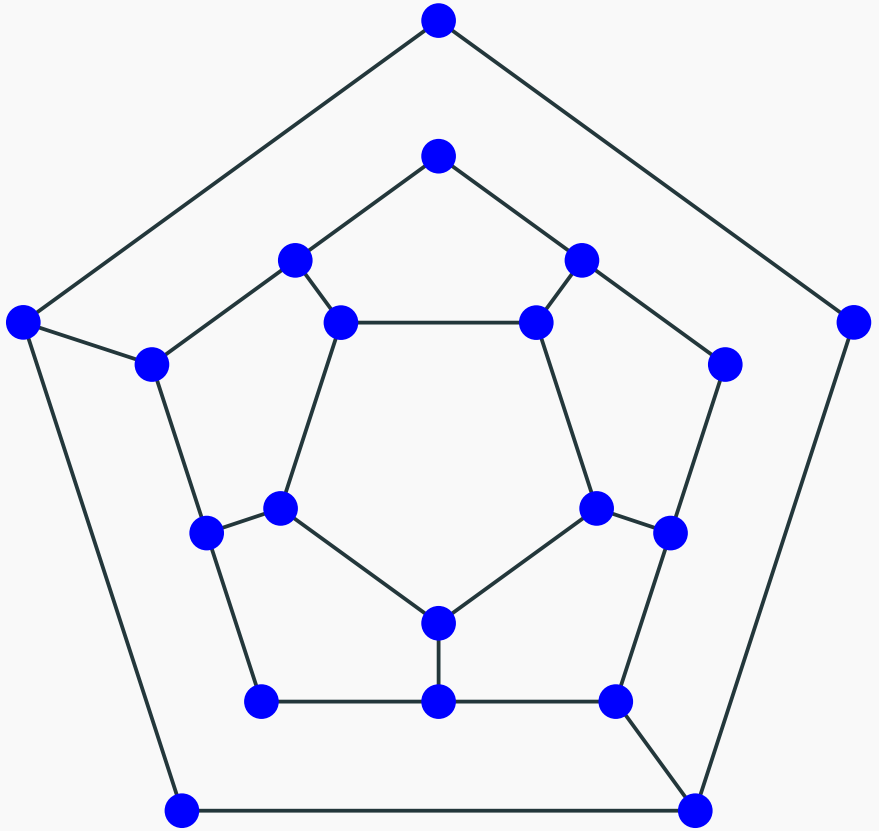
Pre-lecture brain teaser

Does this graph have a hamiltonian cycle?



a Yes.

b No.

ECE-374-B: Lecture 22 - Lots of NP-Complete reductions

Instructor: Nickvash Kani

April 13, 2023

University of Illinois at Urbana-Champaign

Today

NP-Completeness of two problems:

- Hamiltonian Cycle
- 3-Color

Important: understanding the problems and that they are hard.

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

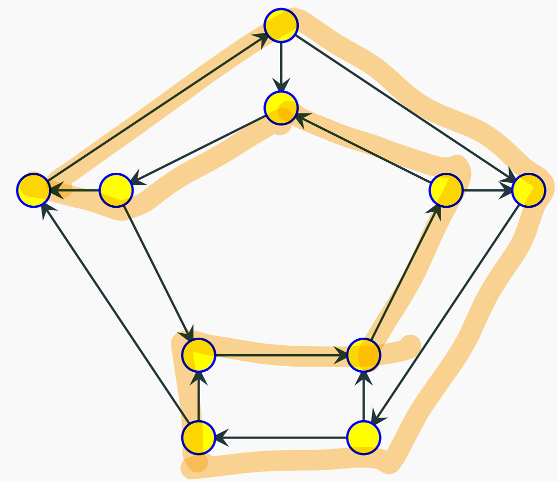
Reduction from 3SAT to Hamiltonian Cycle

Directed Hamiltonian Cycle

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

Goal Does G have a Hamiltonian cycle?

- 2- A Hamiltonian cycle is a cycle in the graph that visits every vertex in G exactly once

