# CS 344: Design and Analysis of Computer Algorithms

(Spring 2022 — Sections 5,6,7,8)

# Lecture 25: NP-hardness Reductions

# (Complexity) Classes

# P & NP

# Classes P and NP

- We use class to refer to a collection of problems

# Classes P and NP

- We use class to refer to a collection of problems
- Class P:
  - ALL problems that can be solved in polynomial time
- Class NP:
  - ALL problems that can be verified in polynomial time

# Classes P an

Problems we can solve efficiently

- We use class to refer to a collection of p ems
- Class P:
  - ALL problems that can be solved in polynomial time
- Class NP:
  - ALL problems that can be verified in polynomial time

Problems we “hope” to be able to solve efficiently

# Classes P and NP

- Clearly any problem in P is also in NP
  - If we can solve a problem in poly-time, we can definitely verify it in poly-time
- Big open question of Computer Science:

Is P=NP or not?

# NP-Hard & NP-Complete problems

# Plan

- We have a problem Q in NP
- We want to show that Q is impossible to solve in polynomial time
- We do not know how to do that

# Plan

- We have a problem Q in NP
- We want to show that Q is impossible to solve in polynomial time
- We do not know how to do that
- Instead, we go for the next best thing:
  - Show that Q is really hard to solve in polynomial time

# Plan

- We have a problem Q in NP
- We want to show that Q is impossible to solve in polynomial time
- We do not know how to do that
- Instead, we go for the next best thing:
  - Show that Q is really hard to solve in polynomial time
- **A simple approach:**
  - Show that if Q can be solved in polynomial time, then P = NP

# NP-Hard and NP-Complete Problems

- **NP-hard problems:**
  - We say a problem R is NP-hard if designing a poly-time algorithm for R implies P=NP
- **NP-complete problems:**
  - We say a problem R is NP-complete if (1) R is in NP itself, and (2) R is NP-hard

# Circuit-SAT problem & Cook-Levin Theorem

# Plan

- **Goal**: Show that Q is NP-hard
- **Approach:**
  - Find any problem R which is already known to be NP-hard
  - Show that R can be reduced to Q:
    - if Q can be solved in poly-time then R can also be solved in poly-time

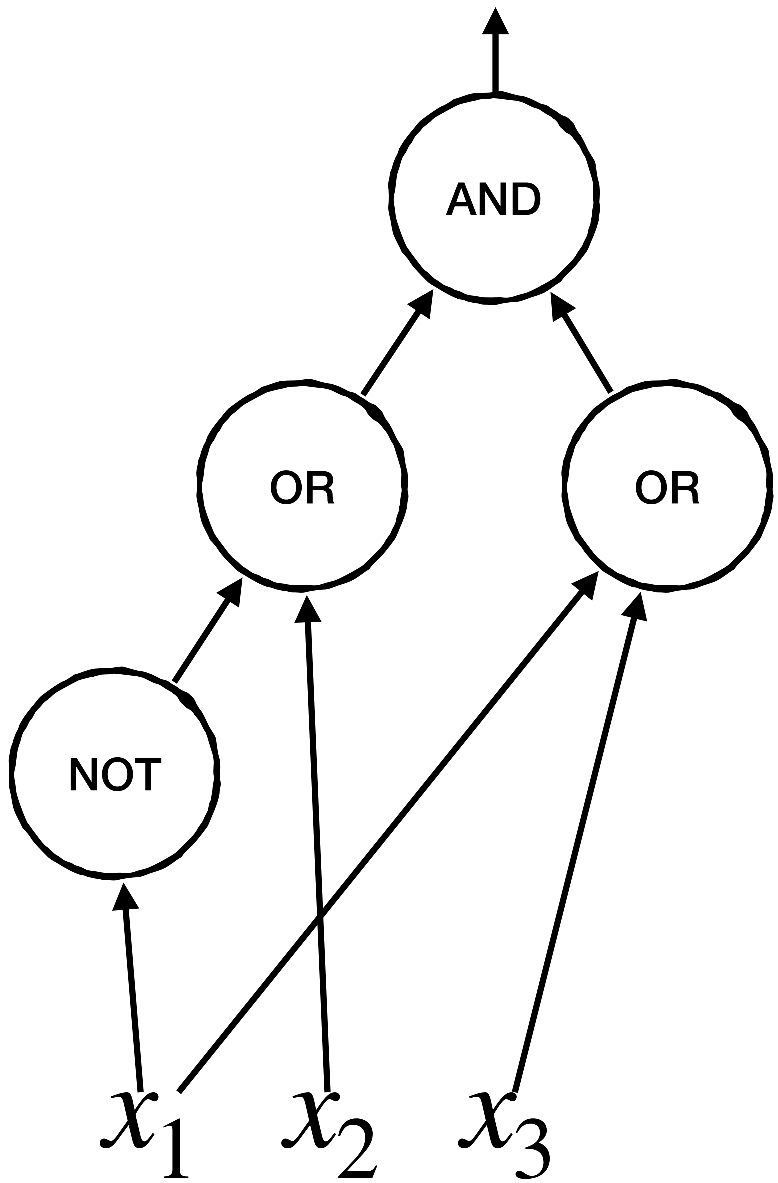
# Circuit-SAT Problem

- The very first NP-hard problem is called the circuit-satisfiability problem or **circuit-SAT**

# Circuit-SAT Problem

- The very first NP-hard problem is called the circuit-satisfiability problem or **circuit-SAT**
- **Input:**
  - A circuit C with binary AND and OR gates and unary NEGATE gates with n inputs in total
  - For any $x \in \{0,1\}^n$ we use C(x) to denote the value of circuit on the input x
- **Output:**
  - Is there any x such that C(x) = True?

# Circuit-SAT Problem: Example



# Circuit-SAT

- Circuit-SAT is in NP

# Circuit-SAT

- Circuit-SAT is in NP
- A poly-time verifier:
  - The proof is an assignment $x$ such that $C(x) = 1$
  - We can evaluate $x$ in $C$ to compute the answer — the evaluation is done bottom-up by computing value of each gate
- **Cook-Levin Theorem:** Circuit-SAT is NP-complete

# Reductions

- We now know that circuit-SAT is NP-complete (and so NP-hard)
- We can use circuit-SAT in reductions to prove other problems are also NP-hard

# Reductions

- We now know that circuit-SAT is NP-complete (and so NP-hard)
- We can use circuit-SAT in reductions to prove other problems are also NP-hard

# Example 1: 3-SAT is NP-Complete

# 3-SAT Problem

- **Definition:**
  - Literal: a variable or its negation
  - Clause: OR of a collection of literals
  - CNF-Formula: AND of a collection of clauses

# 3-SAT Problem

- **Definition:**
  - Literal: a variable or its negation $x$
  - Clause: OR of a collection of literals $(X \vee Y \vee \bar{Z})$
  - CNF-Formula: AND of a collection of clauses

$$(X \vee Y \vee \bar{Z}) \wedge (\bar{X} \vee \bar{Y} \vee Z)$$

# 3-SAT Problem

- **Definition:**
  - Literal: a variable or its negation $x$
  - Clause: OR of a collection of literals $(X \vee Y \vee \bar{Z})$
  - CNF-Formula: AND of a collection of clauses

$$(X \vee Y \vee \bar{Z}) \wedge (\bar{X} \vee \bar{Y} \vee Z)$$

  - 3-CNF formula: any CNF-formula whose clauses have at most three literals

# 3-SAT Problem

- **Input:**
  - A 3-CNF-formula $\Phi$ with n variables and m clauses
- **Output:**
  - Decide if there is an assignment of values in $\{0,1\}^n$ to the variables so that $\Phi$ evaluates to TRUE

# 3-SAT Problem: Example

- **Examples:**
- $\Phi = (X \vee Y \vee \bar{Z}) \wedge (\bar{X} \vee \bar{Y} \vee Z)$

# 3-SAT Problem: Example

- **Examples:**
- $\Phi = (X \vee Y \vee \bar{Z}) \wedge (\bar{X} \vee \bar{Y} \vee Z)$
- Answer is YES: set (X,Y,Z) = (0,1,0)

# 3-SAT Problem: Example

- **Examples:**
- $\Phi = (X \vee Y \vee \bar{Z}) \wedge (\bar{X} \vee \bar{Y} \vee Z)$
- Answer is YES: set (X,Y,Z) = (0,1,0)
- $\Phi = (X \vee Y) \wedge (\bar{X} \vee \bar{Y}) \wedge (X \vee \bar{Y}) \wedge (\bar{X} \vee Y)$
- Answer is NO: any assignment makes (exactly) one of the clauses false

# 3-SAT Problem is in NP

# 3-SAT Problem is in NP

- We need a poly-time verifier
- Proof for verifier: if the answer is YES, give a satisfying assignment $x$ to $\Phi$
- We go over clauses one by one and make sure $x$ satisfies every clause
- This takes only $O(n+m)$ time so poly-time

# 3-SAT Problem is NP-Hard

# 3-SAT Problem is NP-Hard

- We have to show that if 3-SAT can be solved in poly-time then P=NP

- Recall our plan:
  - If 3-SAT can be solved in poly-time, then some NP-hard problem can also be solved in poly-time

# 3-SAT Problem is NP-Hard

- We have to show that if 3-SAT can be solved in poly-time then P=NP

- Recall our plan:
  - If 3-SAT can be solved in poly-time, then some NP-hard problem can also be solved in poly-time
  - At this point, the only NP-hard problem we know is Circuit-SAT

# Reduction

- Given a circuit C as input to the Circuit-SAT problem, we create a new input $\Phi$ for the 3-SAT problem such that:
  - The answer to Circuit-SAT on C is the same as the answer to 3-SAT on $\Phi$
- We then assume that 3-SAT can be solved in poly-time and we run any algorithm for that to solve this instance $\Phi$
- This then gives us a poly-time algorithm for any instance of Circuit-SAT also
- So 3-SAT should be NP-hard too

# Reduction

- Given the circuit C, define a new variable for every wire in C including the input wires

# Reduction

- For every AND gate in C:
  - Let a be the variable of output wire and b and c be the ones for input wires, so we want $a = (b \wedge c)$
  - Add the following clauses to $\Phi$:

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

# Reduction

- For every AND gate in C:
  - Let a be the variable of output wire and b and c be the ones for input wires, so we want $a = (b \wedge c)$
  - Add the following clauses to $\Phi$:

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

**Note:** in any assignment that makes $\Phi$ TRUE, we have $a = (b \wedge c)$

# Reduction

- For every OR gate in C:
  - Let a be the variable of output wire and b and c be the ones for input wires, so we want $a = (b \vee c)$
  - Add the following clauses to $\Phi$:

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

# Reduction

- For every OR gate in C:
  - Let a be the variable of output wire and b and c be the ones for input wires, so we want $a = (b \vee c)$
  - Add the following clauses to $\Phi$:

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

**Note:** in any assignment that makes $\Phi$ TRUE, we have $a = (b \vee c)$

# Reduction

- For every NOT gate in C:
  - Let a be the variable of output wire and b be the one for input wire, so we want $a = \bar{b}$
  - Add the following clauses to $\Phi$:

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

# Reduction

- For every NOT gate in C:
  - Let a be the variable of output wire and b be the one for input wire, so we want $a = \bar{b}$
  - Add the following clauses to $\Phi$:

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

**Note:** in any assignment that makes $\Phi$ TRUE, we have $a = \bar{b}$

# Reduction

- For the output wire in C:
  - Let a be the variable of output wire
  - Add the following clauses to $\Phi$:

$$(a)$$

# Reduction

- For the output wire in C:
  - Let a be the variable of output wire
  - Add the following clauses to $\Phi$:

