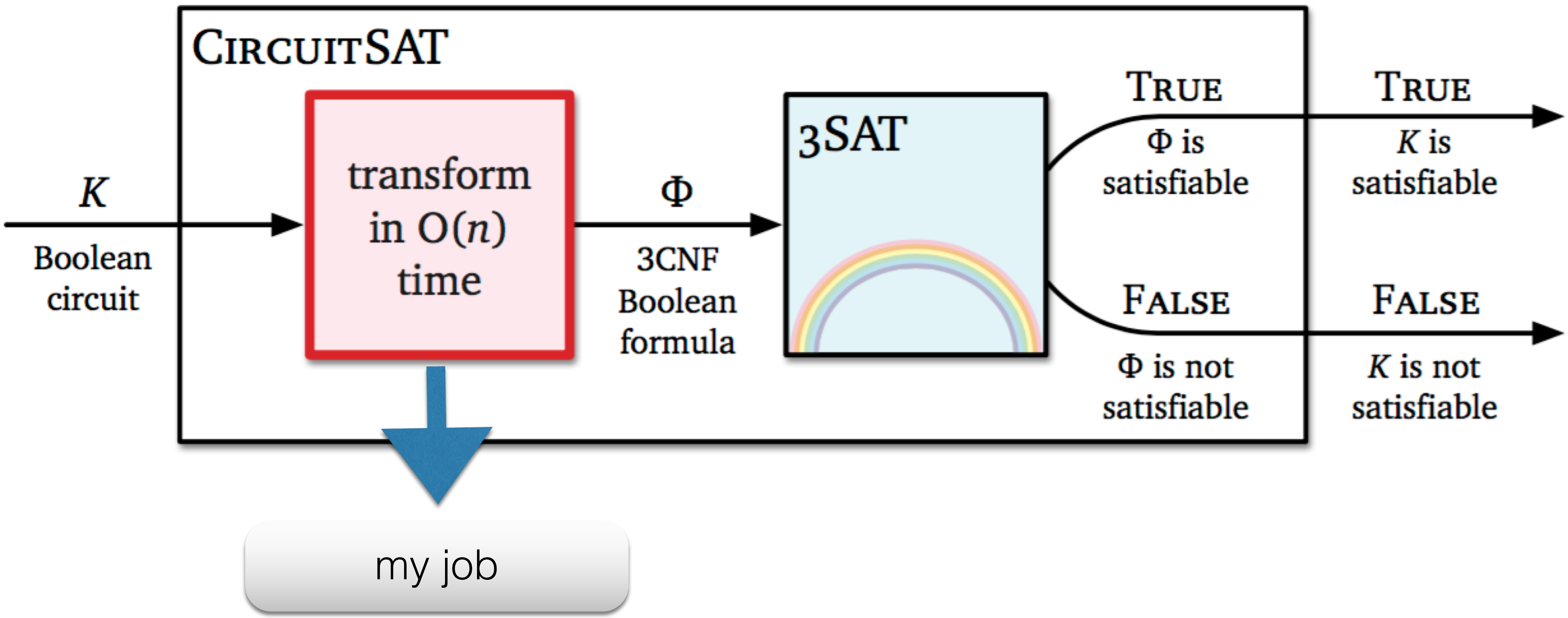
$$a \vee b \longmapsto (a \vee b \vee x) \wedge (a \vee b \vee \bar{x})$$

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



$$
\begin{aligned}
&(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}})
\end{aligned}
$$

Although this formula may look a lot more ugly and complicated than the original circuit at first glance, it's actually only a constant factor larger—every binary gate in the original



- Algo runs in poly time

Proof:

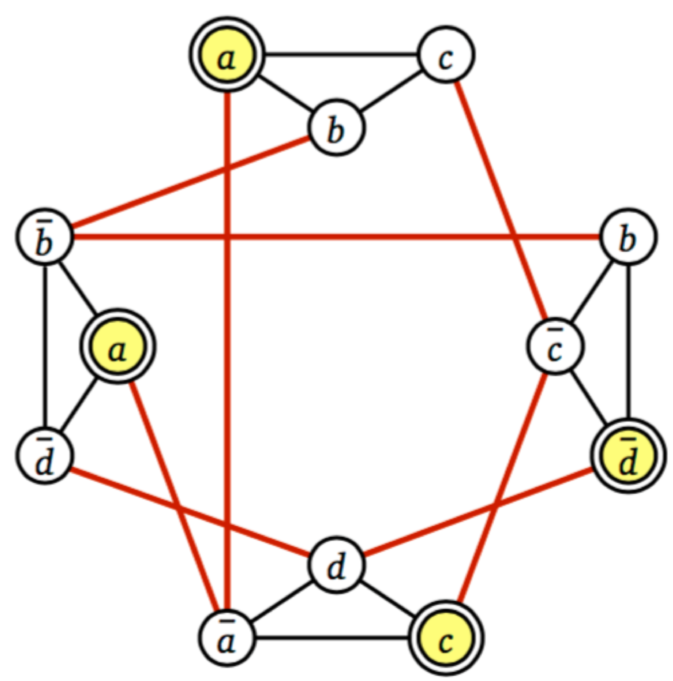
- Circuit satisfiable implies formula satisfiable
- formula satisfiable implies circuit satisfiable

even though the reduction goes one direction, the proof needs to go both directions

MAX Independent Set

- Input: a graph G(V,E)

- Output: Largest subset of vertices with no edges between them. (enough to find size)



How to prove NP hardness

- To prove X is NP-hard:

- **Step 1**: Pick a known NP-hard problem Y
- **Step 2:** Assume for the sake of argument, a polynomial time algorithm for X.
- **Step 3**: Derive a polynomial time algorithm for Y, using algorithm for X as subroutine.
- **Step 4**: Contradiction

