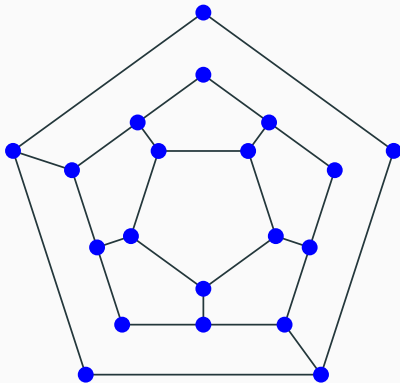




# Pre-lecture brain teaser

Does this graph have a Hamiltonian cycle?



a Yes.

b No.

Answer: B. The outer cycle has two entrances/exits that aren't next to each other always leaving out one or two vertices.

# ECE-374-B: Lecture 21 - Lots of NP-Complete reductions

---

**Instructor:** Abhishek Kumar Umrawal

November 9, 2022

University of Illinois at Urbana-Champaign

# Today

NP-Completeness of the following two problems.

- Hamiltonian cycle
- 3-coloring

**Important:** Understanding the problems and that they are hard.

Proofs and reductions will be sketchy and mainly to give a flavor.

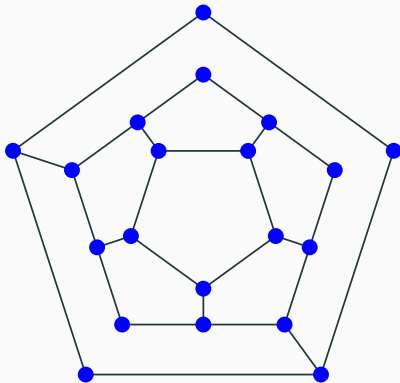
# Reduction from 3SAT to Hamiltonian Cycle

---





Is the following graph Hamiltonian?



a Yes.

b No.

Answer: B. The outer cycle has two entrances/exits that aren't next to each other always leaving out one or two vertices.



# Directed Hamiltonian Cycle is NP-Complete

- Directed Hamiltonian Cycle is in *NP*: exercise.
- **Hardness**: We will show the following.

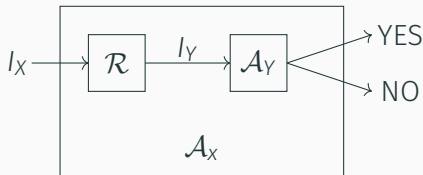
3-SAT  $\leq_P$  Directed Hamiltonian Cycle

# Directed Hamiltonian Cycle is NP-Complete

- Directed Hamiltonian Cycle is in *NP*: exercise.
- **Hardness**: We will show the following.

3-SAT  $\leq_P$  Directed Hamiltonian Cycle

We have the following reduction diagram.



# Reduction

Given 3-SAT formula  $\varphi$  create a graph  $G_\varphi$  such that the following hold.

- $G_\varphi$  has a Hamiltonian cycle if and only if  $\varphi$  is satisfiable.
- $G_\varphi$  should be constructible from  $\varphi$  by a polynomial time algorithm  $\mathcal{A}$ .

**Notation:**  $\varphi$  has  $n$  variables  $x_1, x_2, \dots, x_n$  and  $m$  clauses  $C_1, C_2, \dots, C_m$ .

## Reduction: First Ideas

- Viewing SAT: Assign values to  $n$  variables, and each clause has 3 ways in which it can be satisfied.
- Construct graph with  $2^n$  Hamiltonian cycles, where each cycle corresponds to some boolean assignment.
- Then add more graph structure to encode constraints on assignments imposed by the clauses.

## Reduction: Encoding idea I

Need to create a graph from any arbitrary boolean assignment.  
Consider the following expression.

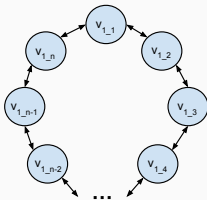
$$f(x_1) = 1 \tag{1}$$

## Reduction: Encoding idea I

Need to create a graph from any arbitrary boolean assignment.  
Consider the following expression.

$$f(x_1) = 1 \tag{1}$$

We create a cyclic graph that always has a Hamiltonian cycle.

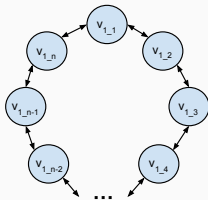


## Reduction: Encoding idea I

Need to create a graph from any arbitrary boolean assignment.  
Consider the following expression.

$$f(x_1) = 1 \quad (1)$$

We create a cyclic graph that always has a Hamiltonian cycle.



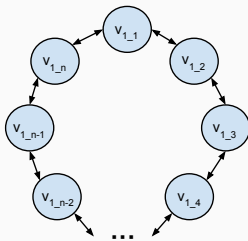
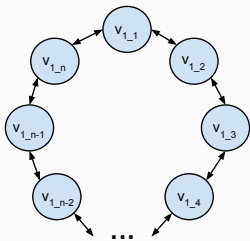
But how do we encode the variable?

## Reduction: Encoding idea I

Need to create a graph from any arbitrary boolean assignment.  
Consider the following expression.

$$f(x_1) = 1$$

Maybe we can encode the variable  $x_1$  in terms of the cycle direction.



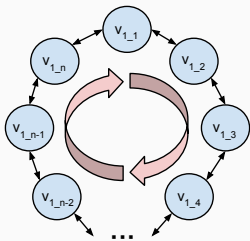


## Reduction: Encoding idea I

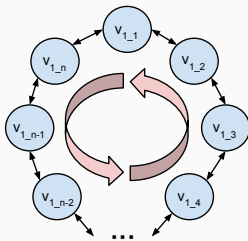
Need to create a graph from any arbitrary boolean assignment.  
Consider the following expression.

$$f(x_1) = 1$$

Maybe we can encode the variable  $x_1$  in terms of the cycle direction.



If  $x_1 = 1$



If  $x_1 = 0$

## Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{2}$$

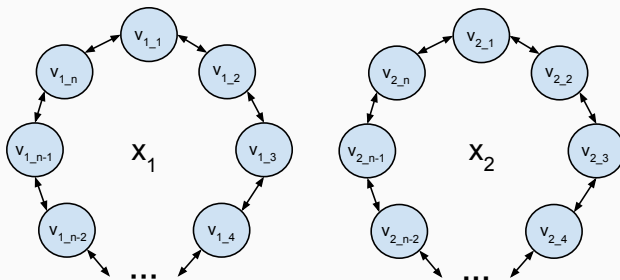
Maybe two circles? Now we need to connect them so that we have a single Hamiltonian cycle.

## Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \quad (2)$$

Maybe two circles? Now we need to connect them so that we have a single Hamiltonian cycle.

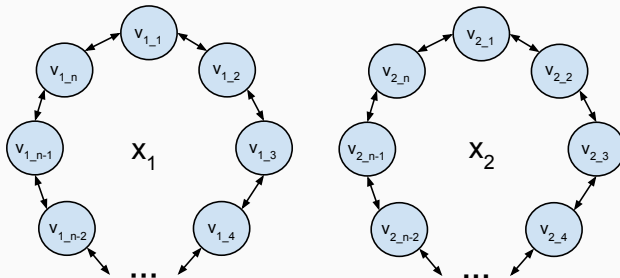


## Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \quad (3)$$

Now we need to connect them so that we have a single Hamiltonian path

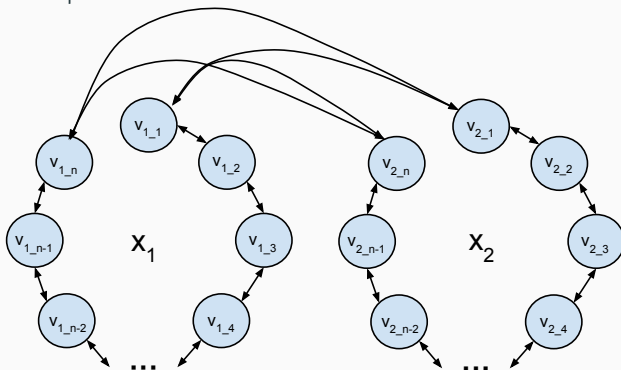


## Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \quad (3)$$

Now we need to connect them so that we have a single Hamiltonian path

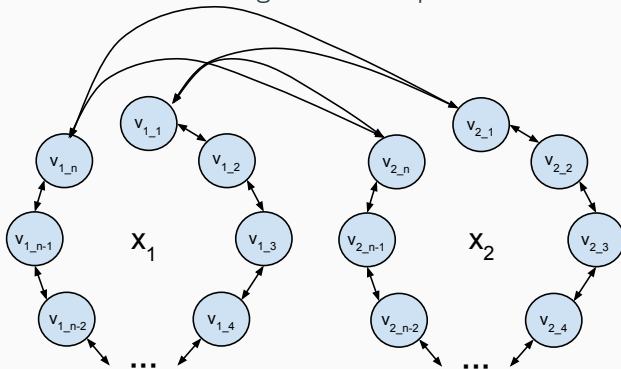


## Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \quad (4)$$

Would be nice to have a single start/stop node.