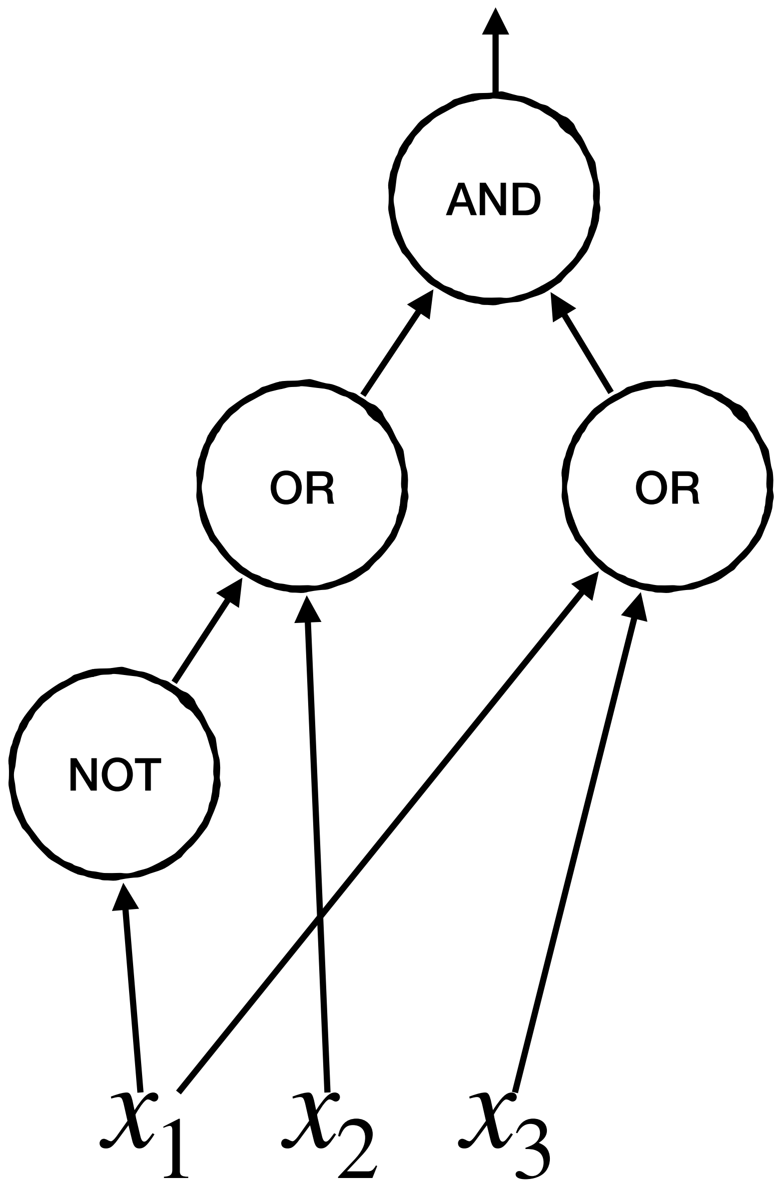
$$(a)$$

**Note:** in any assignment that makes $\Phi$ TRUE, we have $a = 1$

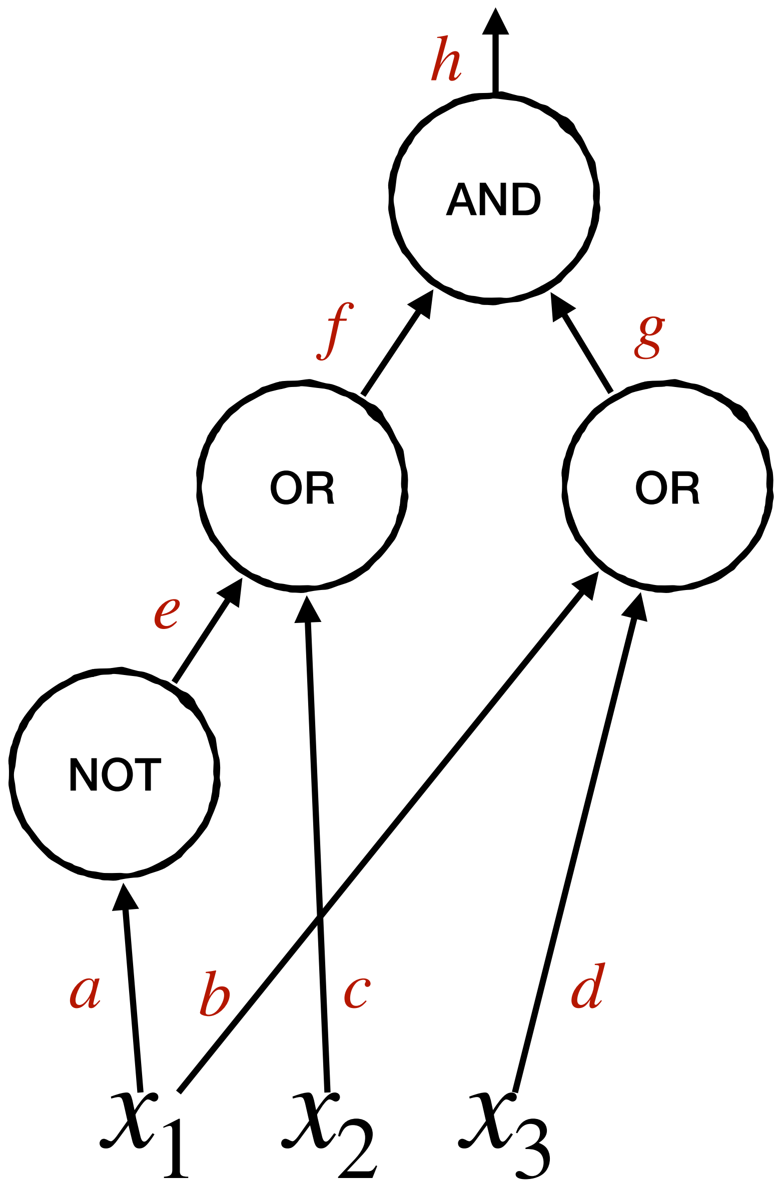
# Reduction

- After creating $\Phi$ this way, we run any algorithm for 3-SAT on $\Phi$
- Return the same answer to the original Circuit-SAT problem