Reduce Y to X

Reduce FROM the problem
I know about
TO the problem
I am curious about

NP hardness

- Library of NP-hard problems

CircuitSAT

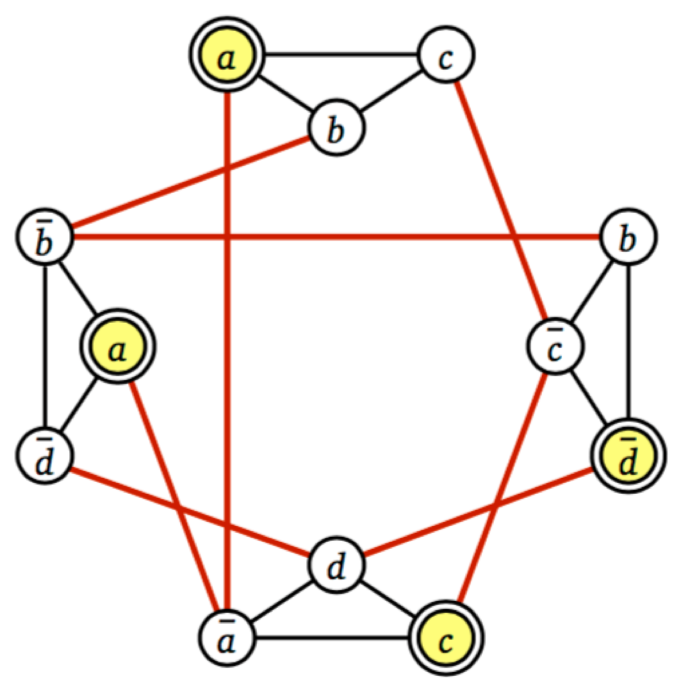
SAT

3SAT

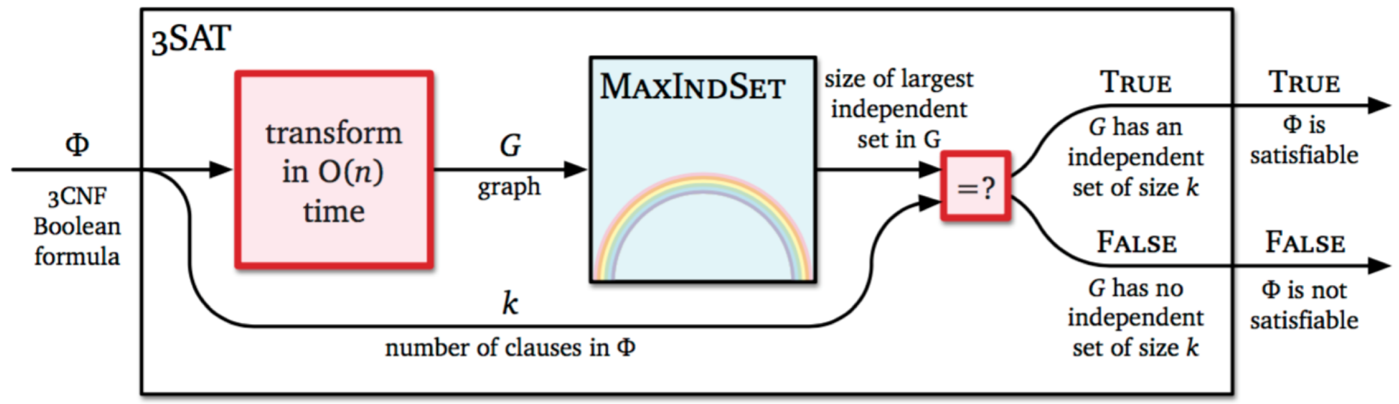
Let's assume the problem is easy
and see what ridiculous consequences follow

MAX Independent Set

- Input: a graph G(V,E)
- Output: Largest subset of vertices with no edges between them. (enough to find size)