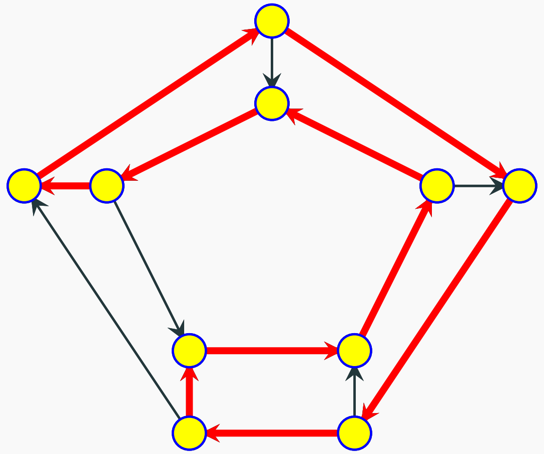


Directed Hamiltonian Cycle

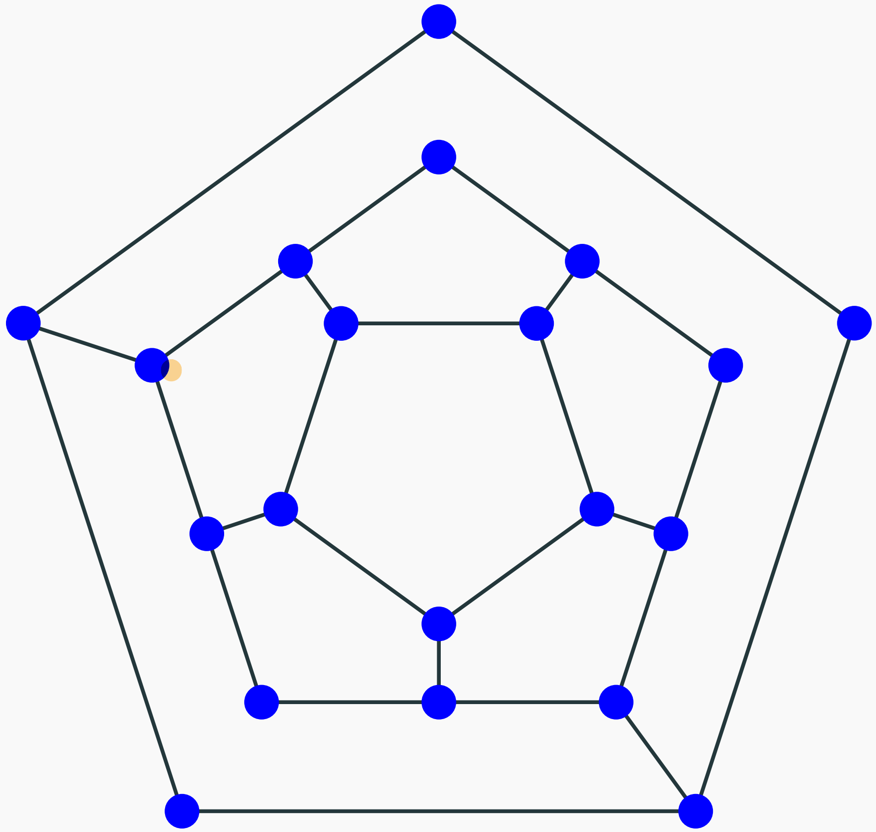
Input Given a directed graph $G = (V, E)$ with n vertices

Goal Does G have a Hamiltonian cycle?

- 2- A Hamiltonian cycle is a cycle in the graph that visits every vertex in G exactly once



Is the following graph Hamiltonianan?



a Yes.

b No.

Directed Hamiltonian Cycle is NP-Complete

- Directed Hamiltonian Cycle is in *NP*: exercise
- Hardness: We will show
  3-SAT $\leq_P$ Directed Hamiltonian Cycle

Certificate: cycle $\Rightarrow$ sequence of vertices

$c = [v_0, v_2, \ldots v_N]$

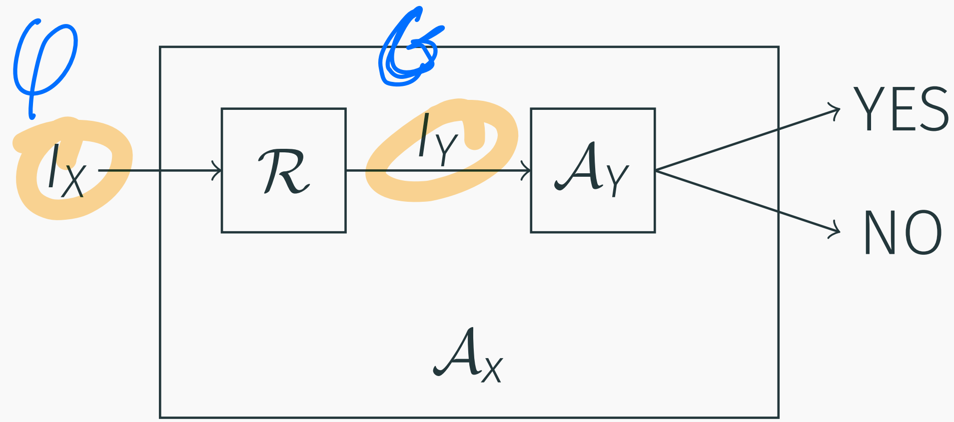
Certifier:

- check that every vertex in c is distinct
- check $|c| = n$
- for every $v_i \in c$, check $(v_i, v_{i+1}) \in E$
  $(v_N, v_1) \in E$