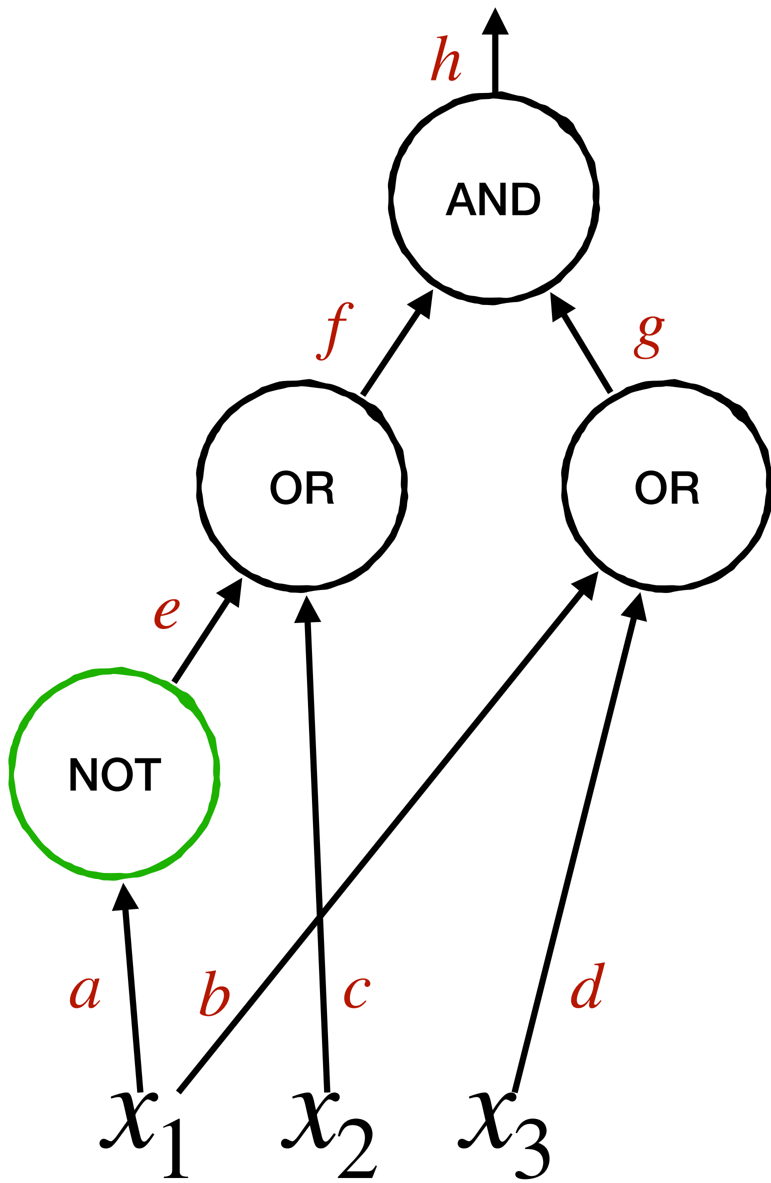
# Reduction: Example



# Reduction: Example

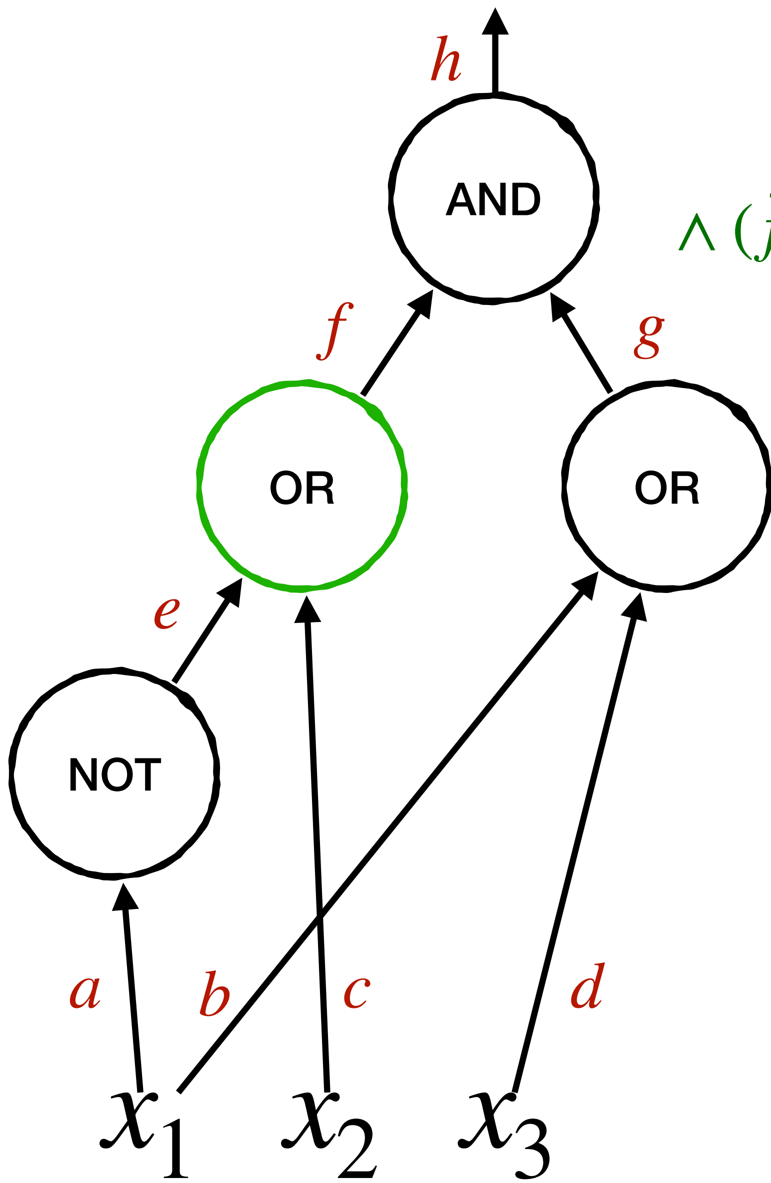


# Reduction: Example




$(a \vee e) \wedge (\bar{a} \vee \bar{e})$

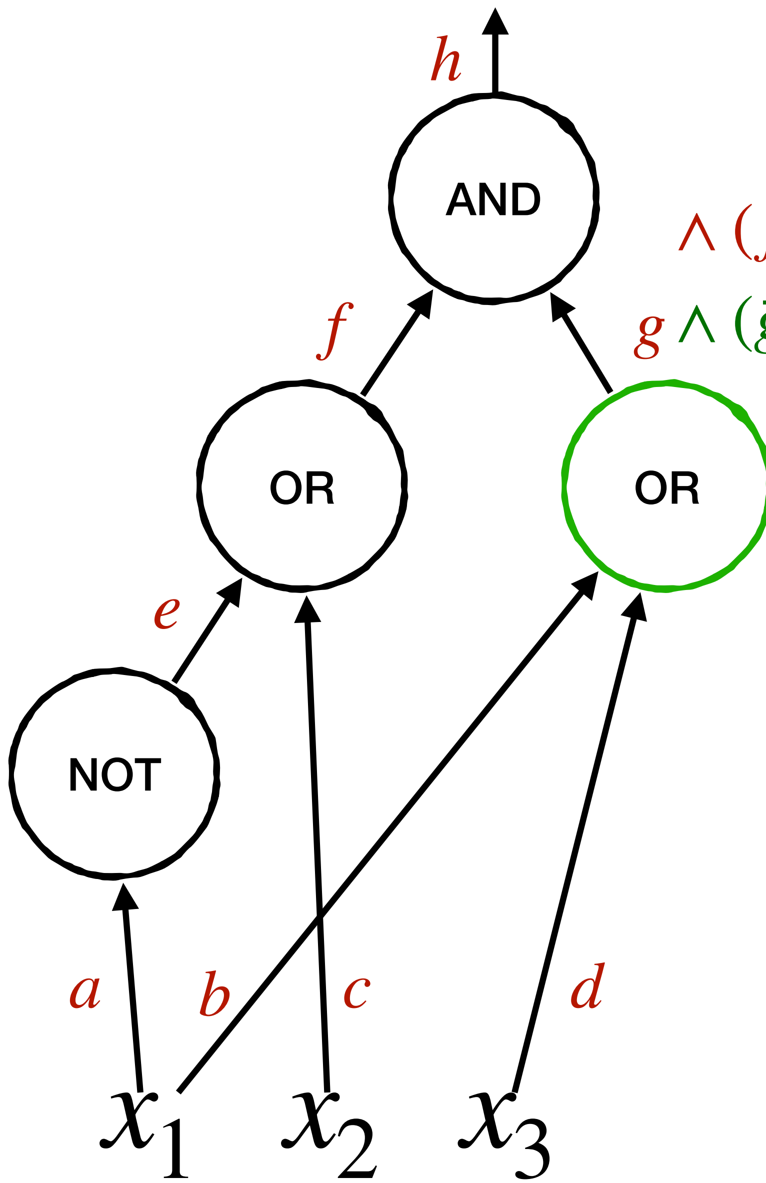
# Reduction: Example




$$(a \vee e) \wedge (\bar{a} \vee \bar{e})$$

$$\wedge\, (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c})$$

# Reduction: Example

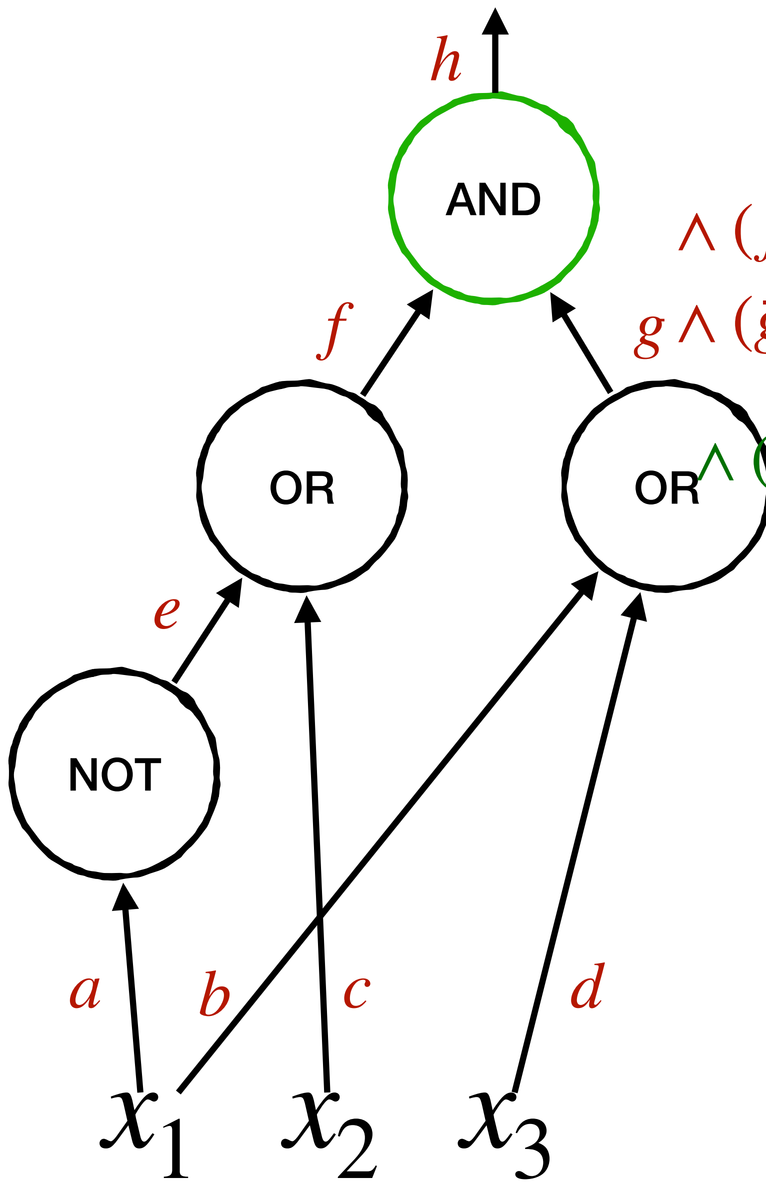



$$(a \vee e) \wedge (\bar{a} \vee \bar{e})$$

$$\wedge\, (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c})$$

$$g \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$

# Reduction: Example



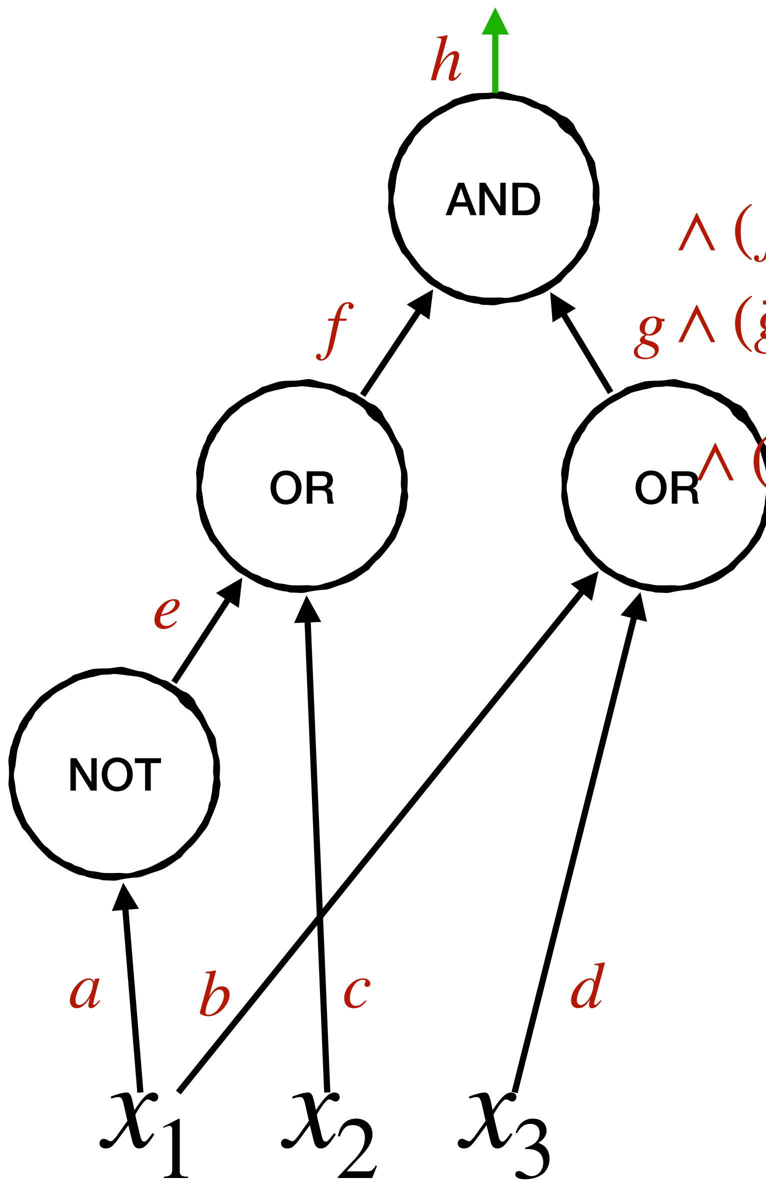

$$(a \vee e) \wedge (\bar{a} \vee \bar{e})$$

$$\wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c})$$

$$g \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$

$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g)$$

# Reduction: Example




$$(a \vee e) \wedge (\bar{a} \vee \bar{e})$$

$$\wedge\ (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c})$$

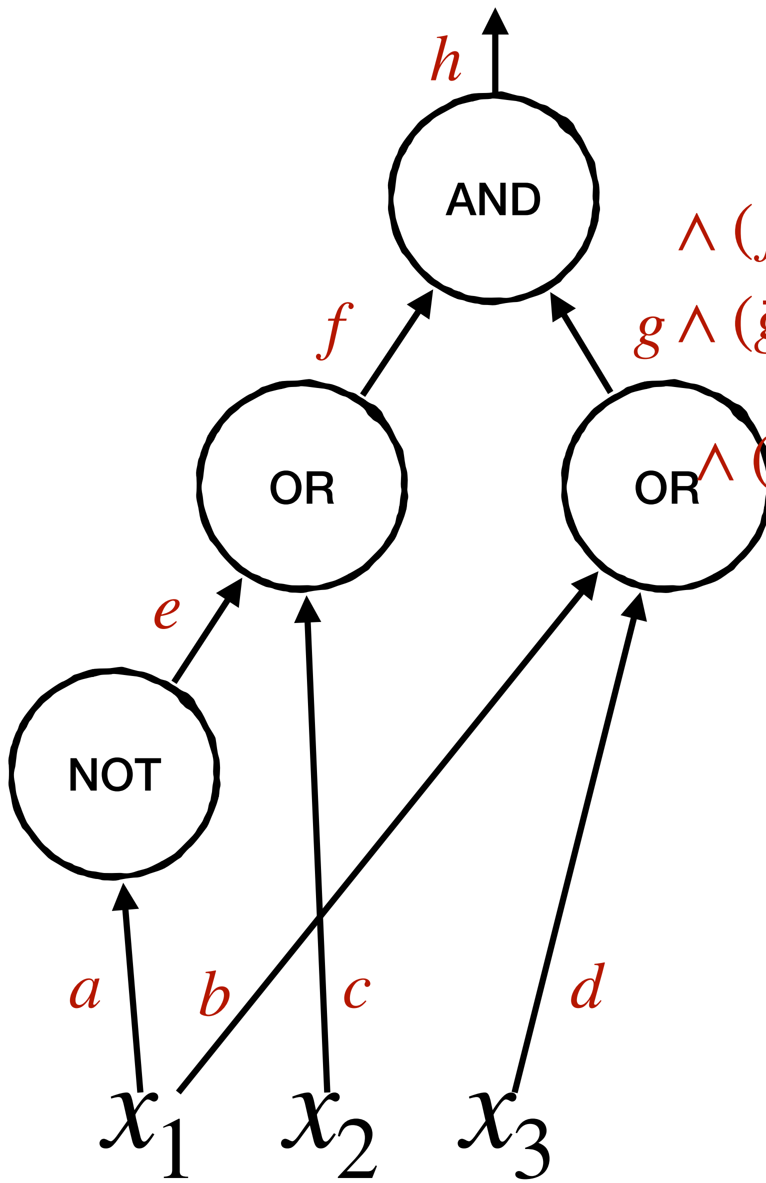
$$g \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$

$$\wedge\ (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g)$$

$$\wedge\ (h)$$

# Reduction: Example

$\Phi$




$$(a \vee e) \wedge (\bar{a} \vee \bar{e})$$

$$\wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c})$$

$$g \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$

$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g)$$

$$\wedge (h)$$

**$\Phi$ is going to be satisfiable if and only if C is satisfiable**

# Reduction: Proof of Correctness

- **Part one:** If $\Phi$ is satisfiable then C is also satisfiable
- Pick a satisfying assignment y to $\Phi$
- Let x be the assignment of variables to input wires in y
- C(x) must be TRUE so C is also satisfiable

# Reduction: Proof of Correctness

- **Part two:** If C is satisfiable then $\Phi$ is also satisfiable
- Pick a satisfying assignment x to C
- Let y be the assignment to all variables of $\Phi$ corresponding to the values of all wires in C(x)
- $\Phi(y)$ must be TRUE so $\Phi$ is also satisfiable

# Reduction: Runtime Analysis

- **IF** we have a poly-time algorithm for 3-SAT we also get a poly-time reduction this way.
- Size of $\Phi$ is just a constant factor larger than the input circuit (at most three clause per wire)
- Creating $\Phi$ takes time linear in the size of C

# Reduction: Conclusion

- So if 3-SAT can be solved in poly-time Circuit-SAT can also be solved in poly-time

- This means if 3-SAT can be solved in poly-time then P=NP because Circuit-SAT is NP-hard

- So 3-SAT is also NP-hard

# Reduction: Conclusion

- So if 3-SAT can be solved in poly-time Circuit-SAT can also be solved in poly-time

- This means if 3-SAT can be solved in poly-time then P=NP because Circuit-SAT is NP-hard

- So 3-SAT is also NP-hard

- Since 3-SAT is also in NP, it is actually NP-complete

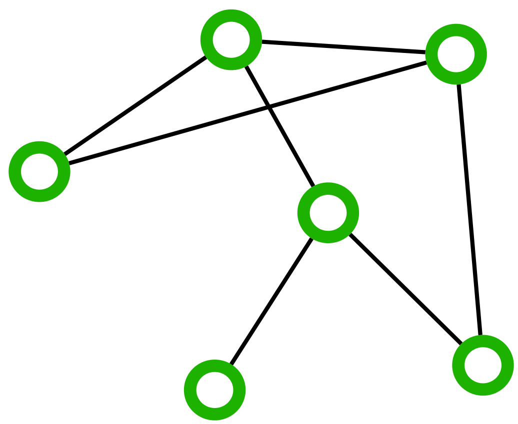
# Example 2: Maximum Independent Set is NP-Hard