Prove this is NP hard by reduction from 3SAT



Polynomial-time reduction from 3SAT to MaxIndSet

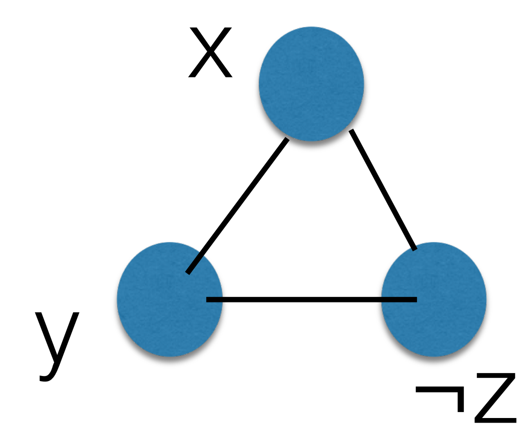
MAX Independent Set

- Given an arbitrary 3CNF formula

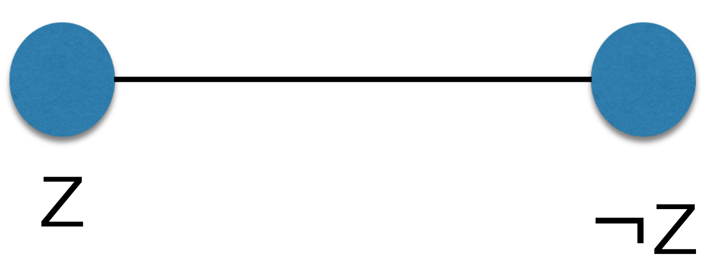
- Build a graph G as follows

1) For every clause 3 vertices connected in a triangle

$x \vee y \vee \neg z$



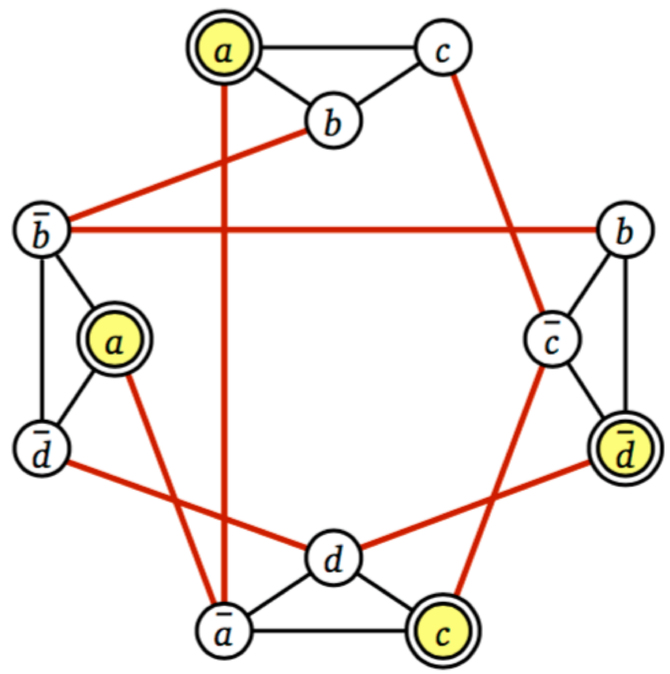
2) add edges between a literal and its negation



MAX Independent Set

- Input: a graph G(V,E)

- Output: Largest subset of vertices with no edges between them. (enough to find size)



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

k clauses -> 3k vertices
graph has IS of size k if and only if the formula is satisfiable

MAX Independent Set

Claim:
Graph has IS of size k if and only if the formula is satisfiable

2 steps to proof:

Step1) Assume formula satisfiable

-Choose satisfying assignment (a=1, b=1, c=1, d=0)

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

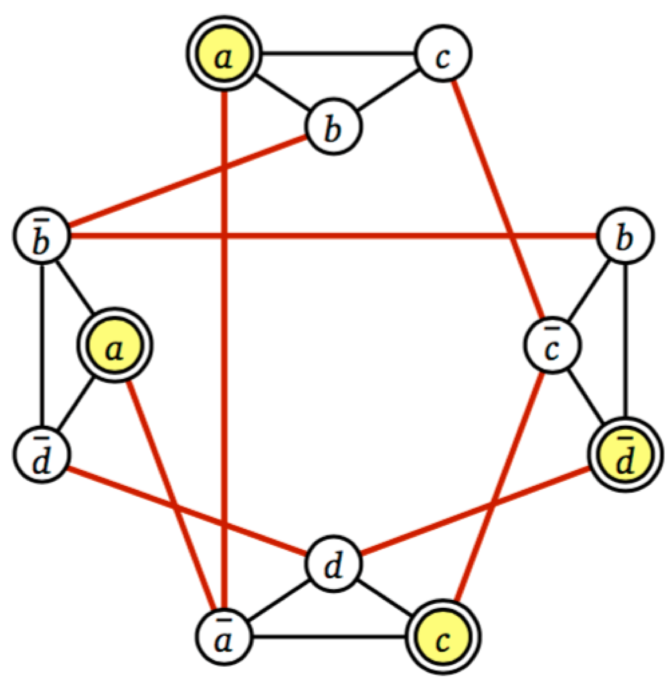
No long edges because every selected literal is true

and no edges between each triangle

G Has IS of size k!

MAX Independent Set

- Input: a graph G(V,E)
- Output: Largest subset of vertices with no edges between them. (enough to find size)



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

k clauses -> 3k vertices
graph has IS of size k if and only if the formula is satisfiable

MAX Independent Set

Claim:
Graph has IS of size k if and only if the formula is satisfiable

2 steps to proof:

Step 2) Suppose G has IS of size k. Then this IS contains at most one node per triangle so it has exactly one node per triangle. These nodes provide a satisfying assignment to the formula

When you write reductions

- **Step 1**: Describe the algorithm (and it runs in poly time)
- **Step 2:** Prove one way
- **Step 3**: Prove the other way