Directed Hamiltonian Cycle is NP-Complete

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

  3-SAT $\leq_P$ Directed Hamiltonian Cycle



X = 3-SAT
Y = HC

A_X = algorithm that solves 3SAT
A_Y = algorithm that solves HC
I_X = instance of 3SAT φ
I_Y = instance of HC

Reduction

Given 3-SAT formula φ create a graph G_φ such that

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

Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction: First Ideas

- Viewing SAT: Assign values to n variables, and each clauses 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 expression:

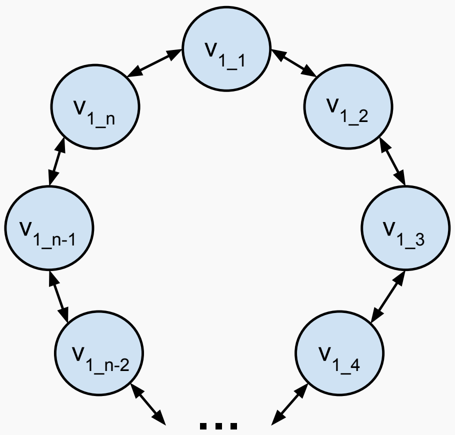
$$f(x_1) = 1 \tag{1}$$

Reduction: Encoding idea I

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

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

We create a cyclic graph that always has a hamiltonian:

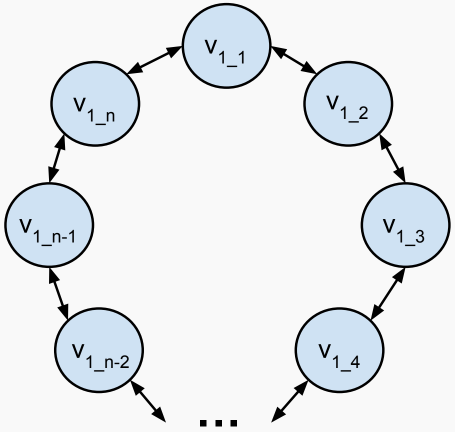


Reduction: Encoding idea I

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

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

We create a cyclic graph that always has a hamiltonian:



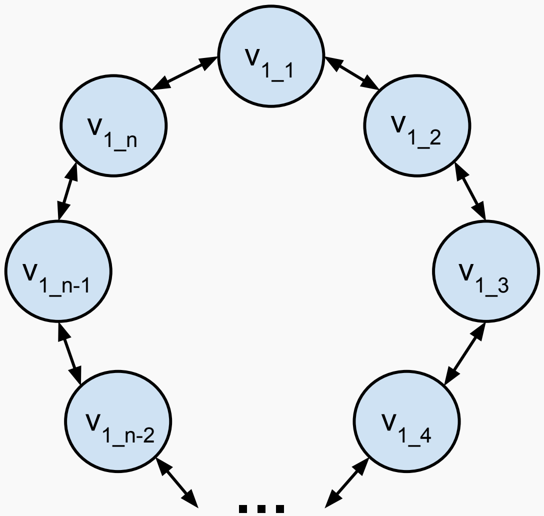
But how do we encode the variable?