# Independent Set

- Given an undirected graph $G=(V,E)$, an independent set is any set of vertices with no edges between them

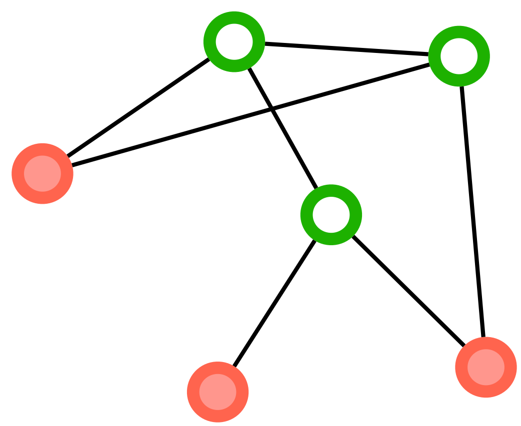
# Independent Set

- Given an undirected graph G=(V,E), an independent set is any set of vertices with no edges between them



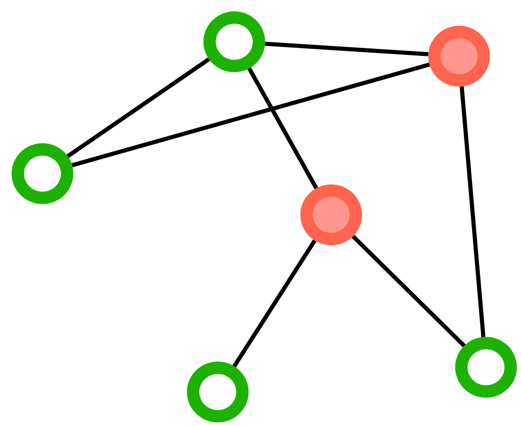
# Independent Set

- Given an undirected graph G=(V,E), an independent set is any set of vertices with no edges between them



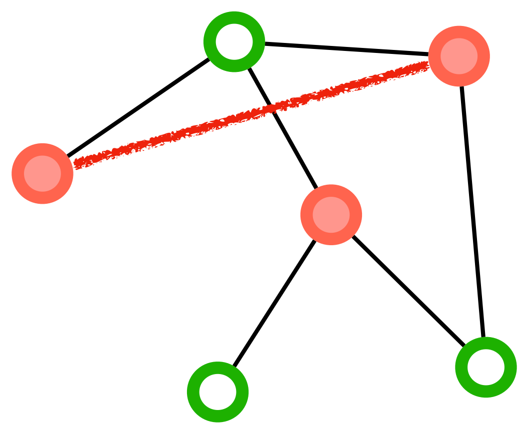
# Independent Set

- Given an undirected graph G=(V,E), an independent set is any set of vertices with no edges between them



# Independent Set

- Given an undirected graph G=(V,E), an independent set is any set of vertices with no edges between them



# Maximum Independent Set Problem

- **Input:**
  - An undirected graph $G=(V,E)$
- **Output:**
  - Size of the largest independent set in $G$

- For simplicity, we are going to call this problem **MaxIndSet**

# MaxIndSet

- Is **MaxIndSet** in NP?

# MaxIndSet

- Is **MaxIndSet** in NP?
- No because it is NOT a decision problem

# MaxIndSet is NP-hard

- We are going to show that it is NP-hard
- This requires proving if **MaxIndSet** can be solved in poly-time, then P=NP
- Using reductions, this requires showing that a poly-time algorithm for **MaxIndSet** can solve another NP-hard problem in poly-time

# MaxIndSet is NP-hard

- We are going to show that it is NP-hard
- This requires proving if **MaxIndSet** can be solved in poly-time, then P=NP
- Using reductions, this requires showing that a poly-time algorithm for **MaxIndSet** can solve another NP-hard problem in poly-time
- We use 3-SAT for this purpose

# MaxIndSet: Reduction From 3-SAT

- Given a formula $\Phi$ as input to the 3-SAT problem, we create the following graph:

# MaxIndSet: Reduction From 3-SAT

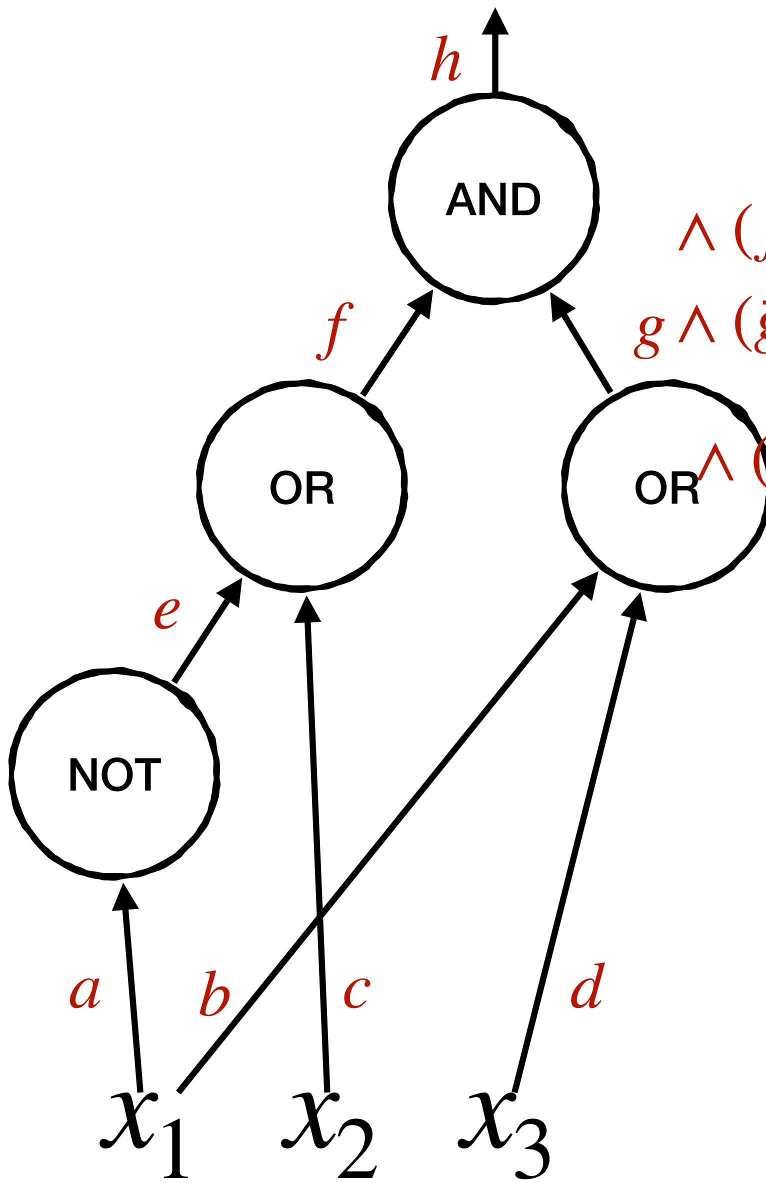
- Given a formula $\Phi$ as input to the 3-SAT problem, we create the following graph:
  - For any clause in $\Phi$, we add one vertex per literal of the clause
  - We connect all vertices in a clause together
  - We connect a vertex u to a vertex v if the literal of u is the negation of the literal of v

# MaxIndSet: Reduction From 3-SAT

- Given a formula $\Phi$ as input to the 3-SAT problem, we create the following graph:
  - For any clause in $\Phi$, we add one vertex per literal of the clause
  - We connect all vertices in a clause together
  - We connect a vertex u to a vertex v if the literal of u is the negation of the literal of v
- After creating G, we run any algorithm for **MaxIndSet** on G
- If size of the returned independent set is equal to the number of clauses, we return $\Phi$ is satisfiable, and otherwise it is not.

# Reduction: Example

$\Phi$




$$(a \vee e) \wedge (\bar{a} \vee \bar{e})$$

$$\wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c})$$

$$g \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$

$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g)$$

$$\wedge (h)$$

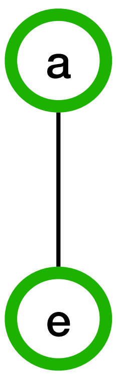
**$\Phi$ is going to be satisfiable**
**if and only if**
**C is satisfiable**

# Reduction: Example

$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$

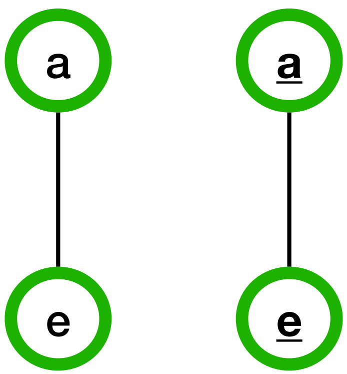
# Reduction: Example

$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



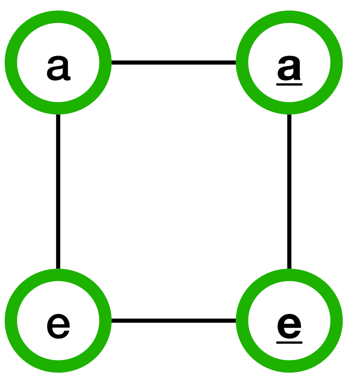
# Reduction: Example

$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



# Reduction: Example

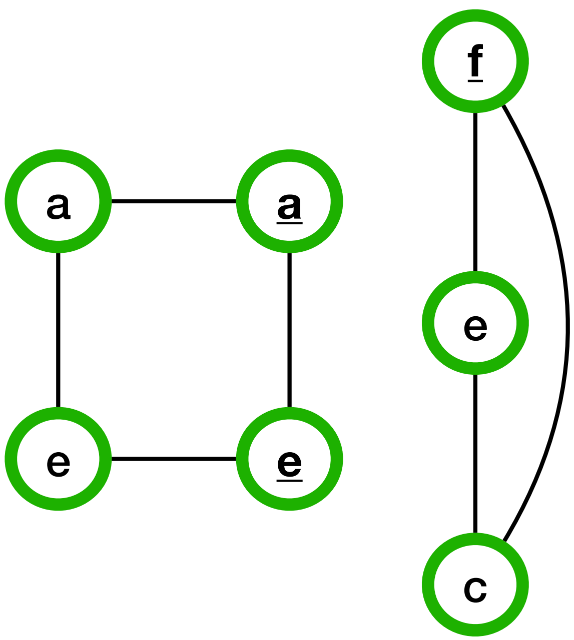
$$\Phi = (a \lor e) \land (\bar{a} \lor \bar{e}) \land (\bar{f} \lor e \lor c) \land (f \lor \bar{e}) \land (f \lor \bar{c}) \land (\bar{g} \lor b \lor d) \land (g \lor \bar{b}) \land (g \lor \bar{d})$$
$$\land (h \lor \bar{f} \lor \bar{g}) \land (\bar{h} \lor f) \land (\bar{h} \lor g) \land (h)$$



# Reduction: Example

$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$

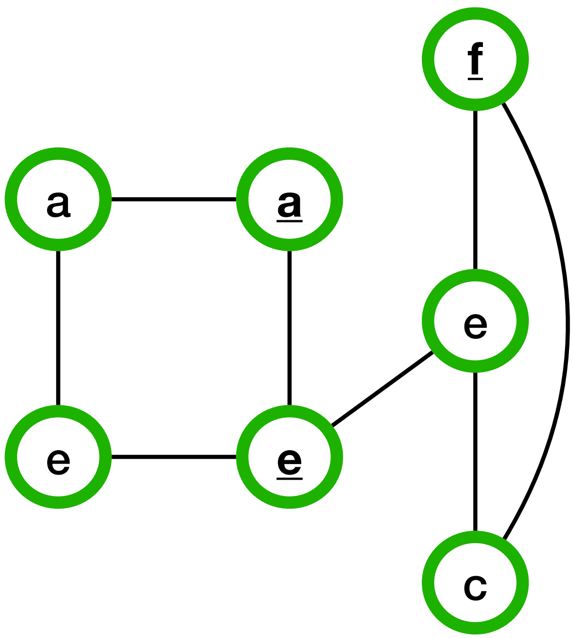
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



# Reduction: Example

$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$

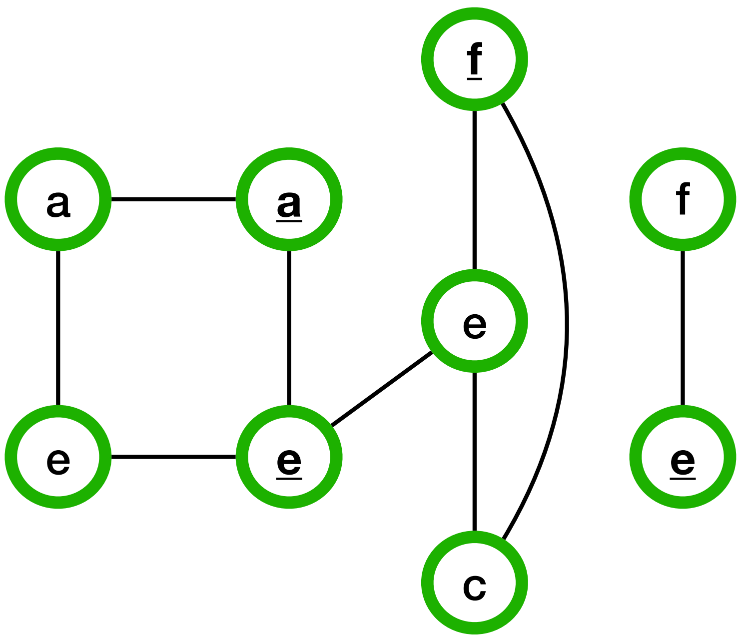
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