NP hardness

- Library of NP-hard problems

CircuitSAT

SAT

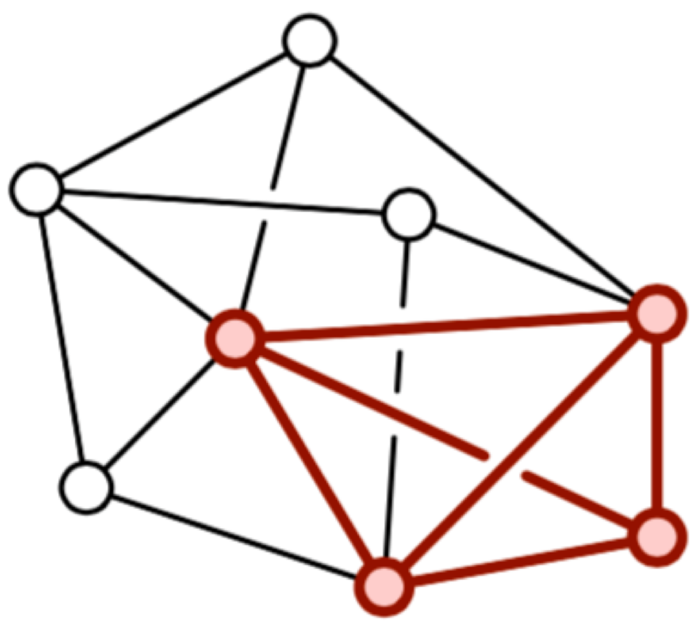
3SAT

MAX IS

Let's assume the problem is easy
and see what ridiculous consequences follow

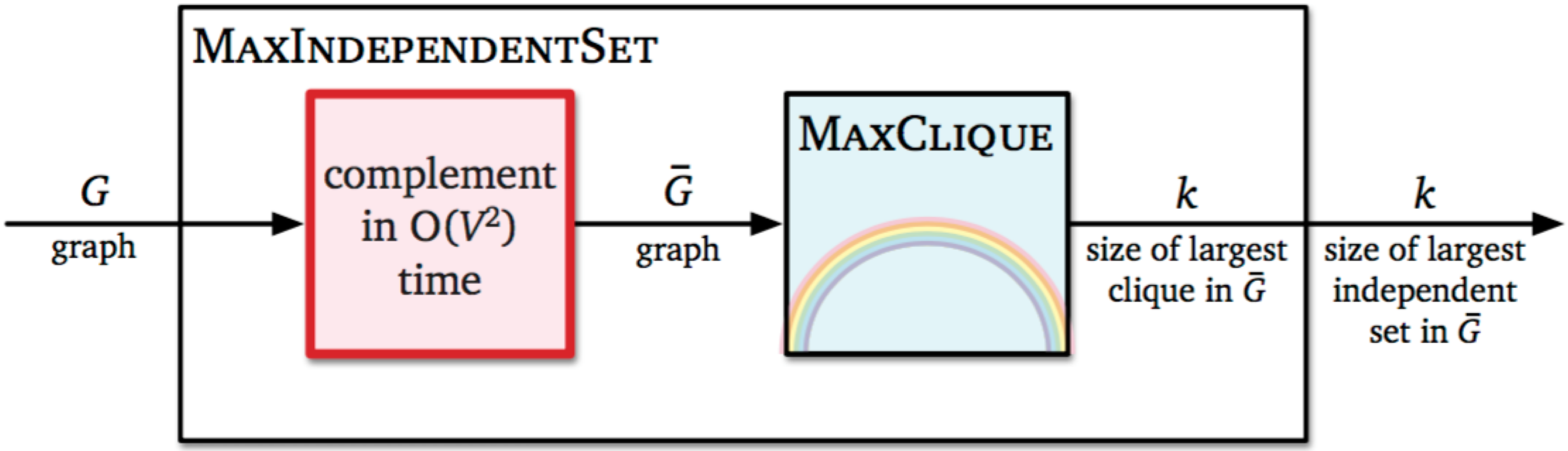
MAX Clique

- Input: a graph G(V,E)
- Output: Largest subset of vertices that are all pairwise connected