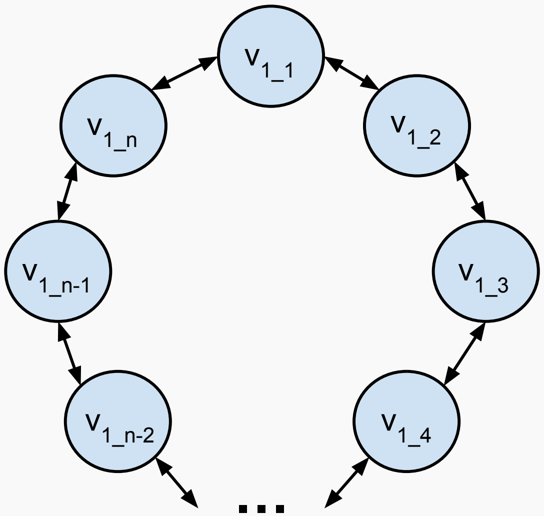
Reduction: Encoding idea I

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

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

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



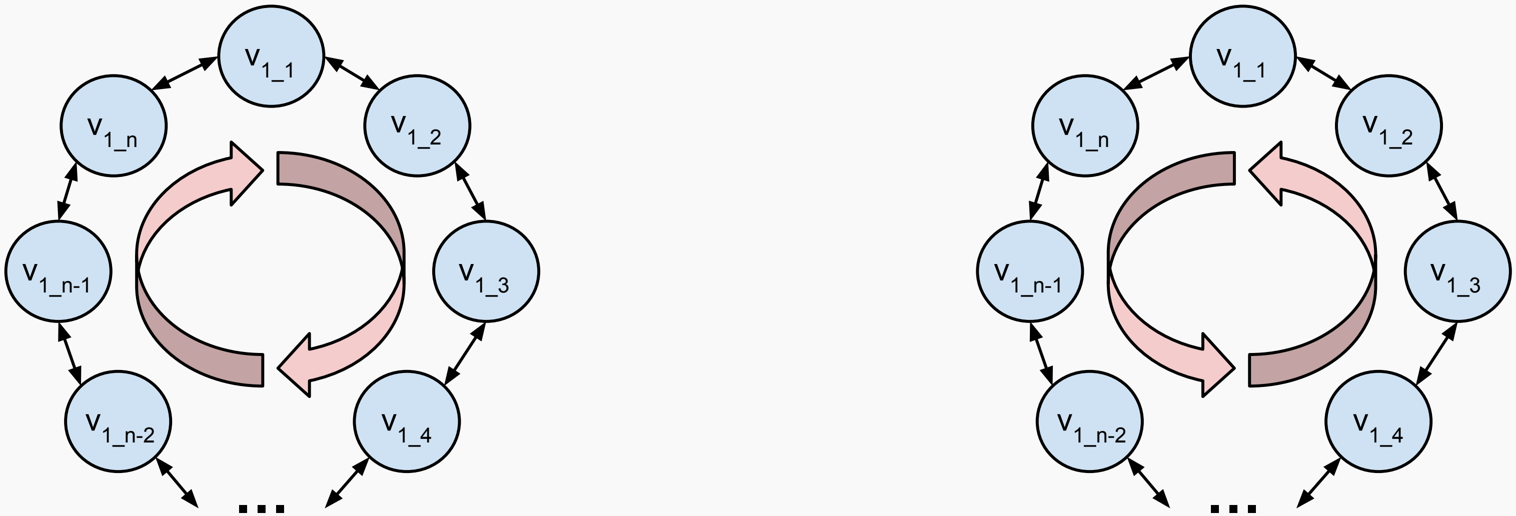


Reduction: Encoding idea I

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

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

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?

$$f(x_1, x_2) = 1 \tag{3}$$

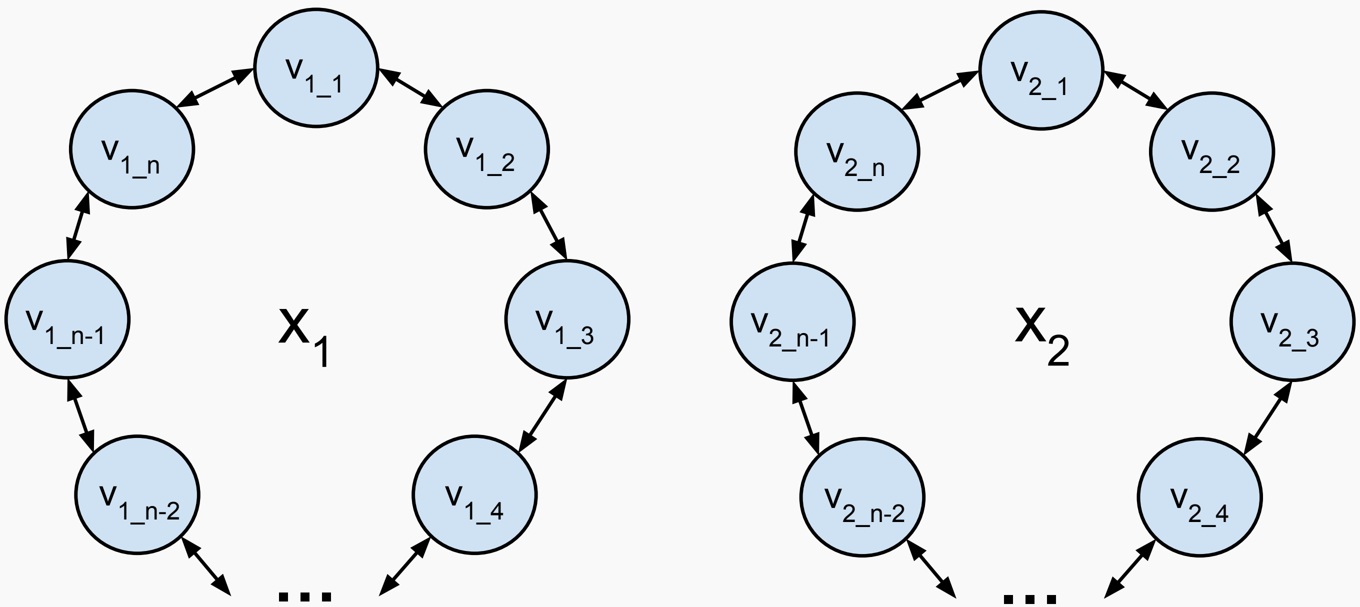
Maybe two circles? Now we need to connect them so that we have a single hamiltonian path

Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{3}$$

Maybe two circles? Now we need to connect them so that we have a single hamiltonian path

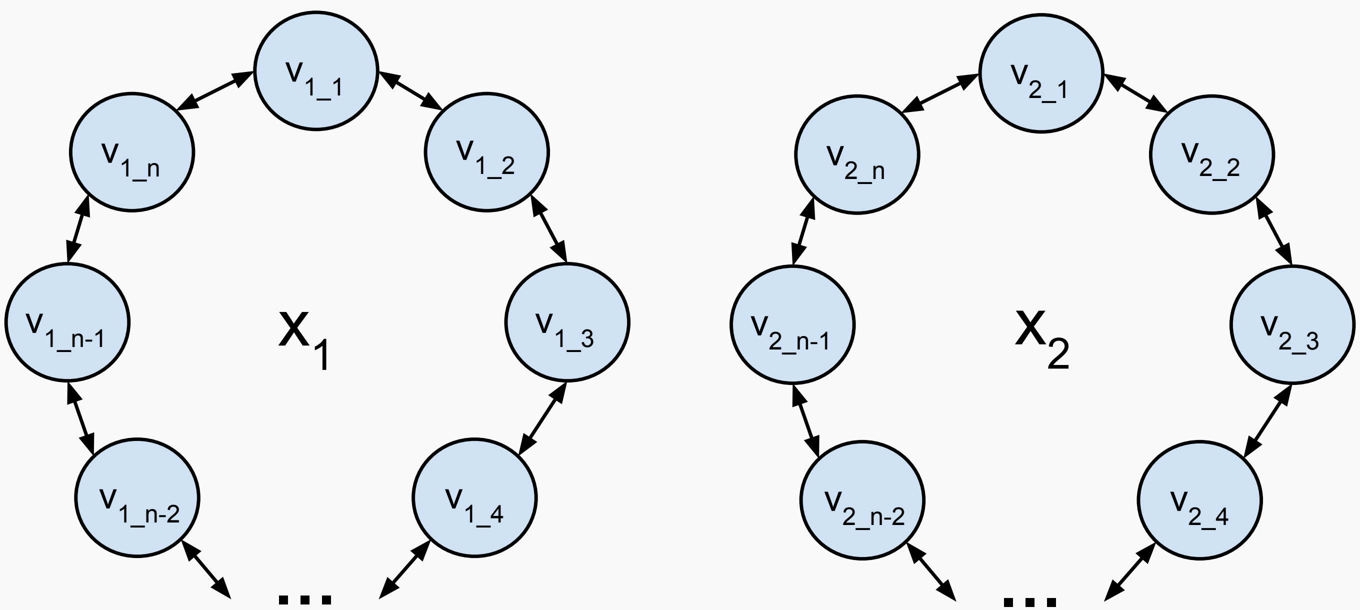


Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{4}$$