# Reduction: Example

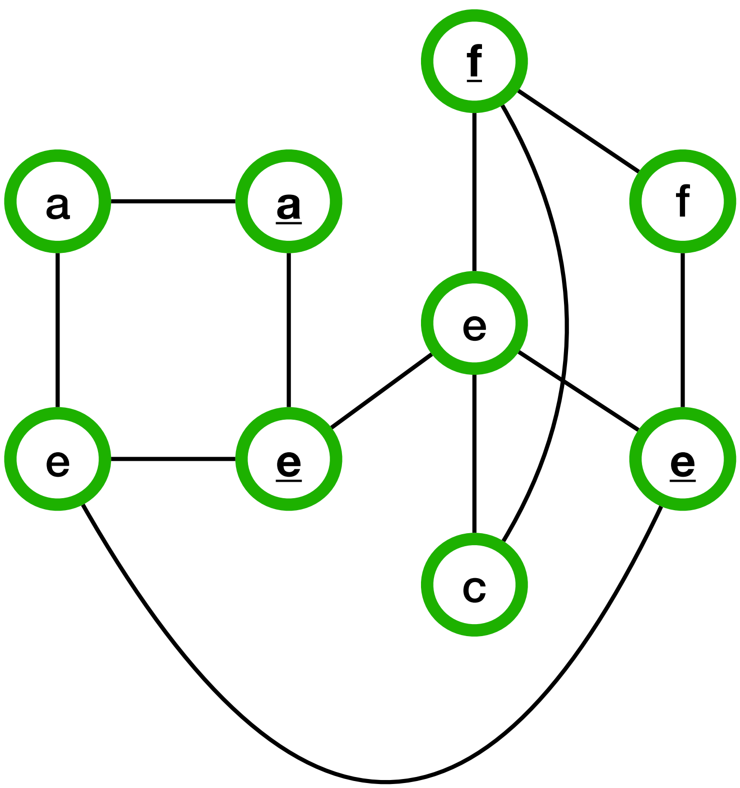
$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$

$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



# Reduction: Example

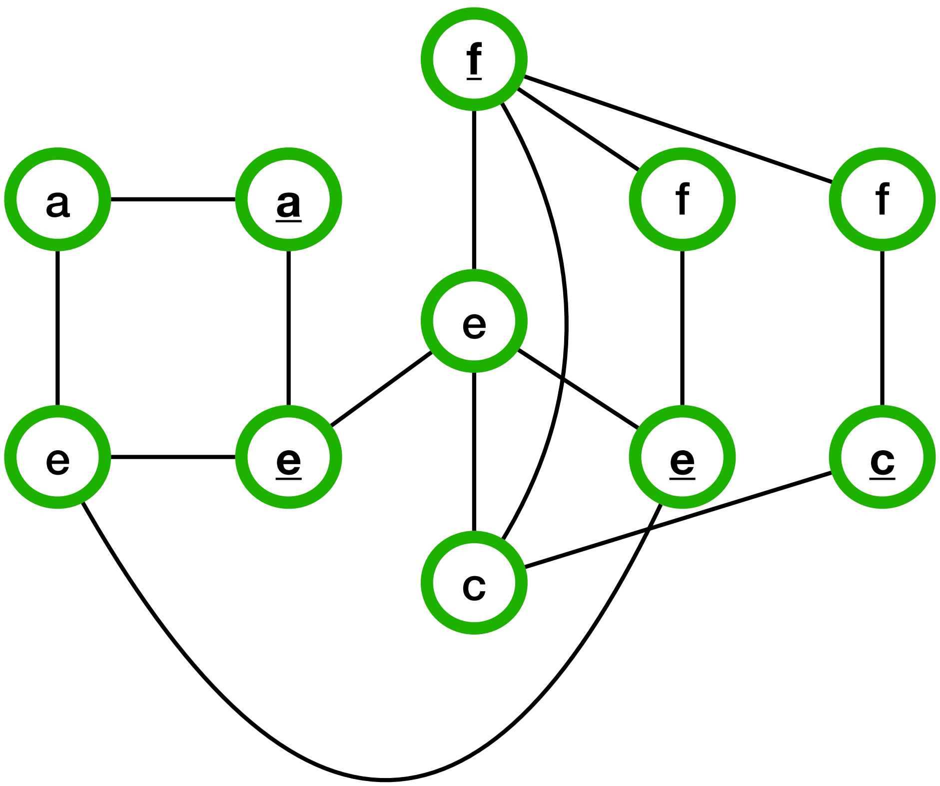
$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



# Reduction: Example

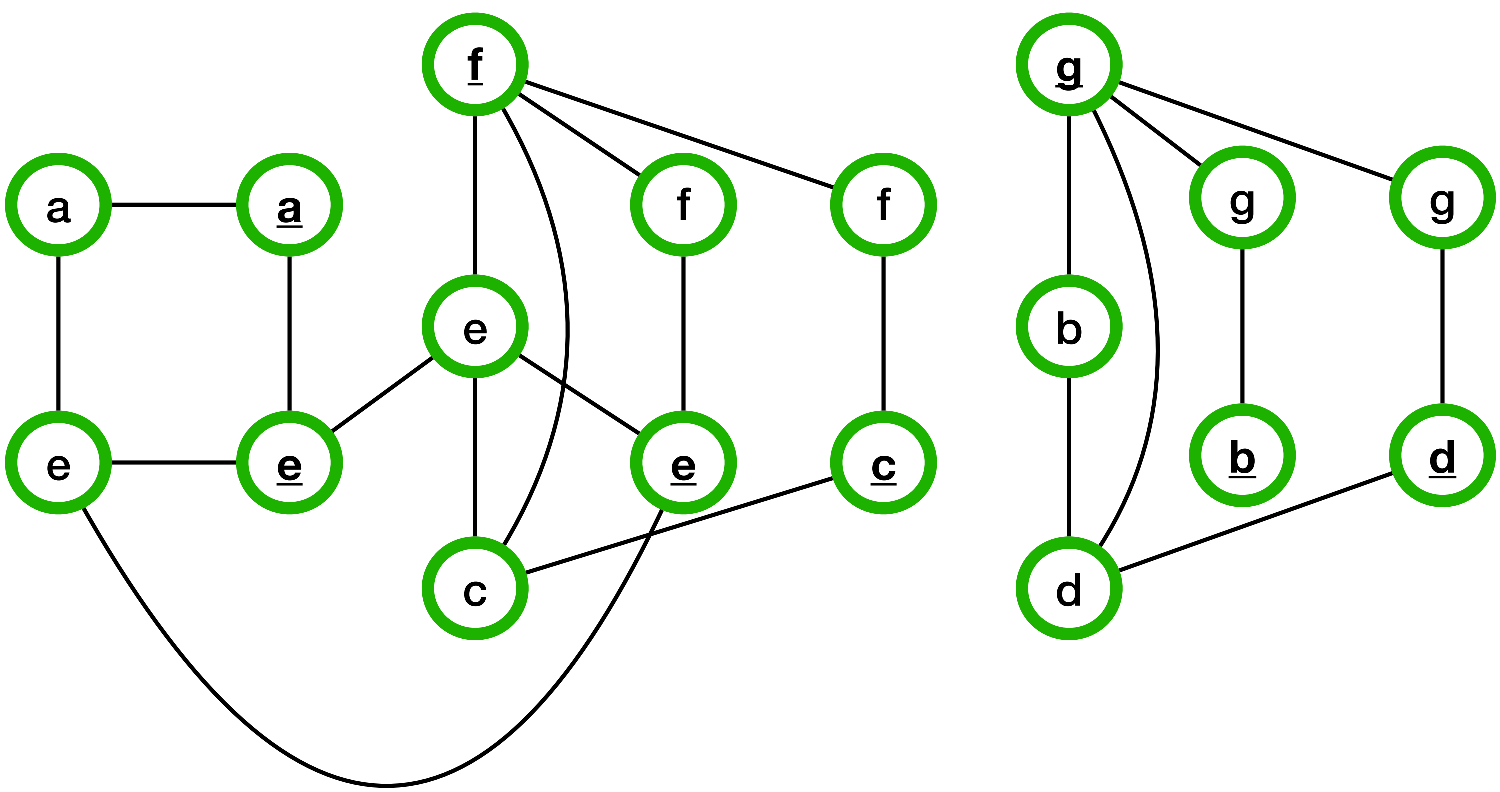
$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$

$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



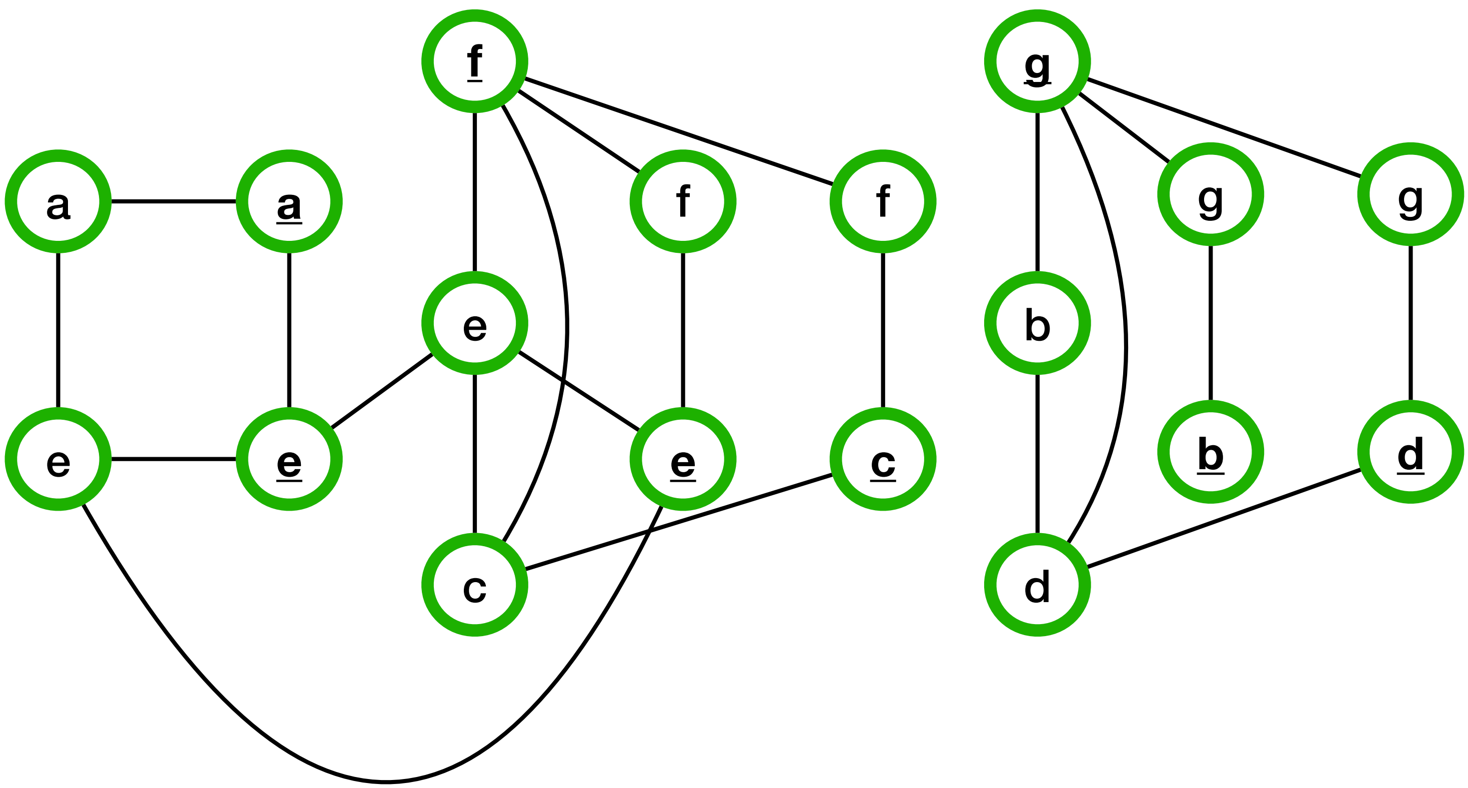
# Reduction: Example

$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$
$$\wedge\, (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