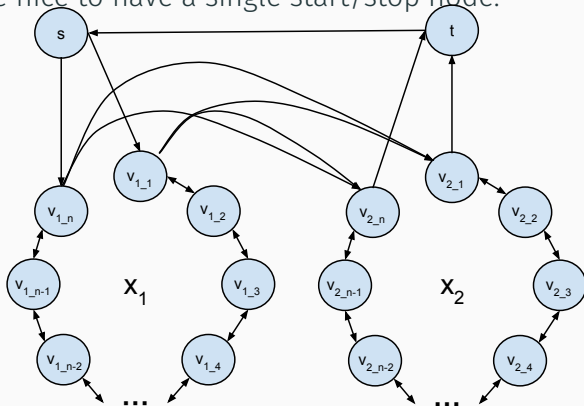


## Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \quad (4)$$

Would be nice to have a single start/stop node.

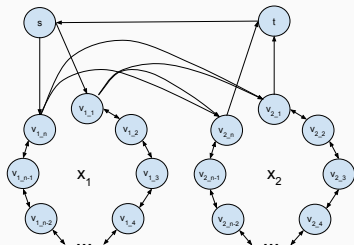


## Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \quad (5)$$

Getting a bit messy. Let's reorganize as follows.



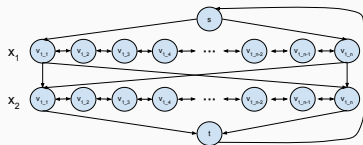
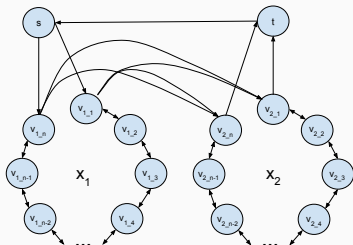


## Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \quad (5)$$

Getting a bit messy. Let's reorganize as follows.

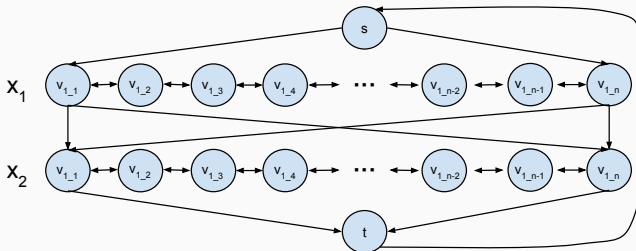


## Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \quad (6)$$

This is how we encode variable assignments in a variable loop!

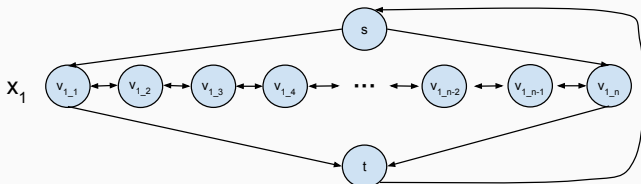


## Reduction: Encoding idea III

How do we handle clauses?

$$f(x_1) = x_1 \quad (7)$$

Lets go back to our one variable graph.

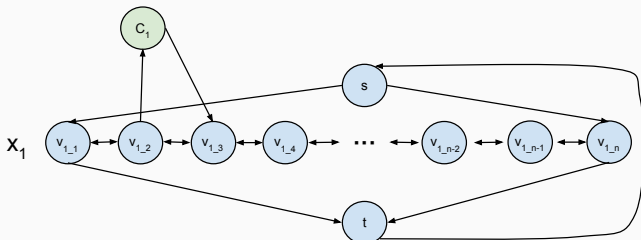


## Reduction: Encoding idea III

How do we handle clauses?

$$f(x_1) = x_1 \quad (8)$$

Add node for clause.

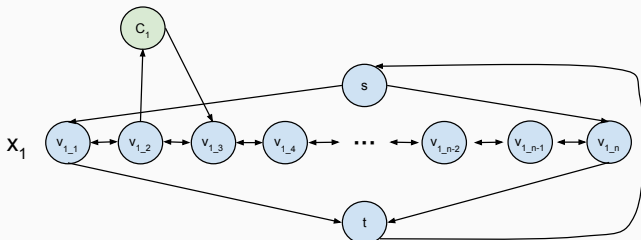


## Reduction: Encoding idea III

How do we handle clauses?

$$f(x_1, x_2) = (x_1 \vee \bar{x}_2) \quad (9)$$

What do we do if the clause has two literals?

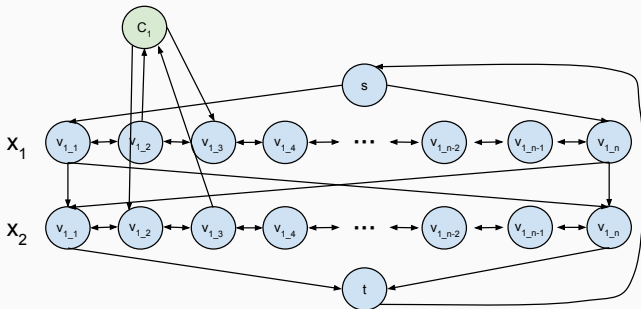


## Reduction: Encoding idea III

How do we handle clauses?

$$f(x_1, x_2) = (x_1 \vee \bar{x}_2) \quad (9)$$

What do we do if the clause has two literals?

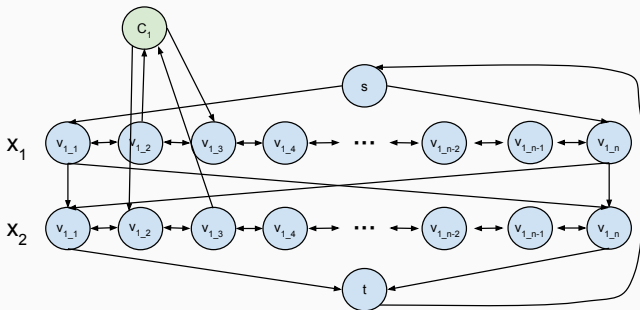


## Reduction: Encoding idea III

How do we handle clauses?

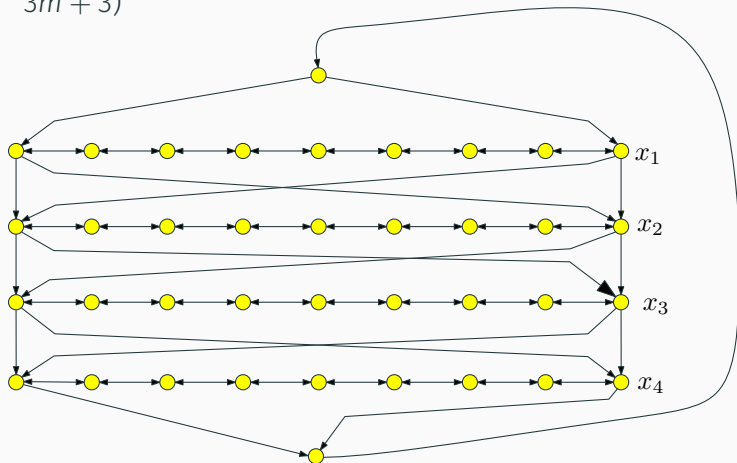
$$f(x_1, x_2) = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \quad (10)$$

What if the expression has multiple clauses:



## The Reduction: Review I

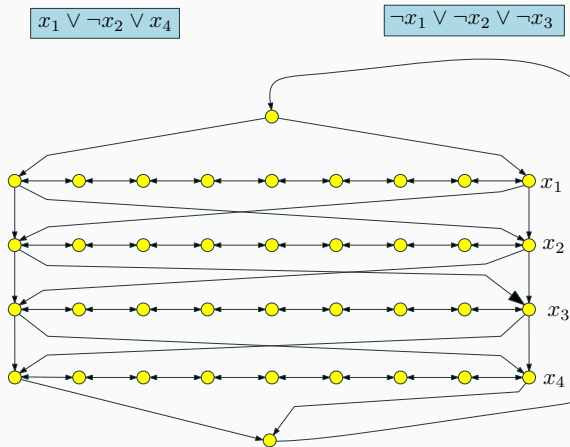
- Traverse path  $i$  from left to right iff  $x_i$  is set to true
- Each path has  $3(m + 1)$  nodes where  $m$  is number of clauses in  $\varphi$ ; nodes numbered from left to right (1 to  $3m + 3$ )





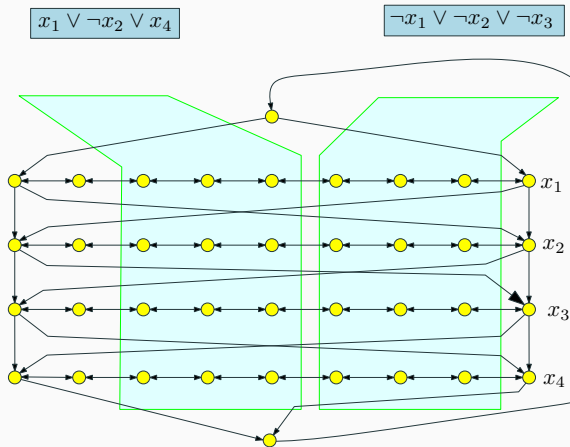
## The Reduction algorithm: Review II

Add vertex  $c_j$  for clause  $C_j$ .  $c_j$  has edge *from* vertex  $3j$  and to vertex  $3j + 1$  on path  $i$  if  $x_i$  appears in clause  $C_j$ , and has edge *from* vertex  $3j + 1$  and to vertex  $3j$  if  $\neg x_i$  appears in  $C_j$ .