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

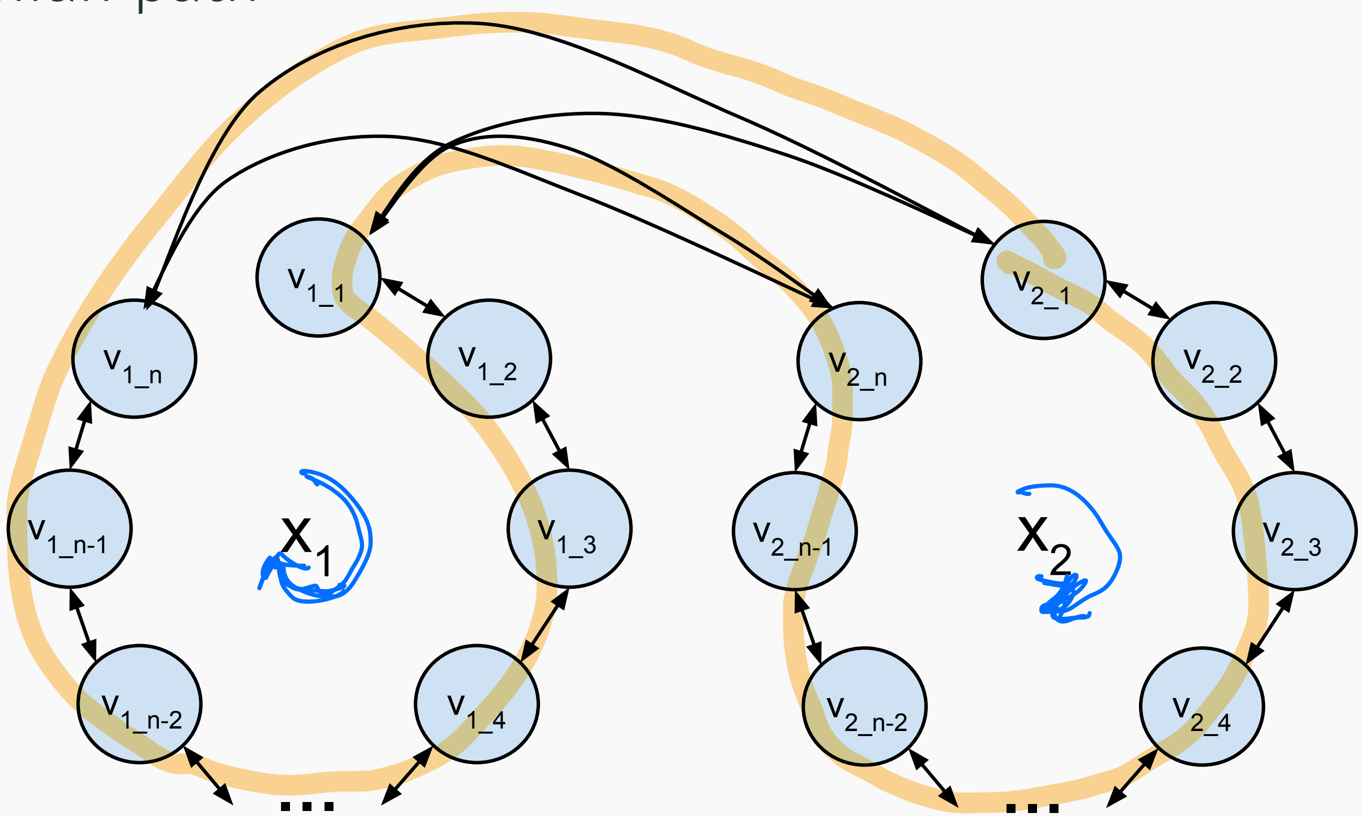


Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{4}$$

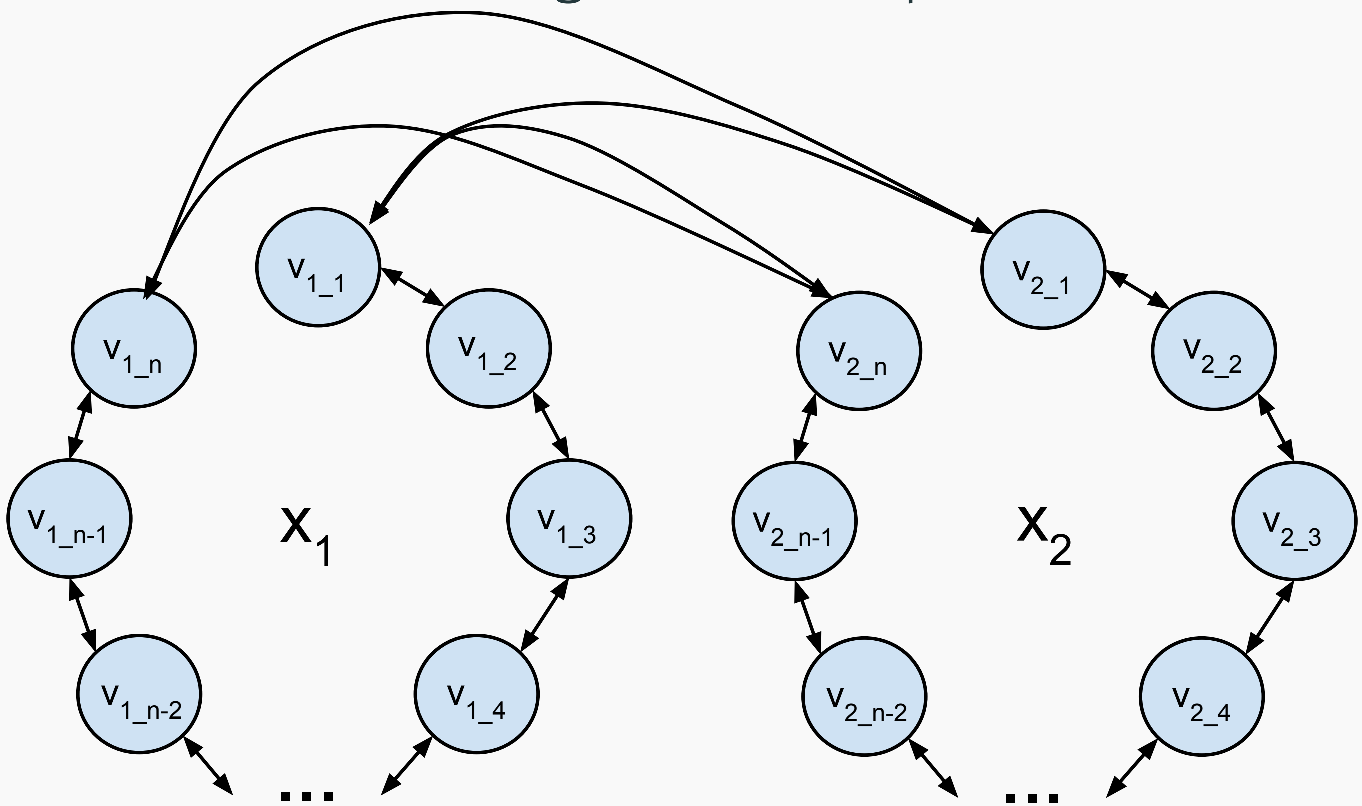
Now we need to connect them so that we have a single hamiltonian path