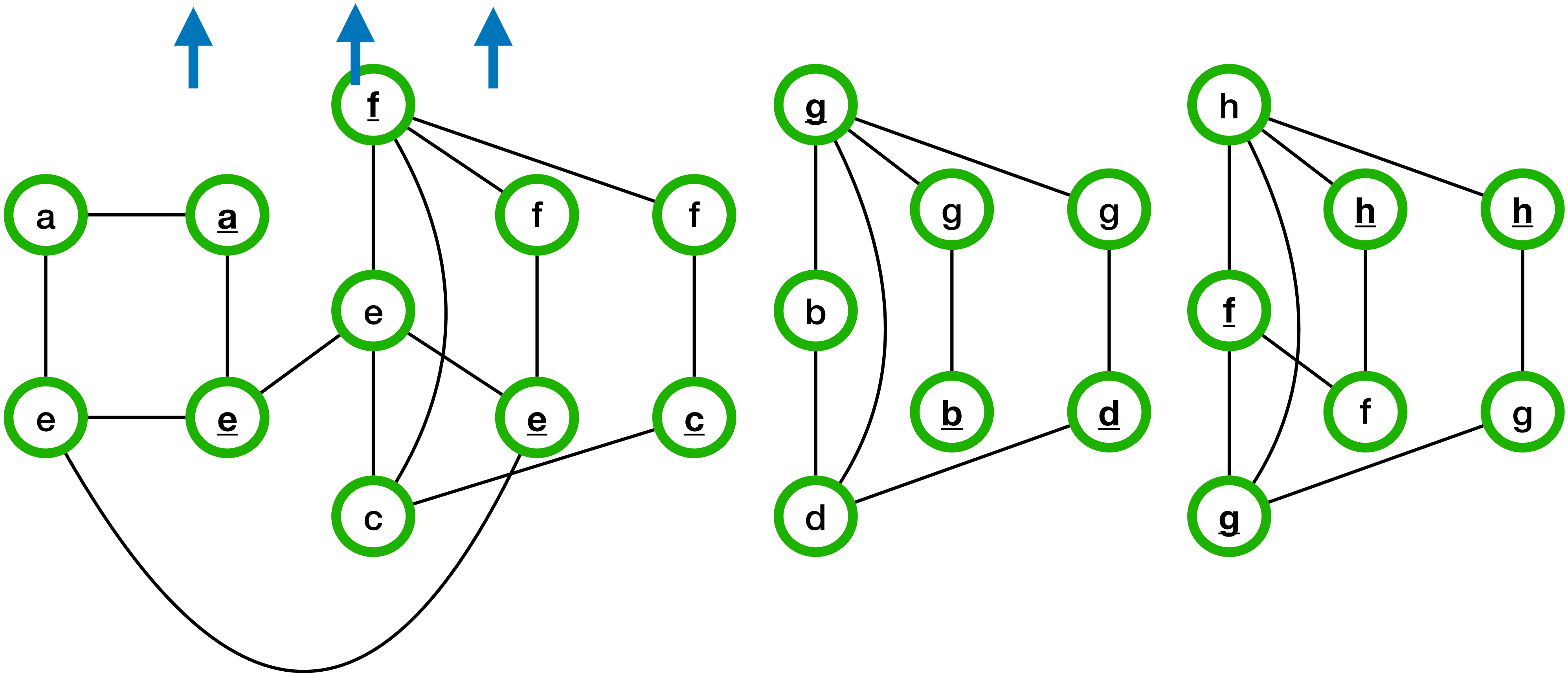
# Reduction: Example

$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



# Reduction: Example

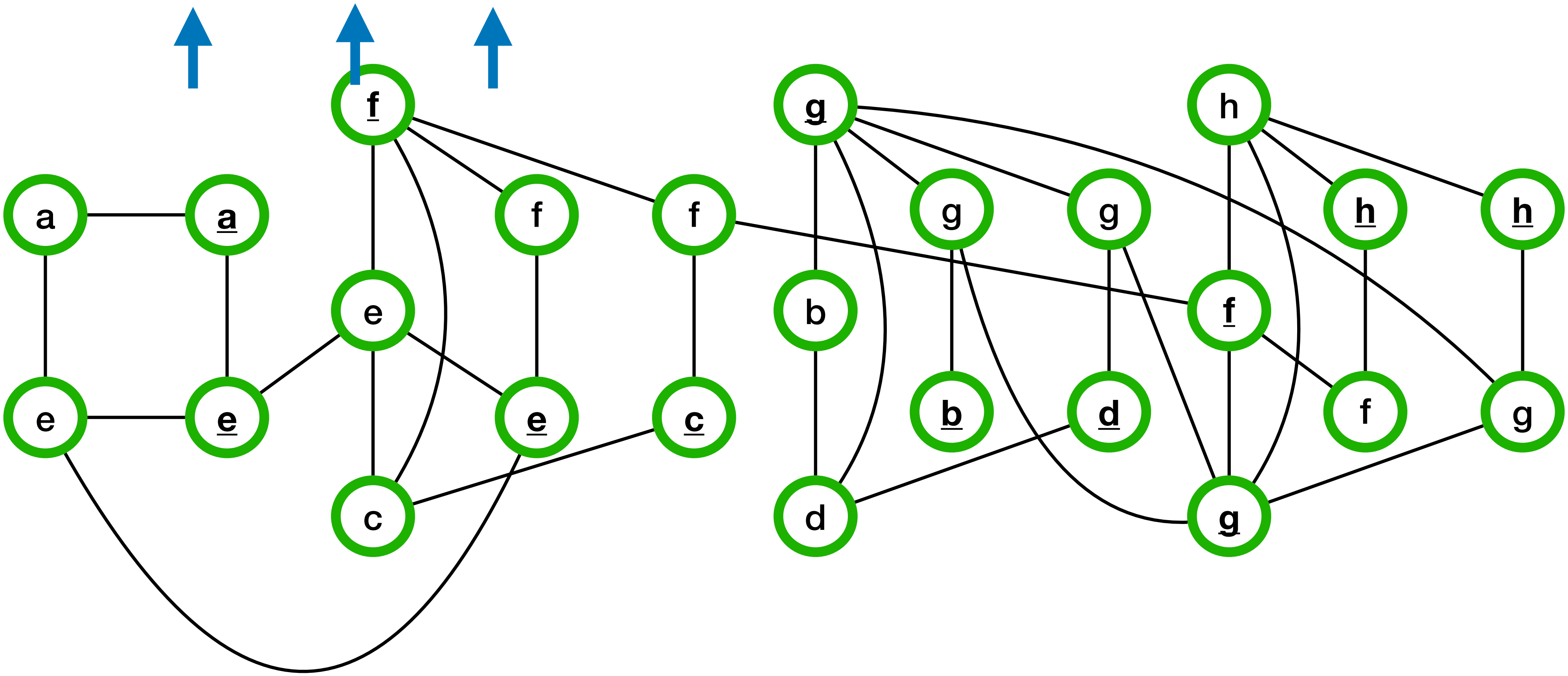
$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



# Reduction: Example

$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$

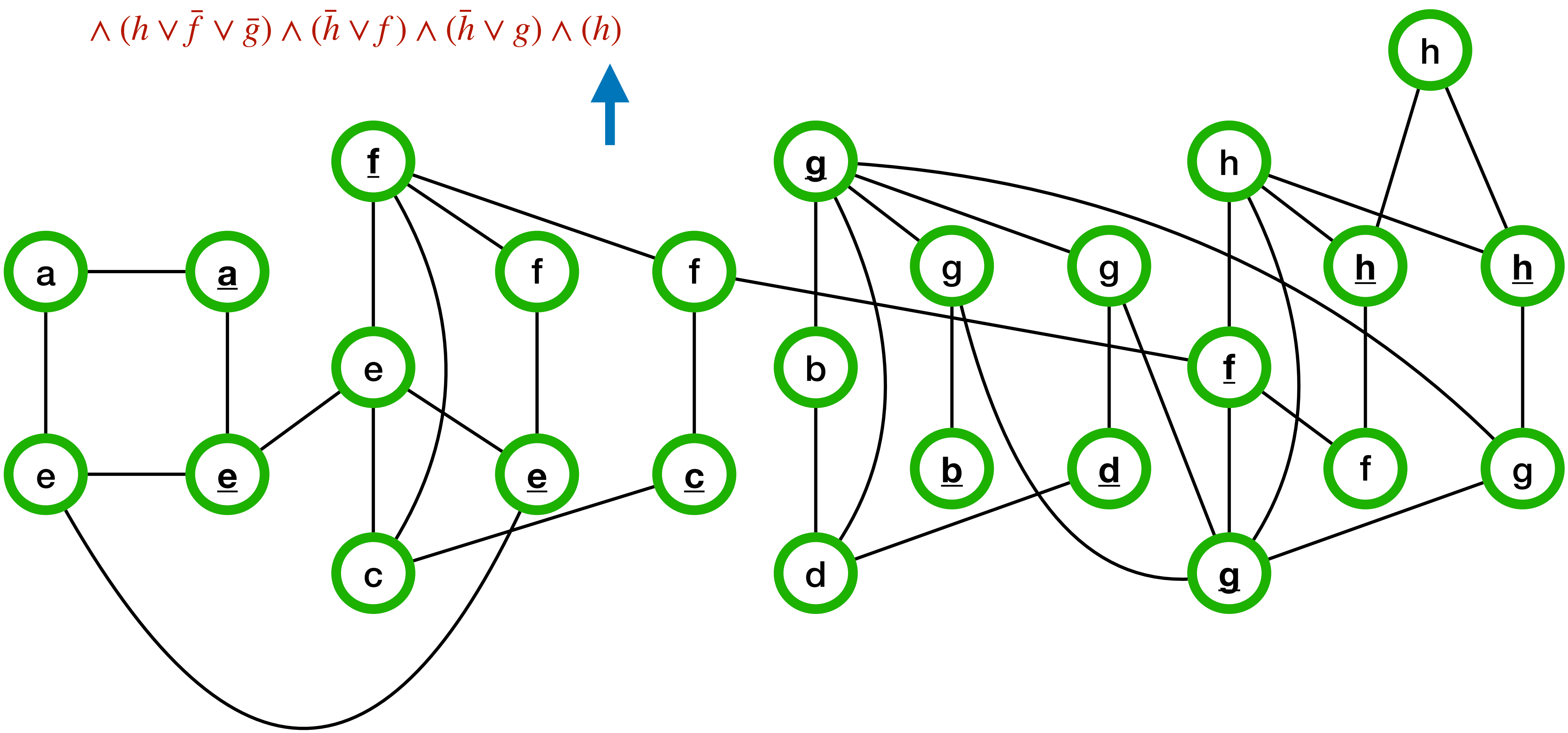
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



# Reduction: Example

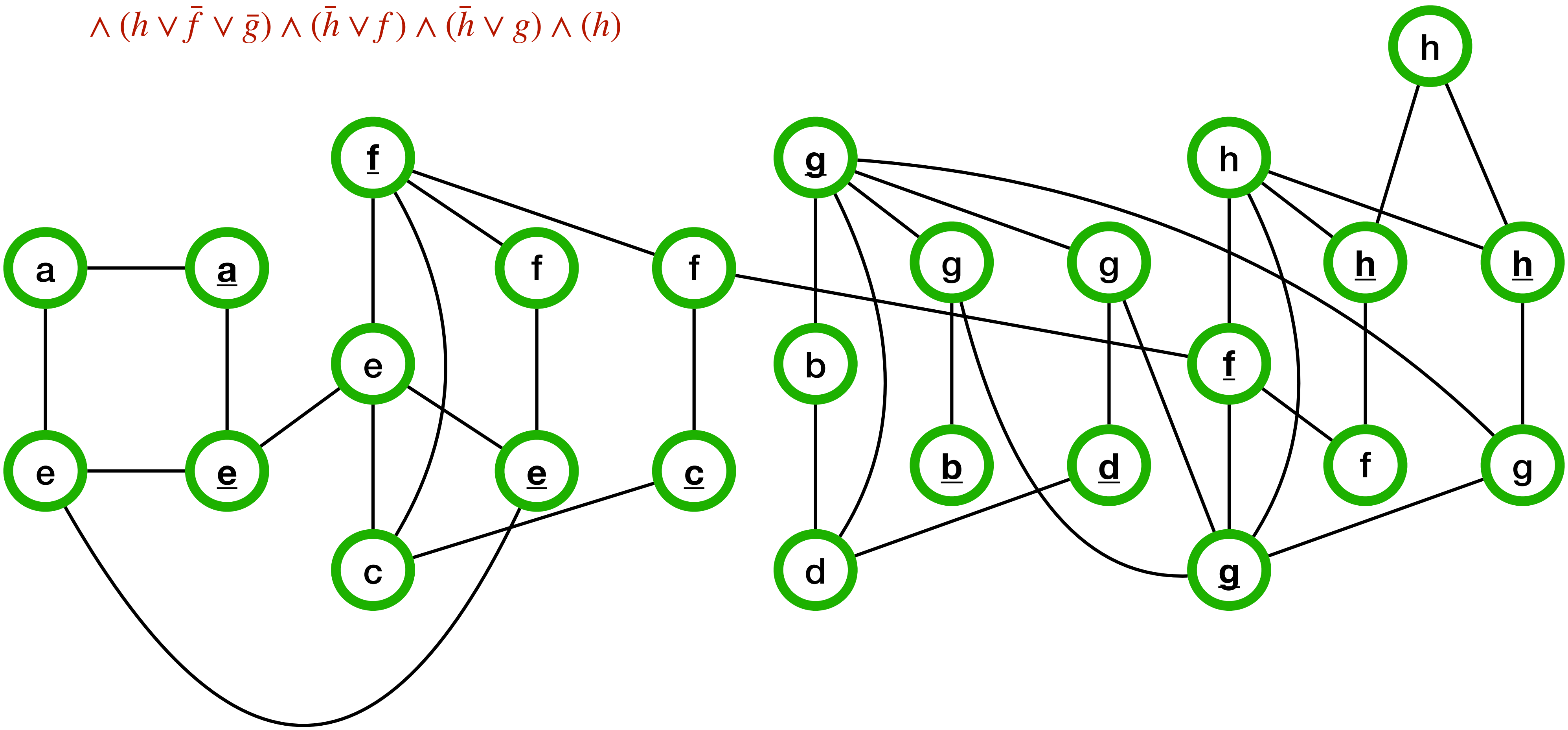
$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$