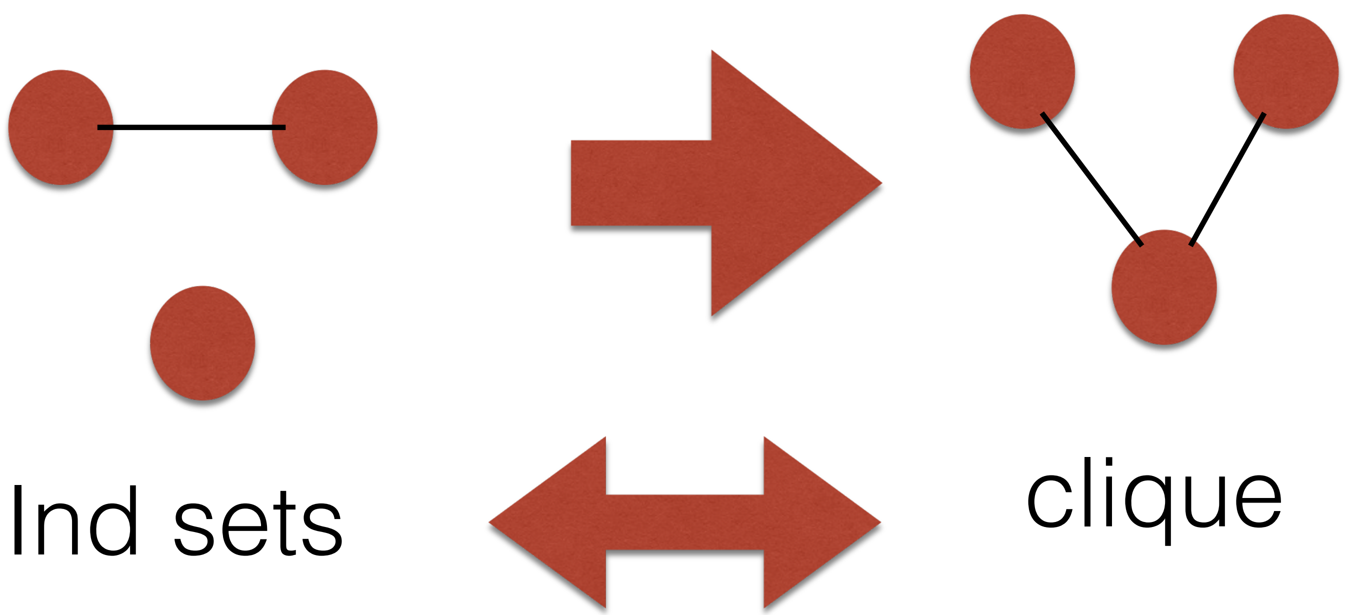


- Reduction from MAX-IS
- Assume poly time algorithm
- for MAX Clique
- Derive poly time algorithm