How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{5}$$

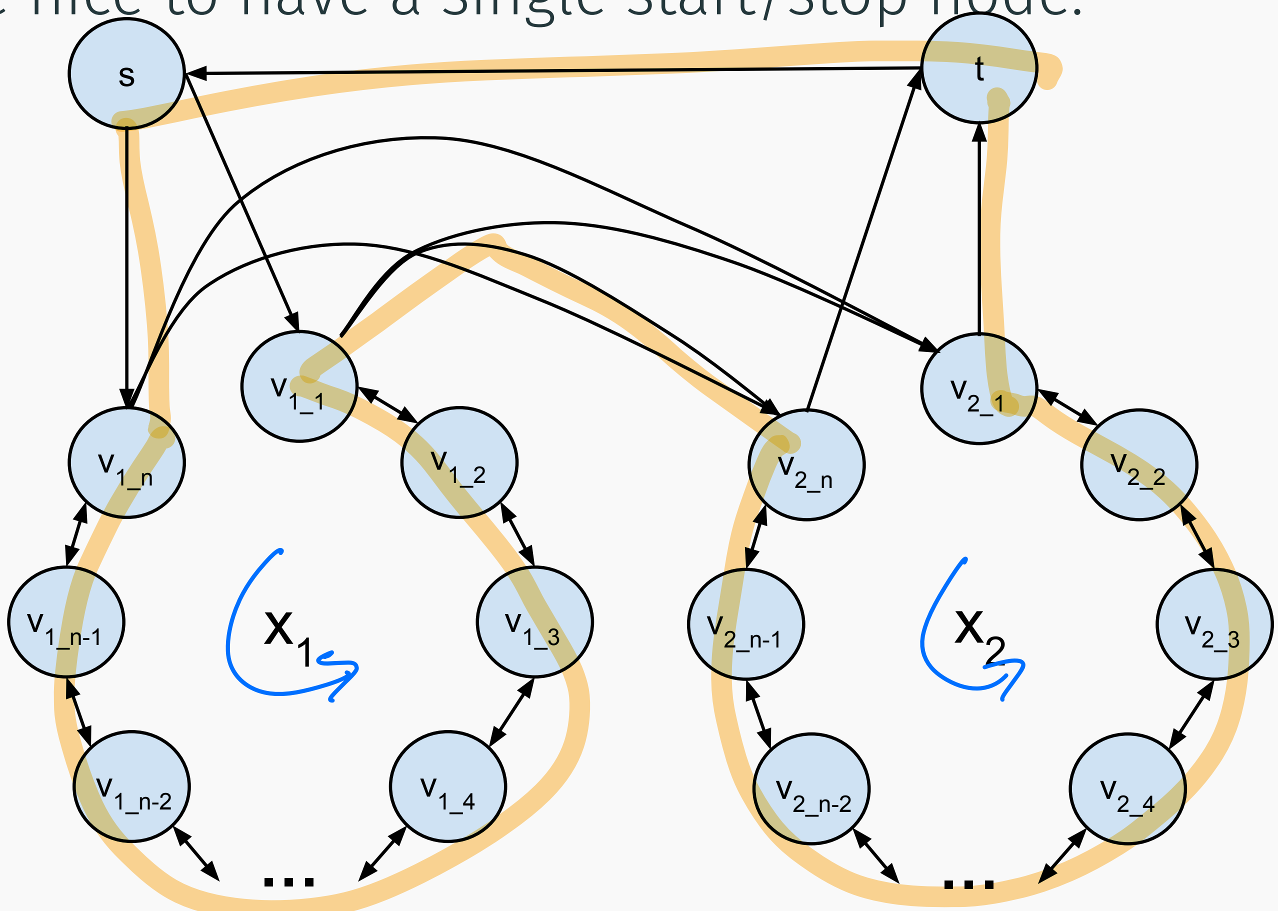
Would be nice to have a single start/stop node.