$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



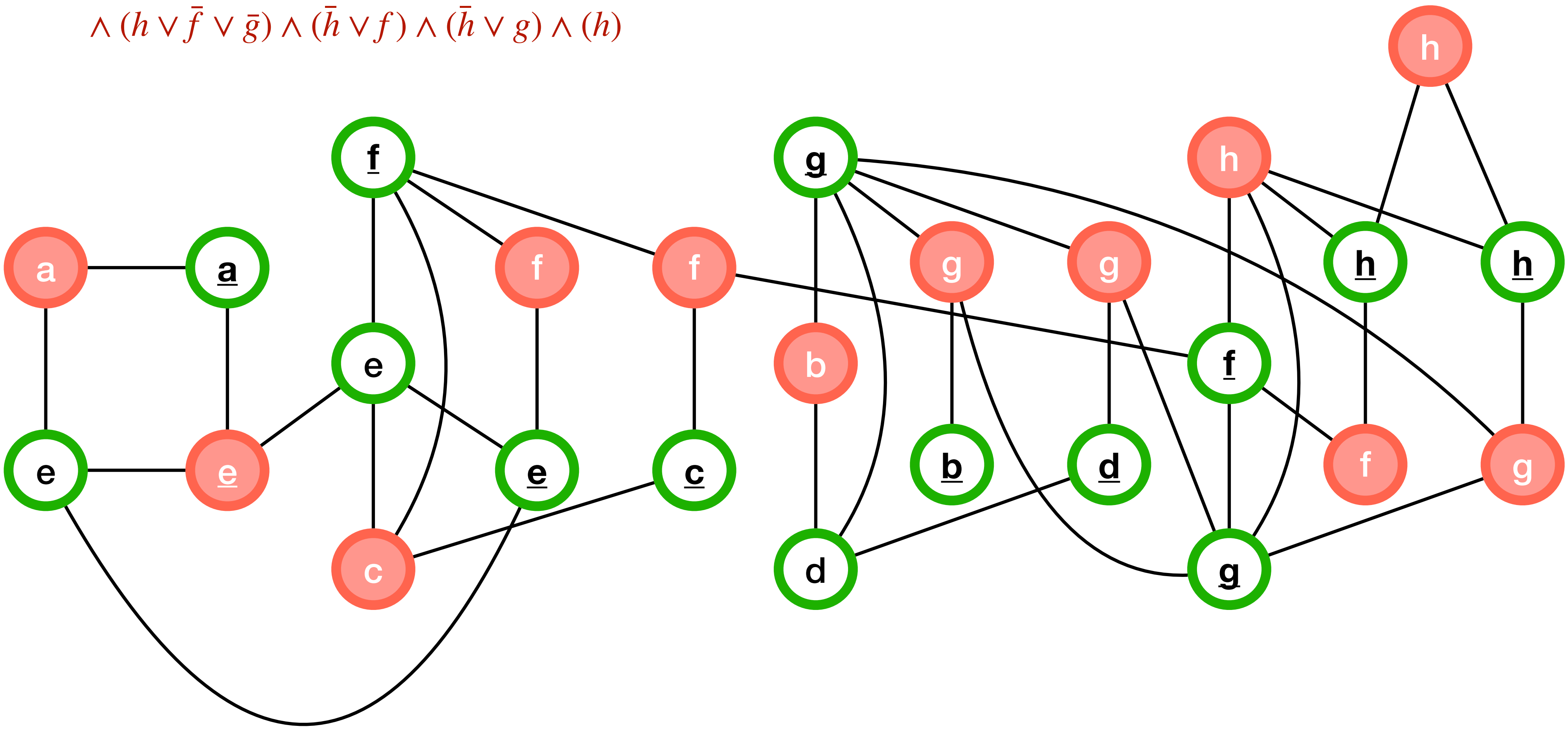
# Reduction: Example

$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



# Reduction: Example

$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



# Reduction: Proof of Correctness

- **Part one:** If maximum independent set size in G is equal to the number of clauses, then $\Phi$ is satisfiable
- Pick a largest independent set S in G
- There is exactly one vertex per each clause
- No literal and its negation can be picked simultaneously in S
- Define an assignment x of $\Phi$: each positive literal chosen in S is set to 1 and each negative literal is set to 0 (remaining variables arbitrary)
- $\Phi(x)$ must be TRUE so $\Phi$ is also satisfiable

# Reduction: Proof of Correctness

- **Part two:** If $\Phi$ is satisfiable, then maximum independent set size in G is equal to the number of clauses.
- Pick a satisfying assignment x of $\Phi$
- Define a set T of vertices: from each clause, pick one literal-vertex whose literal is 1 in x
- T is an independent set because we only pick one vertex per clause and we never pick a literal and its negation
- So T is an independent set with size equal to the number of clauses
- There is no larger independent set in G as we can only pick one vertex per clause

# Reduction: Runtime Analysis

- **IF** we have a poly-time algorithm for **MaxIndSet** we also get a poly-time reduction this way.
- Size of $G$ is just a constant factor larger than the input formula (at most three vertices per clause)
- Creating G takes time linear in the size of $\Phi$

# Reduction: Conclusion

- So if **MaxIndSet** can be solved in poly-time 3-SAT can also be solved in poly-time
- This means if **MaxIndSet** can be solved in poly-time then P=NP because 3-SAT is NP-hard
- So **MaxIndSet** is also NP-hard

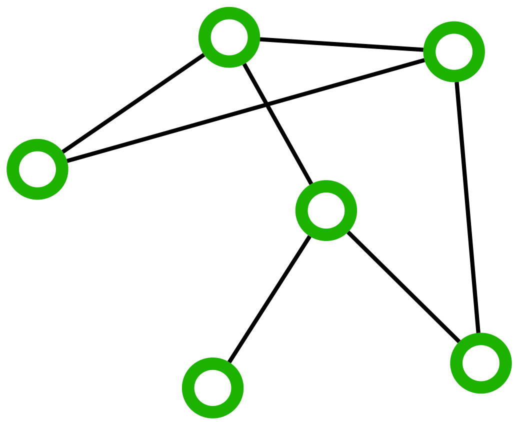
# Example 3: Minimum Vertex Cover is NP-Hard

# Vertex Cover

- Given an undirected graph G=(V,E), a vertex cover is any set of vertices such that any edge has at least one endpoint in G

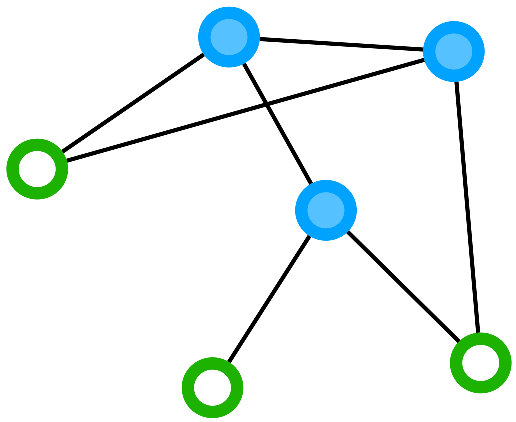
# Vertex Cover

- Given an undirected graph G=(V,E), a vertex cover is any set of vertices such that any edge has at least one endpoint in G



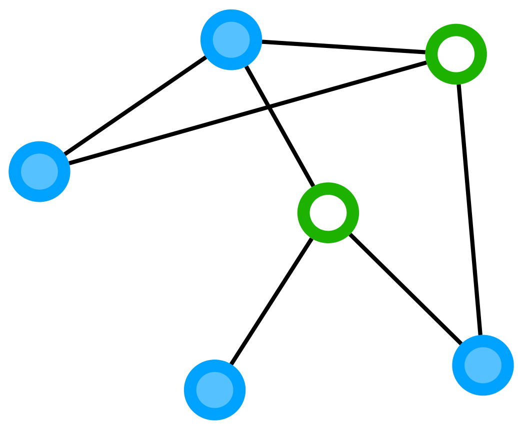
# Vertex Cover

- Given an undirected graph G=(V,E), a vertex cover is any set of vertices such that any edge has at least one endpoint in G



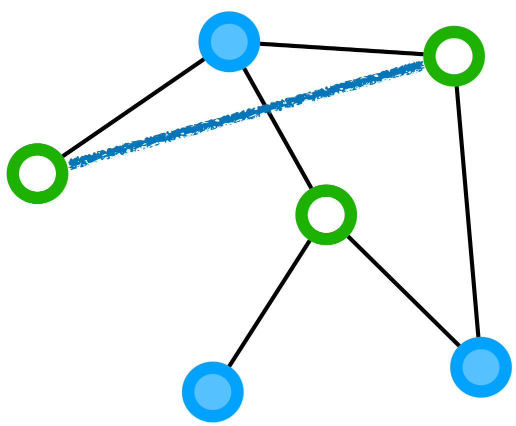
# Vertex Cover

- Given an undirected graph G=(V,E), a vertex cover is any set of vertices such that any edge has at least one endpoint in G



# Vertex Cover

- Given an undirected graph G=(V,E), a vertex cover is any set of vertices such that any edge has at least one endpoint in G



# Minimum Vertex Cover Problem

- **Input:**
  - An undirected graph $G=(V,E)$
- **Output:**
  - Size of the smallest vertex cover in $G$

- For simplicity, we are going to call this problem **MinVC**

# MinVC

- Is **MinVC** in NP?

# MinVC

- Is **MinVC** in NP?
- No because it is NOT a decision problem

# MinVC is NP-hard

- We are going to show that it is NP-hard

- This requires proving if **MinVC** can be solved in poly-time, then P=NP

- Using reductions, this requires showing that a poly-time algorithm for **MinVC** can solve another NP-hard problem in poly-time