  for MAX IS



what is G’? Not the same graph as G



NP hardness

- Library of NP-hard problems

CircuitSAT

SAT

3SAT

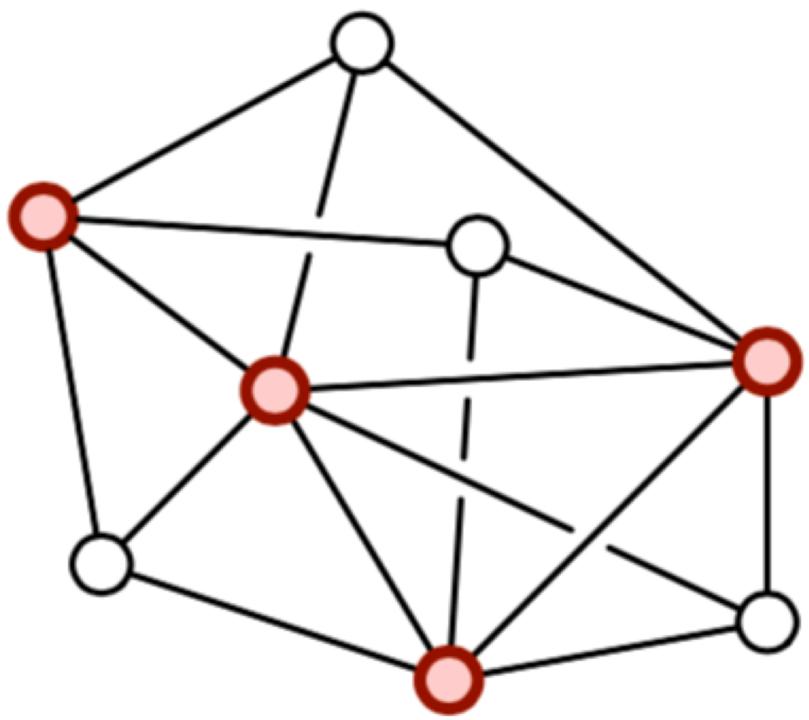
MAX IS

MAX Clique

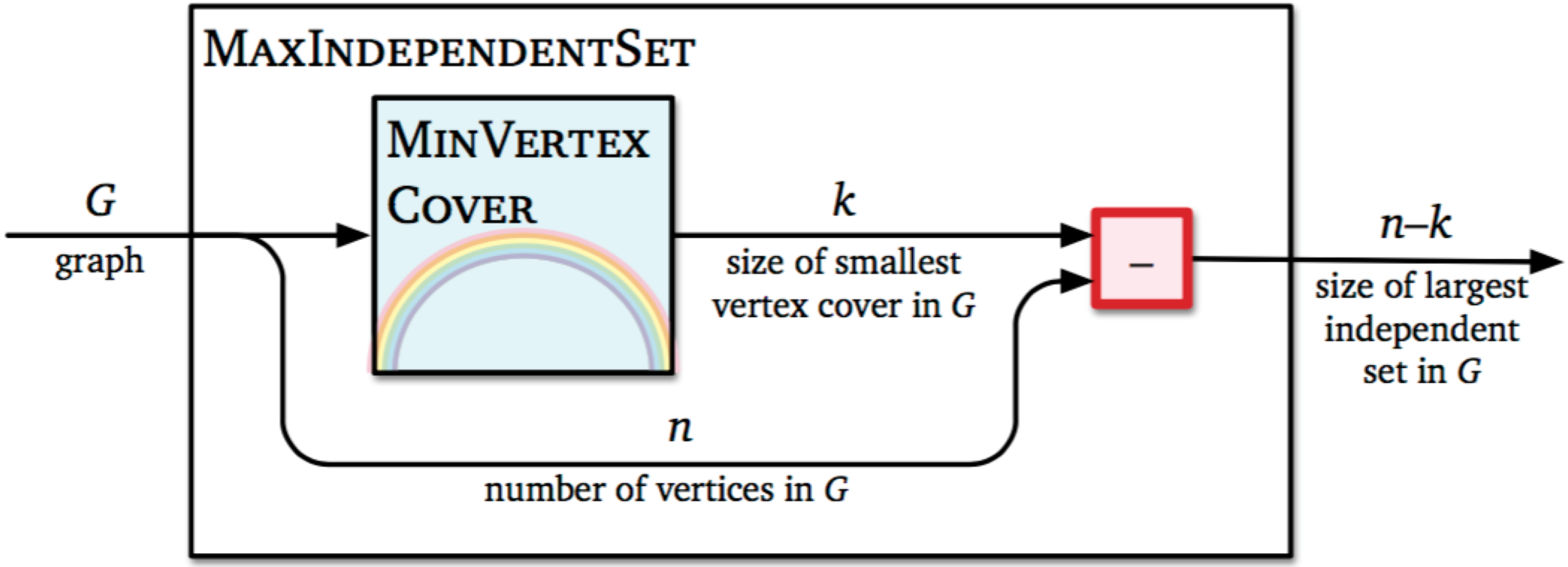
Let's assume the problem is easy
and see what ridiculous consequences follow

MIN Vertex Cover

- Input: a graph G(V,E)
- Output: Smallest set of vertices that touch every edge



- Reduction from MAX-IS
-
- Assume poly time algorithm for MIN Vertex Cover



what is G'? same graph as G
Output is different