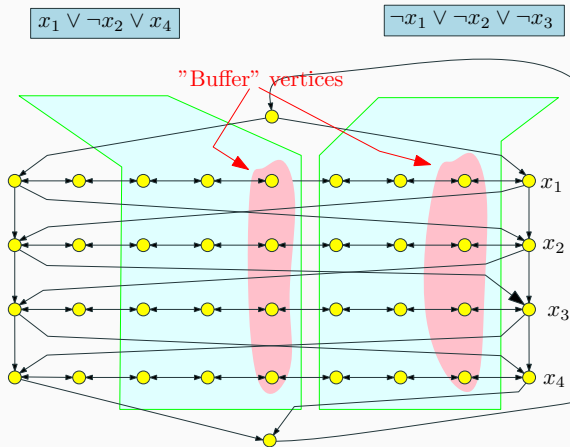
## The Reduction algorithm: Review II

Add vertex  $c_j$  for clause  $C_j$ .  $c_j$  has edge *from* vertex  $3j$  and to vertex  $3j + 1$  on path  $i$  if  $x_i$  appears in clause  $C_j$ , and has edge *from* vertex  $3j + 1$  and to vertex  $3j$  if  $\neg x_i$  appears in  $C_j$ .



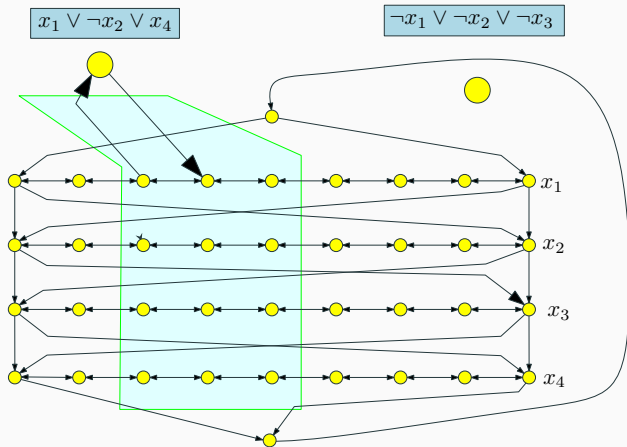
## The Reduction algorithm: Review II

Add vertex  $c_j$  for clause  $C_j$ .  $c_j$  has edge *from* vertex  $3j$  and to vertex  $3j + 1$  on path  $i$  if  $x_i$  appears in clause  $C_j$ , and has edge *from* vertex  $3j + 1$  and to vertex  $3j$  if  $\neg x_i$  appears in  $C_j$ .



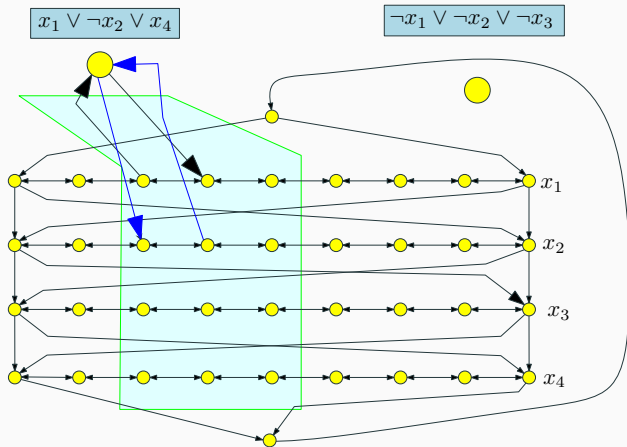
## The Reduction algorithm: Review II

Add vertex  $c_j$  for clause  $C_j$ .  $c_j$  has edge *from* vertex  $3j$  and to vertex  $3j + 1$  on path  $i$  if  $x_i$  appears in clause  $C_j$ , and has edge *from* vertex  $3j + 1$  and to vertex  $3j$  if  $\neg x_i$  appears in  $C_j$ .



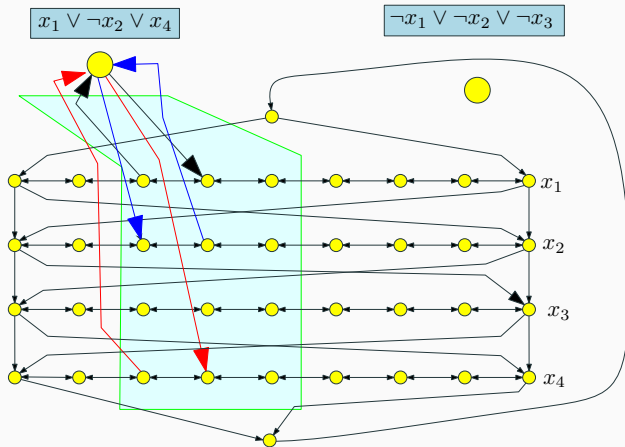
## The Reduction algorithm: Review II

Add vertex  $c_j$  for clause  $C_j$ .  $c_j$  has edge *from* vertex  $3j$  and to vertex  $3j + 1$  on path  $i$  if  $x_i$  appears in clause  $C_j$ , and has edge *from* vertex  $3j + 1$  and to vertex  $3j$  if  $\neg x_i$  appears in  $C_j$ .



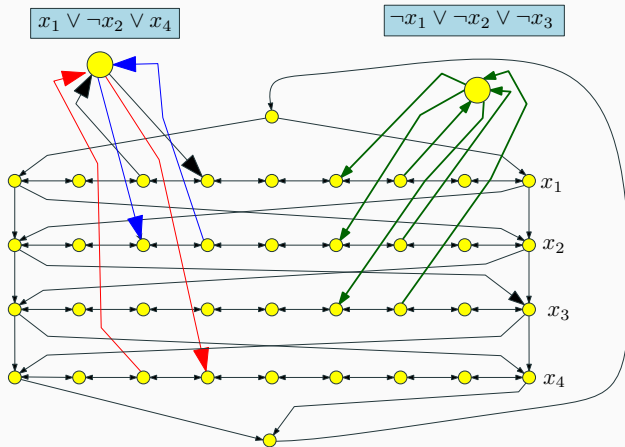
## The Reduction algorithm: Review II

Add vertex  $c_j$  for clause  $C_j$ .  $c_j$  has edge *from* vertex  $3j$  and to vertex  $3j + 1$  on path  $i$  if  $x_i$  appears in clause  $C_j$ , and has edge *from* vertex  $3j + 1$  and to vertex  $3j$  if  $\neg x_i$  appears in  $C_j$ .



## The Reduction algorithm: Review II

Add vertex  $c_j$  for clause  $C_j$ .  $c_j$  has edge *from* vertex  $3j$  and to vertex  $3j + 1$  on path  $i$  if  $x_i$  appears in clause  $C_j$ , and has edge *from* vertex  $3j + 1$  and to vertex  $3j$  if  $\neg x_i$  appears in  $C_j$ .



# Correctness Proof

## Theorem

*$\varphi$  has a satisfying assignment iff  $G_\varphi$  has a Hamiltonian cycle.*

Based on proving the following two lemmas.

## Lemma

*If  $\varphi$  has a satisfying assignment then  $G_\varphi$  has a Hamilton cycle.*

## Lemma

*If  $G_\varphi$  has a Hamilton cycle then  $\varphi$  has a satisfying assignment.*



## Satisfying assignment $\rightarrow$ Hamiltonian Cycle

### Lemma

*If  $\varphi$  has a satisfying assignment then  $G_\varphi$  has a Hamilton cycle.*

### Proof.

$\Rightarrow$  Let  $a$  be the satisfying assignment for  $\varphi$ . Define Hamiltonian cycle as follows.

- If  $a(x_i) = 1$  then traverse path  $i$  from left to right.
- If  $a(x_i) = 0$  then traverse path  $i$  from right to left.
- For each clause, path of at least one variable is in the “right” direction to splice in the node corresponding to clause.

□

## Hamiltonian Cycle $\rightarrow$ Satisfying assignment

Suppose  $\Pi$  is a Hamiltonian cycle in  $G_\varphi$ .