# MinVC is NP-hard

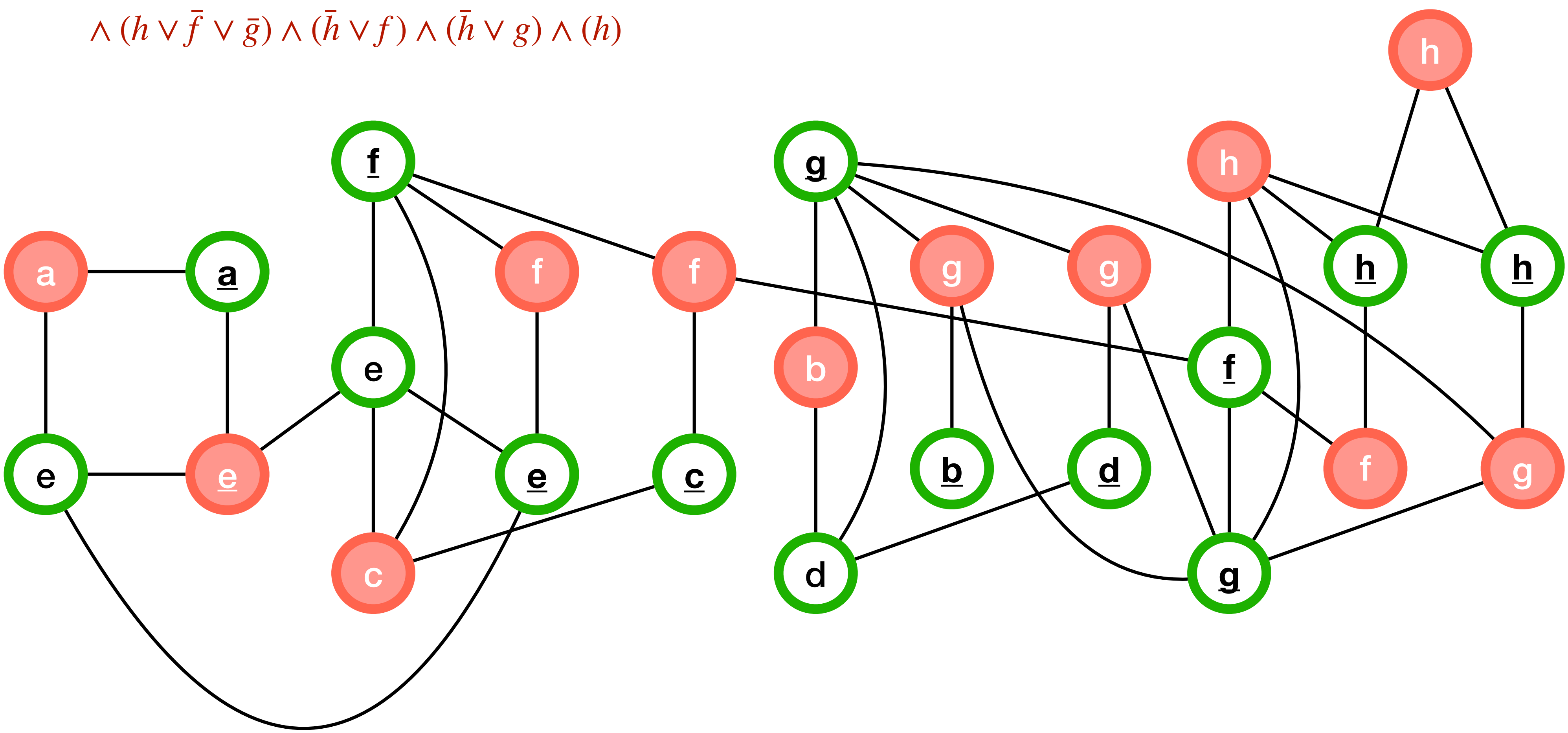
- We are going to show that it is NP-hard
- This requires proving if **MinVC** can be solved in poly-time, then P=NP
- Using reductions, this requires showing that a poly-time algorithm for **MinVC** can solve another NP-hard problem in poly-time
- We use **MaxIndSet** for this purpose

# MinVC: Reduction From MaxIndSet

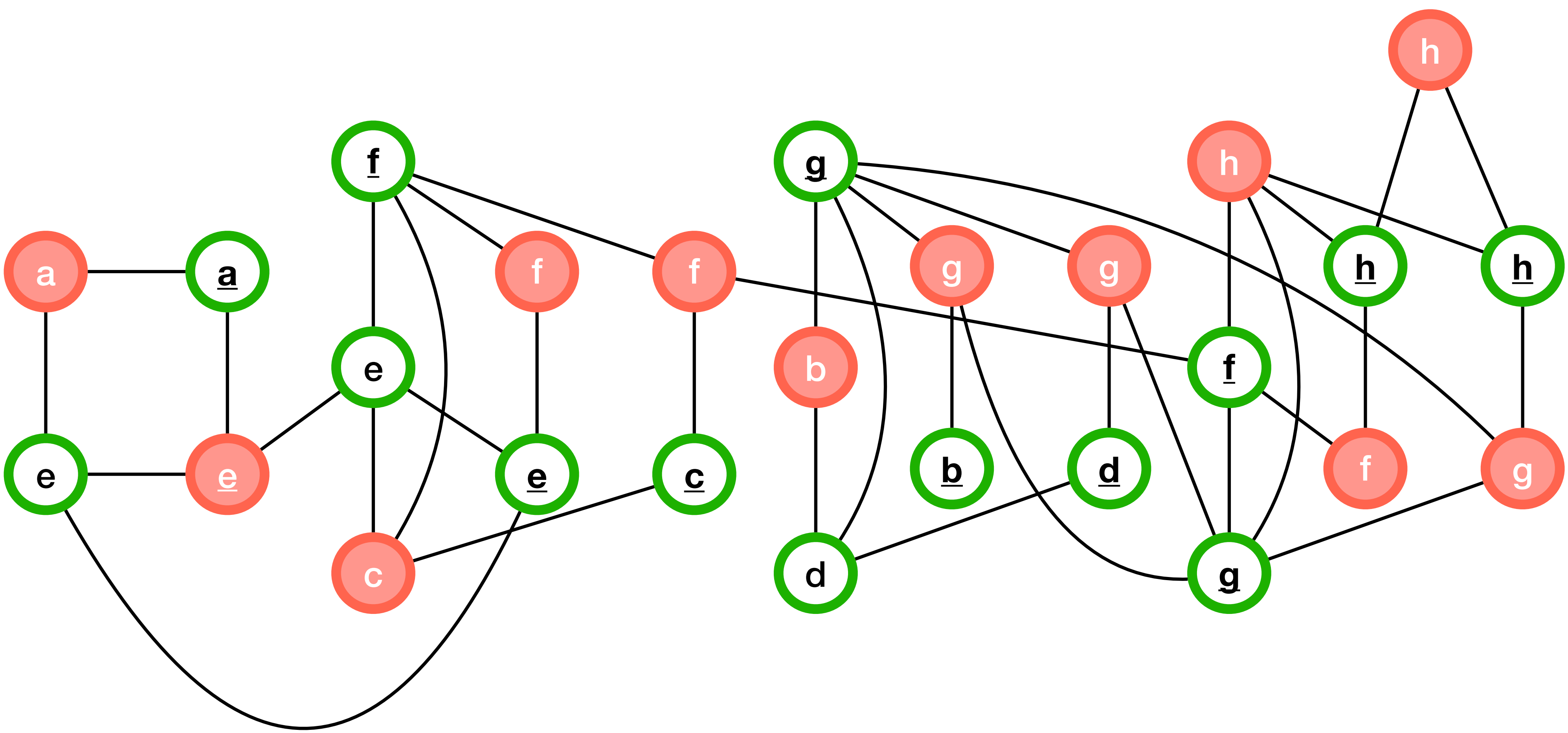
- Given a graph G as input to the **MaxIndSet** problem:
  - Run any algorithm for **MinVC** on G to get k = size of a minimum vertex cover in G
  - Return n-k as the answer to **MaxIndSet**

# Reduction: Example

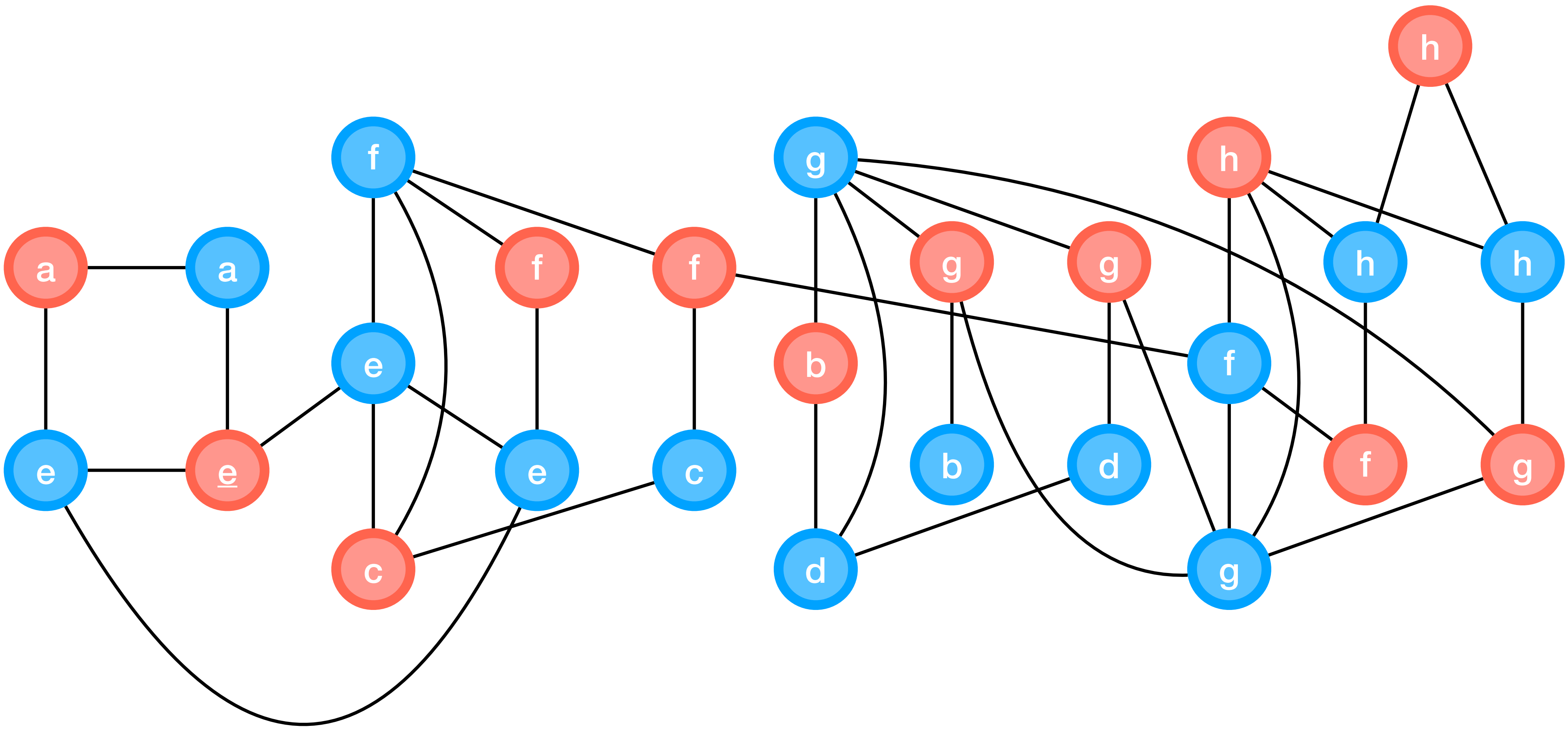
$$\Phi = (a \vee e) \wedge (\bar{a} \vee \bar{e}) \wedge (\bar{f} \vee e \vee c) \wedge (f \vee \bar{e}) \wedge (f \vee \bar{c}) \wedge (\bar{g} \vee b \vee d) \wedge (g \vee \bar{b}) \wedge (g \vee \bar{d})$$
$$\wedge (h \vee \bar{f} \vee \bar{g}) \wedge (\bar{h} \vee f) \wedge (\bar{h} \vee g) \wedge (h)$$



# Reduction: Example



# Reduction: Example



# Reduction: Proof of Correctness

- In any graph G
  - T is an independent set if and only if S=V-T is a vertex cover

# Reduction: Proof of Correctness

- In any graph G
  - T is an independent set if and only if S=V-T is a vertex cover
- **Part one:** If T is an independent set, then S=V-T is a vertex cover:
  - Suppose not, then there is an edge with no endpoint in S
  - So both endpoints in T, a contradiction

# Reduction: Proof of Correctness

- In any graph G
  - T is an independent set if and only if S=V-T is a vertex cover
- **Part two:** If S is a vertex cover, then T=V-S is an independent set:
  - Suppose not, then there is an edge with both endpoints in T
  - So no endpoints in S, a contradiction

# Reduction: Proof of Correctness

- In any graph G
  - T is an independent set if and only if S=V-T is a vertex cover

- So T is a maximum independent set in G if and only if S=V-T is a minimum vertex cover

# Reduction: Proof of Correctness

- In any graph G
  - T is an independent set if and only if S=V-T is a vertex cover

- So T is a maximum independent set in G if and only if S=V-T is a minimum vertex cover

- So answer to **MaxIndSet** is equal to n minus the answer to **MinVC**

# Reduction: Runtime Analysis

- **IF** we have a poly-time algorithm for **MinVC** we also get a poly-time reduction this way.

# Reduction: Conclusion

- So if **MinVC** can be solved in poly-time **MaxIndSet** can also be solved in poly-time
- This means if **MinVC** can be solved in poly-time then P=NP because **MaxIndSet** is NP-hard
- So **MinVC** is also NP-hard

# Concluding Remarks on NP-Hardness

# Concluding Remarks

- Our goal in this part of the course was to show some problems are hard to solve

- Reductions allow us to design an “efficient” algorithm for a problem to show that another problem likely does not have an efficient algorithm

- To prove problem B is hard, we pick a problem A which we know is hard and use ANY algorithm for B in a black-box way to get an algorithm for A also

- This means B should be hard as well

# Concluding Remarks

- Our goal in this part of the course was to show some problems are hard to solve

- Reductions allow us to design an "efficient" algorithm for a problem to show that another problem likely does not have an efficient algorithm

- To prove problem B is ~~hard~~ **NP-hard**, we pick a problem A which we know is ~~hard~~ **NP-hard** and use ANY ~~algorithm~~ **algorithm that runs in poly-time** for B in a black-box way to get an ~~algorithm~~ **algorithm that runs in poly-time** for A also

- This means B should be ~~hard~~ **NP-hard** as well