### **Definition**

We say  $\Pi$  is *canonical* if for each clause vertex  $c_j$  the edge of  $\Pi$  entering  $c_j$  and edge of  $\Pi$  leaving  $c_j$  are from the same path corresponding to some variable  $x_i$ . Otherwise  $\Pi$  is *non-canonical* or *emphcheating*.

## Hamiltonian Cycle → Satisfying assignment

Suppose  $\Pi$  is a Hamiltonian cycle in  $G_\varphi$ .

### Definition

We say  $\Pi$  is *canonical* if for each clause vertex  $c_j$  the edge of  $\Pi$  entering  $c_j$  and edge of  $\Pi$  leaving  $c_j$  are from the same path corresponding to some variable  $x_i$ . Otherwise  $\Pi$  is *non-canonical* or *emphcheating*.

### Lemma

*Every Hamilton cycle in  $G_\varphi$  is canonical.*

# Proof of Lemma

## Lemma

*Every Hamilton cycle in  $G_\varphi$  is canonical.*

- If  $\Pi$  enters  $c_j$  (vertex for clause  $C_j$ ) from vertex  $3j$  on path  $i$  then it must leave the clause vertex on edge to  $3j + 1$  on the *same path  $i$* .
  - If not, then only unvisited neighbor of  $3j + 1$  on path  $i$  is  $3j + 2$ .
  - Thus, we don't have two unvisited neighbors (one to enter from, and the other to leave) to have a Hamiltonian Cycle.
- Similarly, if  $\Pi$  enters  $c_j$  from vertex  $3j + 1$  on path  $i$  then it must leave the clause vertex  $c_j$  on edge to  $3j$  on path  $i$ .

## Hamiltonian Cycle $\implies$ Satisfying assignment (contd)

### Lemma

*Any canonical Hamilton cycle in  $G_\varphi$  corresponds to a satisfying truth assignment to  $\varphi$ .*

Consider a canonical Hamilton cycle  $\Pi$ .

- For every clause vertex  $c_j$ , vertices visited immediately before and after  $c_j$  are connected by an edge on same path corresponding to some variable  $x_i$ .
- We can remove  $c_j$  from cycle, and get Hamiltonian cycle in  $G - c_j$ .
- Hamiltonian cycle from  $\Pi$  in  $G - \{c_1, \dots, c_m\}$  traverses each path in only one direction, which determines truth assignment.
- Easy to verify that this truth assignment satisfies  $\varphi$ .

# Hamiltonian cycle in undirected graph

---

# Hamiltonian Cycle in *Undirected* Graphs

## Problem

**Input** Given *undirected* graph  $G = (V, E)$ .

**Goal** Does  $G$  have a Hamiltonian cycle? That is, is there a cycle that visits every vertex exactly one (except start and end vertex)?

## Theorem

*Hamiltonian cycle* problem for undirected graphs is NP-Complete.

## Proof.

- The problem is in **NP**; proof left as exercise.
- Hardness proved by reducing Directed Hamiltonian Cycle to this problem. □

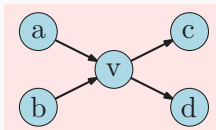


# Reduction Sketch

**Goal:** Given directed graph  $G$ , need to construct undirected graph  $G'$  such that  $G$  has Hamiltonian cycle iff  $G'$  has Hamiltonian path.

## Reduction

- 
- 

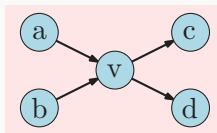


# Reduction Sketch

**Goal:** Given directed graph  $G$ , need to construct undirected graph  $G'$  such that  $G$  has Hamiltonian cycle iff  $G'$  has Hamiltonian path.

## Reduction

- Replace each vertex  $v$  by 3 vertices:  $v_{in}$ ,  $v$ , and  $v_{out}$ .
- 

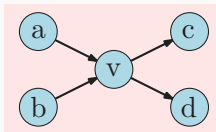


# Reduction Sketch

**Goal:** Given directed graph  $G$ , need to construct undirected graph  $G'$  such that  $G$  has Hamiltonian cycle iff  $G'$  has Hamiltonian path.

## Reduction

- Replace each vertex  $v$  by 3 vertices:  $v_{in}$ ,  $v$ , and  $v_{out}$ .
- A directed edge  $(a, b)$  is replaced by edge  $(a_{out}, b_{in})$ .

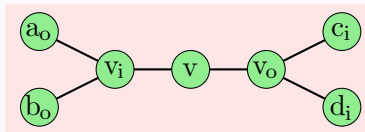
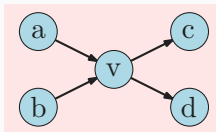


# Reduction Sketch

**Goal:** Given directed graph  $G$ , need to construct undirected graph  $G'$  such that  $G$  has Hamiltonian cycle iff  $G'$  has Hamiltonian path.

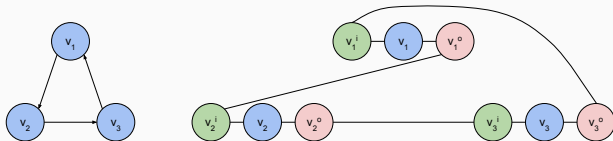
## Reduction

- Replace each vertex  $v$  by 3 vertices:  $v_{in}$ ,  $v$ , and  $v_{out}$ .
- A directed edge  $(a, b)$  is replaced by edge  $(a_{out}, b_{in})$ .



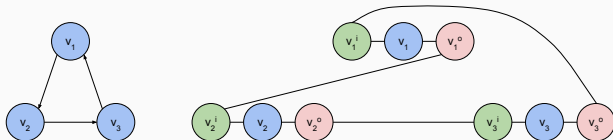
# Reduction Sketch Example

Graph *with* cycle:

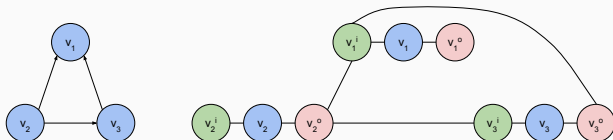


# Reduction Sketch Example

Graph *with* cycle:



Graph *without* cycle:



## Reduction: Wrapup

- The reduction is polynomial time: exercise.
- The reduction is correct: exercise.

# Hamiltonian Path

**Input** Given a graph  $G = (V, E)$  with  $n$  vertices

**Goal** Does  $G$  have a **Hamiltonian path**?

- A Hamiltonian path is a path in the graph that visits every vertex in  $G$  exactly once.



# Hamiltonian Path

**Input** Given a graph  $G = (V, E)$  with  $n$  vertices

**Goal** Does  $G$  have a **Hamiltonian path**?

- A Hamiltonian path is a path in the graph that visits every vertex in  $G$  exactly once.

## Theorem

*Directed Hamiltonian Path and Undirected Hamiltonian Path are NP-Complete.*

Easy to modify the reduction from **3-SAT** to **Hamiltonian Cycle** or do a reduction from **Hamiltonian Cycle**.

# Hamiltonian Path

**Input** Given a graph  $G = (V, E)$  with  $n$  vertices

**Goal** Does  $G$  have a **Hamiltonian path**?

- A Hamiltonian path is a path in the graph that visits every vertex in  $G$  exactly once.

## Theorem

*Directed Hamiltonian Path and Undirected Hamiltonian Path are NP-Complete.*

Easy to modify the reduction from **3-SAT** to **Hamiltonian Cycle** or do a reduction from **Hamiltonian Cycle**.

Implies that **Longest Simple Path** in a graph is NP-Complete.

# NP-Completeness of Graph Coloring

---

## Problem: Graph Coloring

**Instance:**  $G = (V, E)$ : Undirected graph, integer  $k$ .

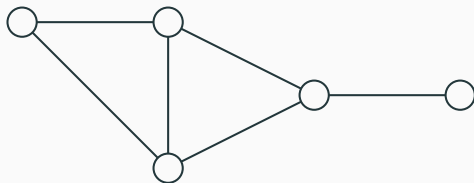
**Question:** Can the vertices of the graph be colored using  $k$  colors so that vertices connected by an edge do not get the same color?

# Graph 3-Coloring

## Problem: 3 Coloring

**Instance:**  $G = (V, E)$ : Undirected graph.

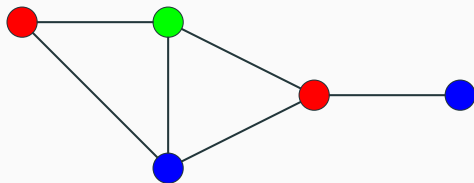
**Question:** Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?



## Problem: 3 Coloring

**Instance:**  $G = (V, E)$ : Undirected graph.

**Question:** Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?



# Graph Coloring

**Observation:** If  $G$  is colored with  $k$  colors then each color class (nodes of same color) form an independent set in  $G$ . Thus,  $G$  can be partitioned into  $k$  independent sets iff  $G$  is  $k$ -colorable.

Graph 2-Coloring can be decided in polynomial time.

$G$  is 2-colorable iff  $G$  is bipartite! There is a linear time algorithm to check if  $G$  is bipartite using breadth first search.

## Problems related to graph coloring

---



# Graph Coloring and Register Allocation

## Register Allocation

Assign variables to (at most)  $k$  registers such that variables needed at the same time are not assigned to the same register.

## Interference Graph

Vertices are variables, and there is an edge between two vertices, if the two variables are “live” at the same time.

## Observations

- [Chaitin] Register allocation problem is equivalent to coloring the interference graph with  $k$  colors.
- Moreover,  $3\text{-COLOR} \leq_P k - \text{Register Allocation}$ , for any  $k \geq 3$ .

# Class Room Scheduling

Given  $n$  classes and their meeting times, are  $k$  rooms sufficient?

Reduce to Graph  $k$ -Coloring problem.

Create graph  $G$

- a node  $v_i$  for each class  $i$ .
- an edge between  $v_i$  and  $v_j$  if classes  $i$  and  $j$  *conflict*.

Exercise:  $G$  is  $k$ -colorable iff  $k$  rooms are sufficient.

# Frequency Assignments in Cellular Networks

Cellular telephone systems that use Frequency Division Multiple Access (FDMA) (example: GSM in Europe and Asia and AT&T in USA).

- Breakup a frequency range  $[a, b]$  into disjoint *bands* of frequencies  $[a_0, b_0], [a_1, b_1], \dots, [a_k, b_k]$ .
- Each cell phone tower (simplifying) gets one band.
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference.

# Frequency Assignments in Cellular Networks

Cellular telephone systems that use Frequency Division Multiple Access (FDMA) (example: GSM in Europe and Asia and AT&T in USA).

- Breakup a frequency range  $[a, b]$  into disjoint *bands* of frequencies  $[a_0, b_0], [a_1, b_1], \dots, [a_k, b_k]$ .
- Each cell phone tower (simplifying) gets one band.
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference.

**Problem:** given  $k$  bands and some region with  $n$  towers, is there a way to assign the bands to avoid interference?

Can reduce to  $k$ -coloring by creating interference/conflict graph on towers.

Showing hardness of 3 COLORING

---

# 3-Coloring is NP-Complete

- **3-Coloring** is in NP.
  - Non-deterministically guess a 3-coloring for each node
  - Check if for each edge  $(u, v)$ , the color of  $u$  is different from that of  $v$ .
- **Hardness:** We will show  $3\text{-SAT} \leq_P 3\text{-Coloring}$ .

# Reduction Idea

Start with **3SAT** formula (i.e., **3CNF** formula)  $\varphi$  with  $n$  variables  $x_1, \dots, x_n$  and  $m$  clauses  $C_1, \dots, C_m$ . Create graph  $G_\varphi$  such that  $G_\varphi$  is 3-colorable iff  $\varphi$  is satisfiable.

- need to establish truth assignment for  $x_1, \dots, x_n$  via colors for some nodes in  $G_\varphi$ .
- create triangle with node True, False, Base.
- for each variable  $x_i$  two nodes  $v_i$  and  $\bar{v}_i$  connected in a triangle with common Base.
- If graph is 3-colored, either  $v_i$  or  $\bar{v}_i$  gets the same color as True. Interpret this as a truth assignment to  $v_i$ .
- Need to add constraints to ensure clauses are satisfied (next phase).

## Reduction Idea 1 - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true.
- Not 3-colorable if none of the literals are true.



## Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true.
- Not 3-colorable if none of the literals are true.

Let's start off with the simplest SAT we can think of:

$$f(x_1, x_2) = (x_1 \vee x_2) \quad (11)$$

## Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true.
- Not 3-colorable if none of the literals are true.

Let's start off with the simplest SAT we can think of:

$$f(x_1, x_2) = (x_1 \vee x_2) \quad (11)$$

Assume green=true and red=false,

## Reduction Idea 1 - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

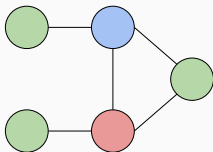
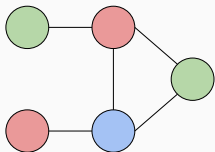
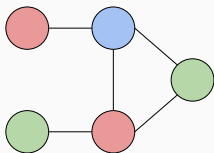
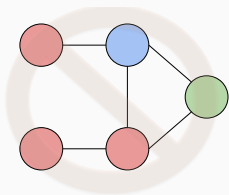
**Let's try some stuff:**

## Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3-colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

Seems to work:



## Reduction Idea 1 - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true.
- Not 3-colorable if none of the literals are true.

## Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true.
- Not 3-colorable if none of the literals are true.

How do we do the same thing for 3 variables?

$$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \quad (12)$$

## Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true.
- Not 3-colorable if none of the literals are true.

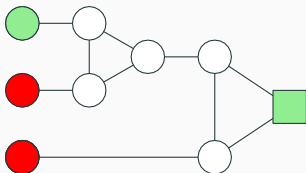
How do we do the same thing for 3 variables?

$$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \quad (12)$$

Assume green=true and red=false.

## 3 color this gadget II

You are given three colors: red, green and blue. Can the following graph be three colored in a valid way (assuming that some of the nodes are already colored as indicated).

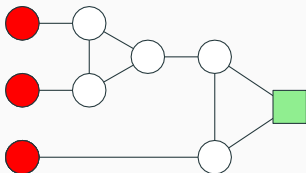


- a Yes.
- b No.



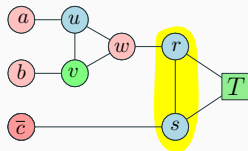
## 3 color this gadget.

You are given three colors: red, green and blue. Can the following graph be three colored in a valid way (assuming that some of the nodes are already colored as indicated).

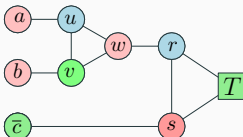


- a Yes.
- b No.

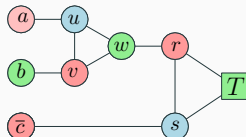
# 3-coloring of the clause gadget



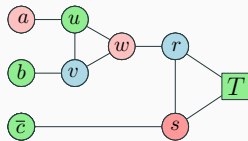
FFF - BAD



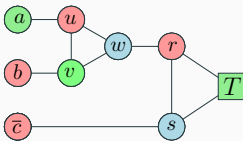
FFT



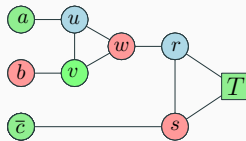
FTF



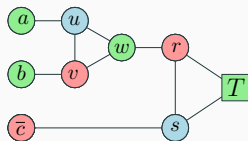
FTT



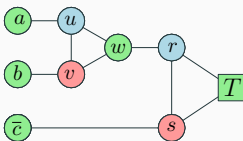
TFF



TFT



TTF



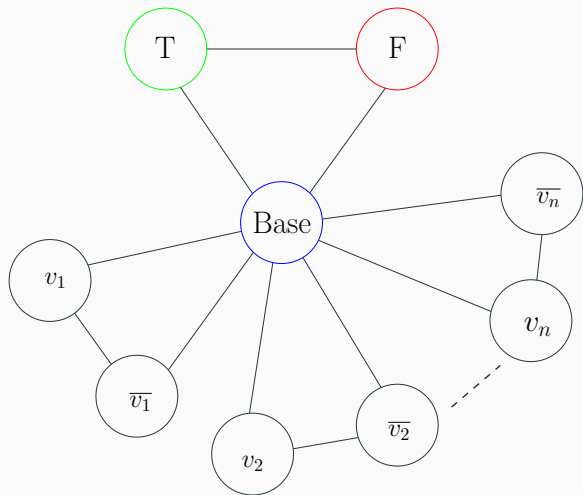
TTT

## Reduction Idea II - Literal Assignment I

Next we need a gadget that assigns literals. Our previously constructed gadget assumes the following.

- All literals are either red or green.
- Need to limit graph so only  $x_1$  or  $\bar{x}_1$  is green. Other must be red.

## Reduction Idea II - Literal Assignment II

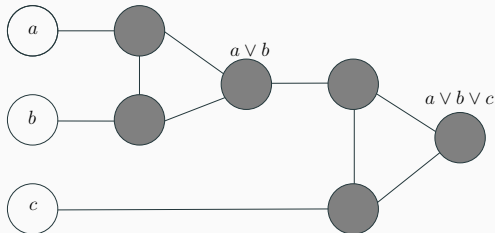


## Review Clause Satisfiability Gadget

For each clause  $C_j = (a \vee b \vee c)$ , create a small gadget. graph

- Gadget graph connects to nodes corresponding to  $a, b, c$ .
- Needs to implement OR.

OR-gadget-graph:



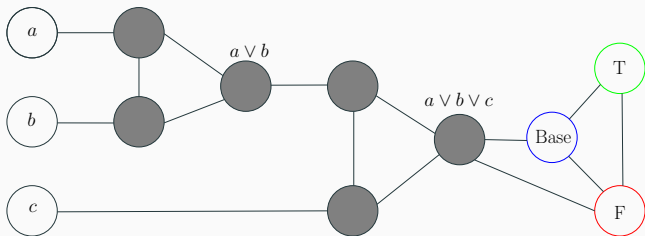
## OR-Gadget Graph

**Property:** If  $a, b, c$  are colored False in a 3-coloring then output node of OR-gadget has to be colored False.

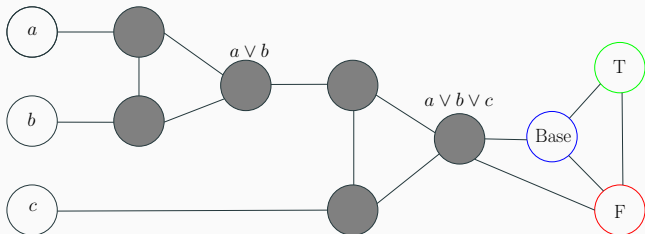
**Property:** If one of  $a, b, c$  is colored True then OR-gadget can be 3-colored such that output node of OR-gadget is colored True.

# Reduction

- Create triangle with nodes True, False, Base.
- For each variable  $x_i$  two nodes  $v_i$  and  $\bar{v}_i$  connected in a triangle with common Base.
- For each clause  $C_j = (a \vee b \vee c)$ , add OR-gadget graph with input nodes  $a, b, c$  and connect output node of gadget to both False and Base.



# Reduction



## Lemma

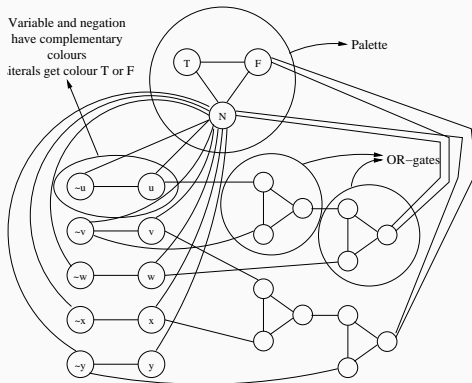
*No legal 3-coloring of above graph (with coloring of nodes  $T, F, B$  fixed) in which  $a, b, c$  are colored False. If any of  $a, b, c$  are colored True then there is a legal 3-coloring of above graph.*



# Reduction Outline

## Example

$$\varphi = (u \vee \neg v \vee w) \wedge (v \vee x \vee \neg y)$$



## Correctness of Reduction

$\varphi$  is satisfiable implies  $G_\varphi$  is 3-colorable

- If  $x_i$  is assigned True, color  $v_i$  True and  $\bar{v}_i$  False.

## Correctness of Reduction

$\varphi$  is satisfiable implies  $G_\varphi$  is 3-colorable

- If  $x_i$  is assigned True, color  $v_i$  True and  $\bar{v}_i$  False.
- For each clause  $C_j = (a \vee b \vee c)$  at least one of  $a, b, c$  is colored True. OR-gadget for  $C_j$  can be 3-colored such that output is True.

## Correctness of Reduction

$\varphi$  is satisfiable implies  $G_\varphi$  is 3-colorable

- If  $x_i$  is assigned True, color  $v_i$  True and  $\bar{v}_i$  False.
- For each clause  $C_j = (a \vee b \vee c)$  at least one of  $a, b, c$  is colored True. OR-gadget for  $C_j$  can be 3-colored such that output is True.

## Correctness of Reduction

$\varphi$  is satisfiable implies  $G_\varphi$  is 3-colorable

- If  $x_i$  is assigned True, color  $v_i$  True and  $\bar{v}_i$  False.
- For each clause  $C_j = (a \vee b \vee c)$  at least one of  $a, b, c$  is colored True. OR-gadget for  $C_j$  can be 3-colored such that output is True.

$G_\varphi$  is 3-colorable implies  $\varphi$  is satisfiable

- If  $v_i$  is colored True then set  $x_i$  to be True, this is a legal truth assignment.

## Correctness of Reduction

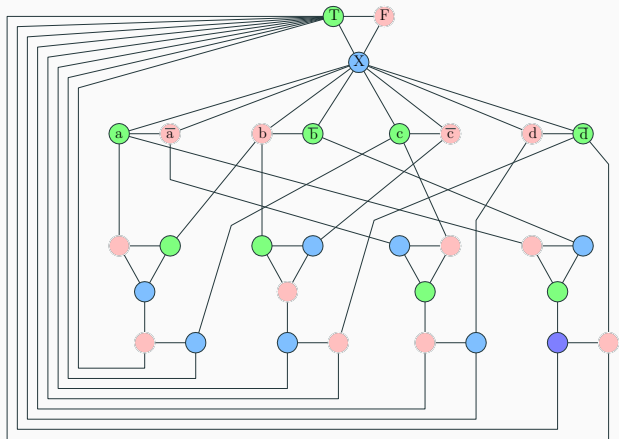
$\varphi$  is satisfiable implies  $G_\varphi$  is 3-colorable

- If  $x_i$  is assigned True, color  $v_i$  True and  $\bar{v}_i$  False.
- For each clause  $C_j = (a \vee b \vee c)$  at least one of  $a, b, c$  is colored True. OR-gadget for  $C_j$  can be 3-colored such that output is True.

$G_\varphi$  is 3-colorable implies  $\varphi$  is satisfiable

- If  $v_i$  is colored True then set  $x_i$  to be True, this is a legal truth assignment.
- Consider any clause  $C_j = (a \vee b \vee c)$ . it cannot be that all  $a, b, c$  are False. If so, output of OR-gadget for  $C_j$  has to be colored False but output is connected to Base and False!

# Graph generated in reduction from 3SAT to 3COLOR



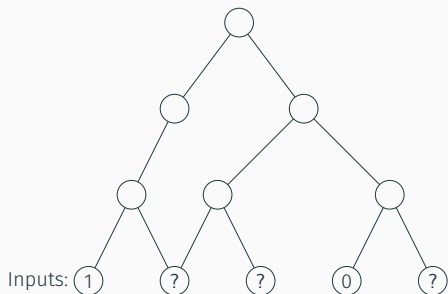
# Circuit-Sat Problem

---



# Circuits

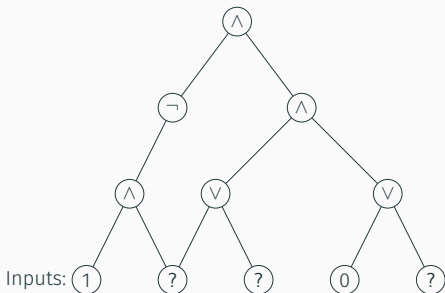
A circuit is a directed *acyclic* graph with



- **Input** vertices (without incoming edges) labeled with 0, 1 or a distinct variable.
- Every other vertex is labeled  $\vee$ ,  $\wedge$  or  $\neg$ .
- Single node **output** vertex with no outgoing edges.

# Circuits

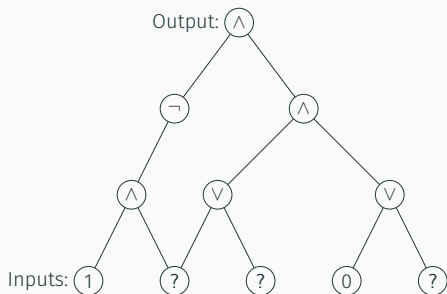
A circuit is a directed *acyclic* graph with



- **Input** vertices (without incoming edges) labeled with 0, 1 or a distinct variable.
- Every other vertex is labeled  $\vee$ ,  $\wedge$  or  $\neg$ .
- Single node **output** vertex with no outgoing edges.

# Circuits

A circuit is a directed *acyclic* graph with



- **Input** vertices (without incoming edges) labeled with 0, 1 or a distinct variable.
- Every other vertex is labeled  $\vee$ ,  $\wedge$  or  $\neg$ .
- Single node **output** vertex with no outgoing edges.

## CSAT: Circuit Satisfaction

### Definition (Circuit Satisfaction (CSAT).)

Given a circuit as input, is there an assignment to the input variables that causes the output to get value 1?

# CSAT: Circuit Satisfaction

## Definition (Circuit Satisfaction (CSAT).)

Given a circuit as input, is there an assignment to the input variables that causes the output to get value 1?

## Lemma

*CSAT is in NP.*

- **Certificate:** Assignment to input variables.
- **Certifier:** Evaluate the value of each gate in a topological sort of **DAG** and check the output gate value.

## Circuit SAT vs SAT

CNF formulas are a rather restricted form of Boolean formulas.

Circuits are a much more powerful (and hence easier) way to express Boolean formulas

# Circuit SAT vs SAT

CNF formulas are a rather restricted form of Boolean formulas.

Circuits are a much more powerful (and hence easier) way to express Boolean formulas

However they are equivalent in terms of polynomial-time solvability.

**Theorem**

$$SAT \leq_P 3SAT \leq_P CSAT.$$

**Theorem**

$$CSAT \leq_P SAT \leq_P 3SAT.$$

## Converting a CNF formula into a Circuit

Given 3CNF formula  $\varphi$  with  $n$  variables and  $m$  clauses, create a Circuit  $C$ .

- Inputs to  $C$  are the  $n$  boolean variables  $x_1, x_2, \dots, x_n$
- Use NOT gate to generate literal  $\neg x_i$  for each variable  $x_i$
- For each clause  $(l_1 \vee l_2 \vee l_3)$  use two OR gates to mimic formula
- Combine the outputs for the clauses using AND gates to obtain the final output



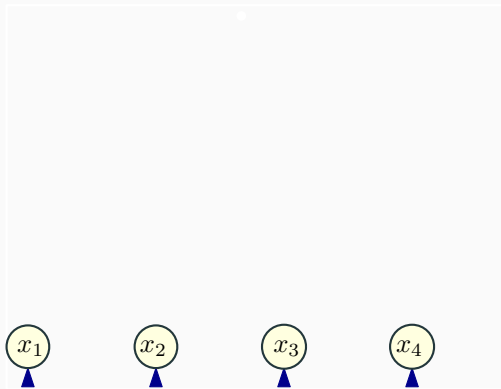
## Example: $3SAT \leq_P CSAT$

$$\varphi = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$



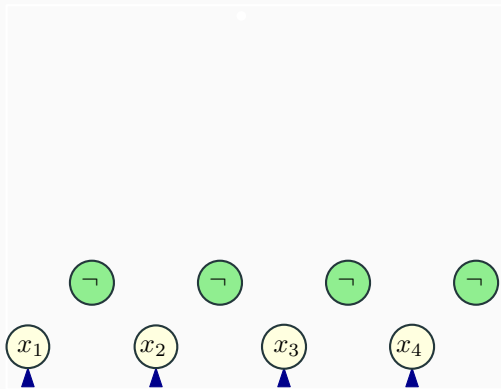
## Example: $3SAT \leq_P CSAT$

$$\varphi = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$



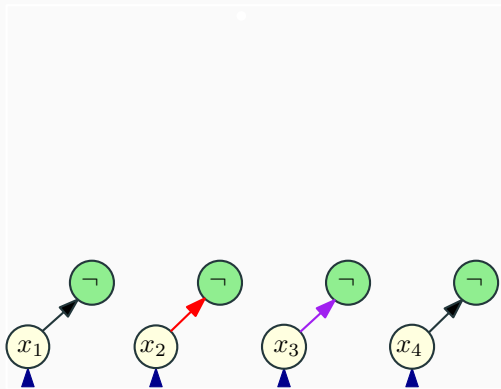
# Example: $3SAT \leq_P CSAT$

$$\varphi = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$



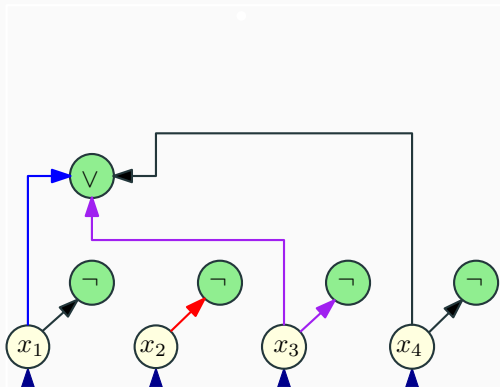
# Example: $3SAT \leq_P CSAT$

$$\varphi = (x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$



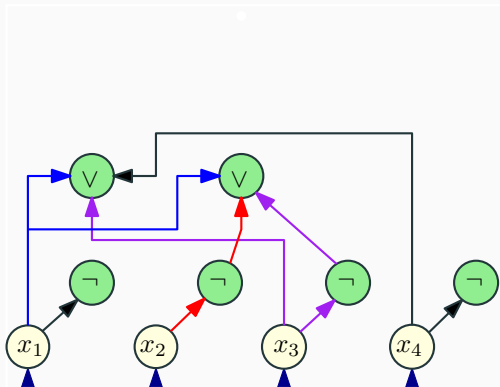
# Example: $3SAT \leq_P CSAT$

$$\varphi = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$



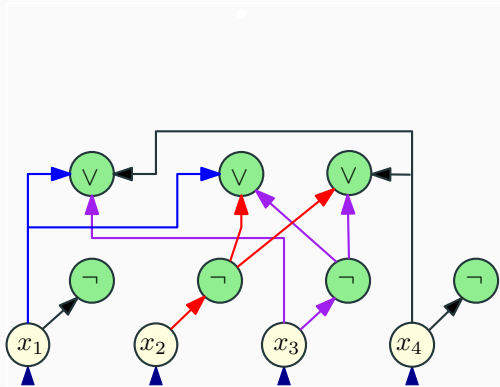
# Example: $3SAT \leq_P CSAT$

$$\varphi = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$



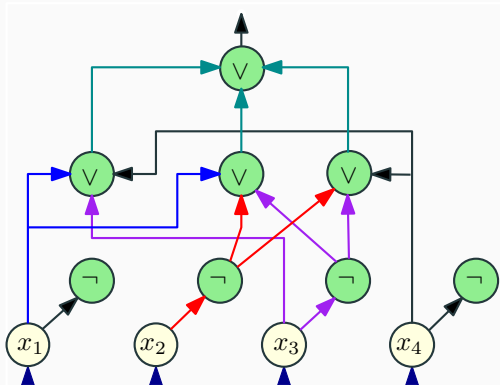
# Example: $3SAT \leq_P CSAT$

$$\varphi = (x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$



# Example: $3SAT \leq_P CSAT$

$$\varphi = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$





## Converting a circuit to a SAT formula

What will converting a circuit to a SAT formula prove?

## Converting a circuit to a SAT formula

What will converting a circuit to a SAT formula prove?

But first we need to look back at a gadget!

## Converting $z = x \wedge y$ to 3SAT

z	x	y					
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

## Converting $z = x \wedge y$ to 3SAT

$z$	$x$	$y$	$z = x \wedge y$				
0	0	0	1				
0	0	1	1				
0	1	0	1				
0	1	1	0				
1	0	0	0				
1	0	1	0				
1	1	0	0				
1	1	1	1				

## Converting $z = x \wedge y$ to 3SAT

$z$	$x$	$y$	$z = x \wedge y$				
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

## Converting $z = x \wedge y$ to 3SAT

$z$	$x$	$y$	$z = x \wedge y$	$z \vee \bar{x} \vee \bar{y}$			
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

# Converting $z = x \wedge y$ to 3SAT

$z$	$x$	$y$	$z = x \wedge y$	$z \vee \bar{x} \vee \bar{y}$	$\bar{z} \vee x \vee y$		
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

# Converting $z = x \wedge y$ to 3SAT

$z$	$x$	$y$	$z = x \wedge y$	$z \vee \bar{x} \vee \bar{y}$	$\bar{z} \vee x \vee y$	$\bar{z} \vee x \vee \bar{y}$	
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1



## Converting $z = x \wedge y$ to 3SAT

$z$	$x$	$y$	$z = x \wedge y$	$z \vee \bar{x} \vee \bar{y}$	$\bar{z} \vee x \vee y$	$\bar{z} \vee x \vee \bar{y}$	$\bar{z} \vee \bar{x} \vee y$
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

## Converting $z = x \wedge y$ to 3SAT

$z$	$x$	$y$	$z = x \wedge y$	$z \vee \bar{x} \vee \bar{y}$	$\bar{z} \vee x \vee y$	$\bar{z} \vee x \vee \bar{y}$	$\bar{z} \vee \bar{x} \vee y$
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

## Converting $z = x \wedge y$ to 3SAT

$z$	$x$	$y$	$z = x \wedge y$	$z \vee \bar{x} \vee \bar{y}$	$\bar{z} \vee x \vee y$	$\bar{z} \vee x \vee \bar{y}$	$\bar{z} \vee \bar{x} \vee y$
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

$$(z = x \wedge y)$$

$$\equiv$$

$$(z \vee \bar{x} \vee \bar{y}) \wedge (\bar{z} \vee x \vee y) \wedge (\bar{z} \vee x \vee \bar{y}) \wedge (\bar{z} \vee \bar{x} \vee y)$$

## Summary of formulas we derived

### Lemma

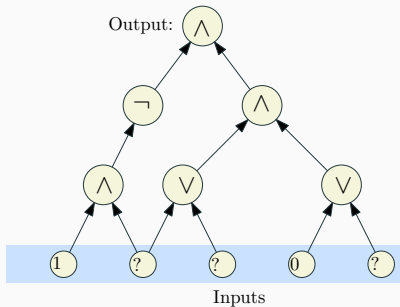
*The following identities hold:*

$$\bullet z = \bar{x} \quad \equiv \quad (z \vee x) \wedge (\bar{z} \vee \bar{x}).$$

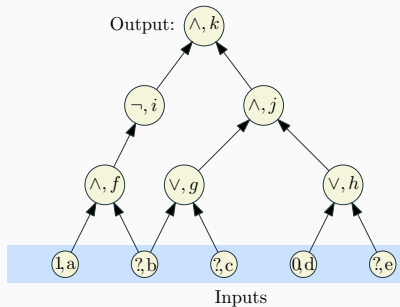
$$\bullet (z = x \vee y) \quad \equiv \quad (z \vee \bar{y}) \wedge (z \vee \bar{x}) \wedge (\bar{z} \vee x \vee y)$$

$$\bullet (z = x \wedge y) \quad \equiv \quad (z \vee \bar{x} \vee \bar{y}) \wedge (\bar{z} \vee x) \wedge (\bar{z} \vee y)$$

# Converting a circuit into a CNF formula

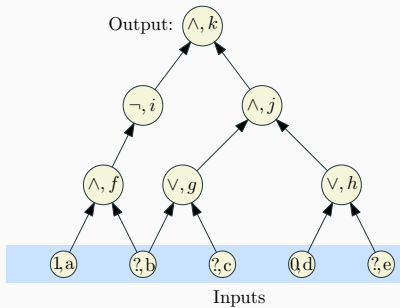


(A) Input circuit

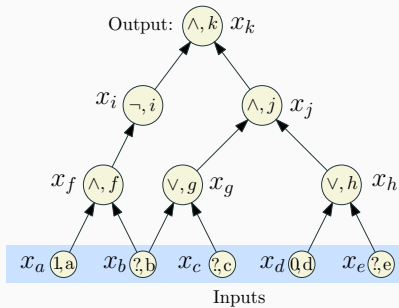


(B) Label the nodes.

# Converting a circuit into a CNF formula

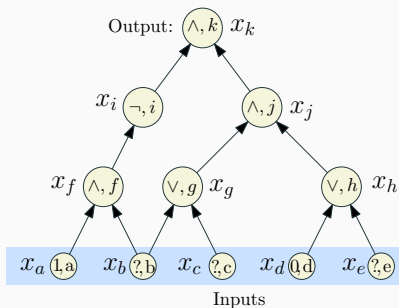


(B) Label the nodes.



(C) Introduce var for each node.

# Converting a circuit into a CNF formula



(C) Introduce var for each node.

$x_k$  (Demand a sat' assignment!)

$$x_k = x_i \wedge x_j$$

$$x_j = x_g \wedge x_h$$

$$x_i = \neg x_f$$

$$x_h = x_d \vee x_e$$

$$x_g = x_b \vee x_c$$

$$x_f = x_a \wedge x_b$$

$$x_d = 0$$

$$x_a = 1$$

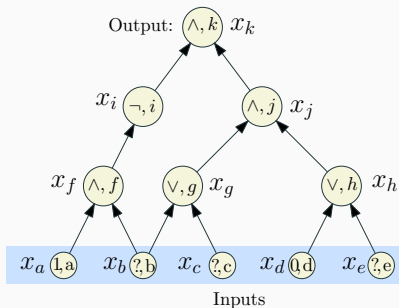
(D) Write a sub-formula for each variable that is true if the var is computed correctly.

## Converting a circuit into a CNF formula

$X_k$	$X_k$
$X_k = X_i \wedge X_j$	$(\neg X_k \vee X_i) \wedge (\neg X_k \vee X_j) \wedge (X_k \vee \neg X_i \vee \neg X_j)$
$X_j = X_g \wedge X_h$	$(\neg X_j \vee X_g) \wedge (\neg X_j \vee X_h) \wedge (X_j \vee \neg X_g \vee \neg X_h)$
$X_i = \neg X_f$	$(X_i \vee X_f) \wedge (\neg X_i \vee \neg X_f)$
$X_h = X_d \vee X_e$	$(X_h \vee \neg X_d) \wedge (X_h \vee \neg X_e) \wedge (\neg X_h \vee X_d \vee X_e)$
$X_g = X_b \vee X_c$	$(X_g \vee \neg X_b) \wedge (X_g \vee \neg X_c) \wedge (\neg X_g \vee X_b \vee X_c)$
$X_f = X_a \wedge X_b$	$(\neg X_f \vee X_a) \wedge (\neg X_f \vee X_b) \wedge (X_f \vee \neg X_a \vee \neg X_b)$
$X_d = 0$	$\neg X_d$
$X_a = 1$	$X_a$



## Converting a circuit into a CNF formula



$$\begin{aligned} & x_k \wedge (\neg x_k \vee x_i) \wedge (\neg x_k \vee x_j) \\ & \wedge (x_k \vee \neg x_i \vee \neg x_j) \wedge (\neg x_j \vee x_g) \\ & \wedge (\neg x_j \vee x_h) \wedge (x_j \vee \neg x_g \vee \neg x_h) \\ & \wedge (x_i \vee x_f) \wedge (\neg x_i \vee \neg x_f) \\ & \wedge (x_h \vee \neg x_d) \wedge (x_h \vee \neg x_e) \\ & \wedge (\neg x_h \vee x_d \vee x_e) \wedge (x_g \vee \neg x_b) \\ & \wedge (x_g \vee \neg x_c) \wedge (\neg x_g \vee x_b \vee x_c) \\ & \wedge (\neg x_f \vee x_a) \wedge (\neg x_f \vee x_b) \\ & \wedge (x_f \vee \neg x_a \vee \neg x_b) \wedge (\neg x_d) \wedge x_a \end{aligned}$$

We got a CNF formula that is satisfiable if and only if the original circuit is satisfiable.

## Reduction: $CSAT \leq_P SAT$

- For each gate (vertex)  $v$  in the circuit, create a variable  $x_v$
- **Case  $\neg$ :**  $v$  is labeled  $\neg$  and has one incoming edge from  $u$  (so  $x_v = \neg x_u$ ). In **SAT** formula generate, add clauses  $(x_u \vee x_v)$ ,  $(\neg x_u \vee \neg x_v)$ . Observe that

$$x_v = \neg x_u \text{ is true} \iff \begin{array}{l} (x_u \vee x_v) \\ (\neg x_u \vee \neg x_v) \end{array} \text{ both true.}$$

## Reduction: $CSAT \leq_P SAT$

- **Case v:** So  $x_v = x_u \vee x_w$ . In **SAT** formula generated, add clauses  $(x_v \vee \neg x_u)$ ,  $(x_v \vee \neg x_w)$ , and  $(\neg x_v \vee x_u \vee x_w)$ . Again, observe that

$$(x_v = x_u \vee x_w) \text{ is true} \iff \begin{array}{l} (x_v \vee \neg x_u), \\ (x_v \vee \neg x_w), \\ (\neg x_v \vee x_u \vee x_w) \end{array} \text{ all true.}$$

## Reduction: $CSAT \leq_P SAT$

- **Case  $\wedge$ :** So  $x_V = x_U \wedge x_W$ . In **SAT** formula generated, add clauses  $(\neg x_V \vee x_U)$ ,  $(\neg x_V \vee x_W)$ , and  $(x_V \vee \neg x_U \vee \neg x_W)$ . Again observe that

$$x_V = x_U \wedge x_W \text{ is true} \iff \begin{array}{l} (\neg x_V \vee x_U), \\ (\neg x_V \vee x_W), \\ (x_V \vee \neg x_U \vee \neg x_W) \end{array} \text{ all true.}$$

## Reduction: $CSAT \leq_P SAT$

- If  $v$  is an input gate with a fixed value then we do the following. If  $x_v = 1$  add clause  $x_v$ . If  $x_v = 0$  add clause  $\neg x_v$
- Add the clause  $x_v$  where  $v$  is the variable for the output gate

# Correctness of Reduction

Need to show circuit  $C$  is satisfiable iff  $\varphi_C$  is satisfiable

$\Rightarrow$  Consider a satisfying assignment  $a$  for  $C$

- Find values of all gates in  $C$  under  $a$
- Give value of gate  $v$  to variable  $x_v$ ; call this assignment  $a'$
- $a'$  satisfies  $\varphi_C$  (exercise)

$\Leftarrow$  Consider a satisfying assignment  $a$  for  $\varphi_C$

- Let  $a'$  be the restriction of  $a$  to only the input variables
- Value of gate  $v$  under  $a'$  is the same as value of  $x_v$  in  $a$
- Thus,  $a'$  satisfies  $C$