How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{5}$$

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

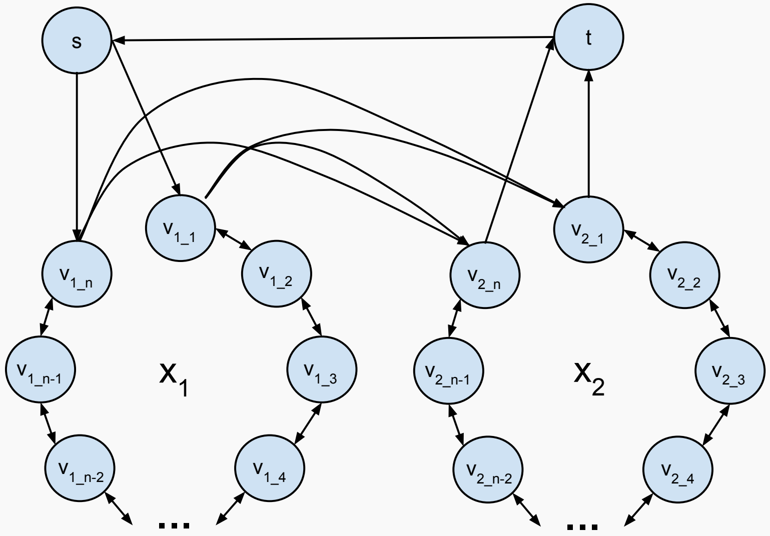


Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{6}$$

Getting a bit messy. Let's reorganize:

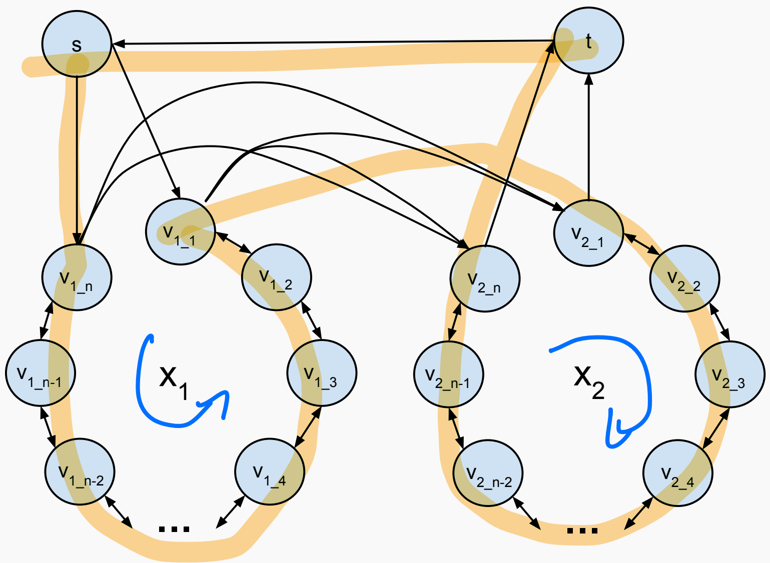


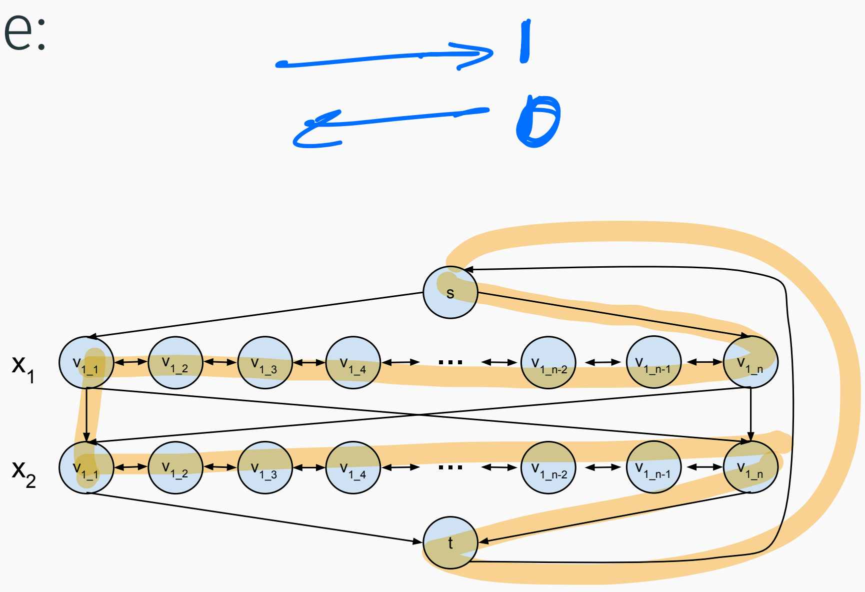
Reduction: Encoding idea II

How do we encode multiple variables?

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

Getting a bit messy. Let's reorganize:



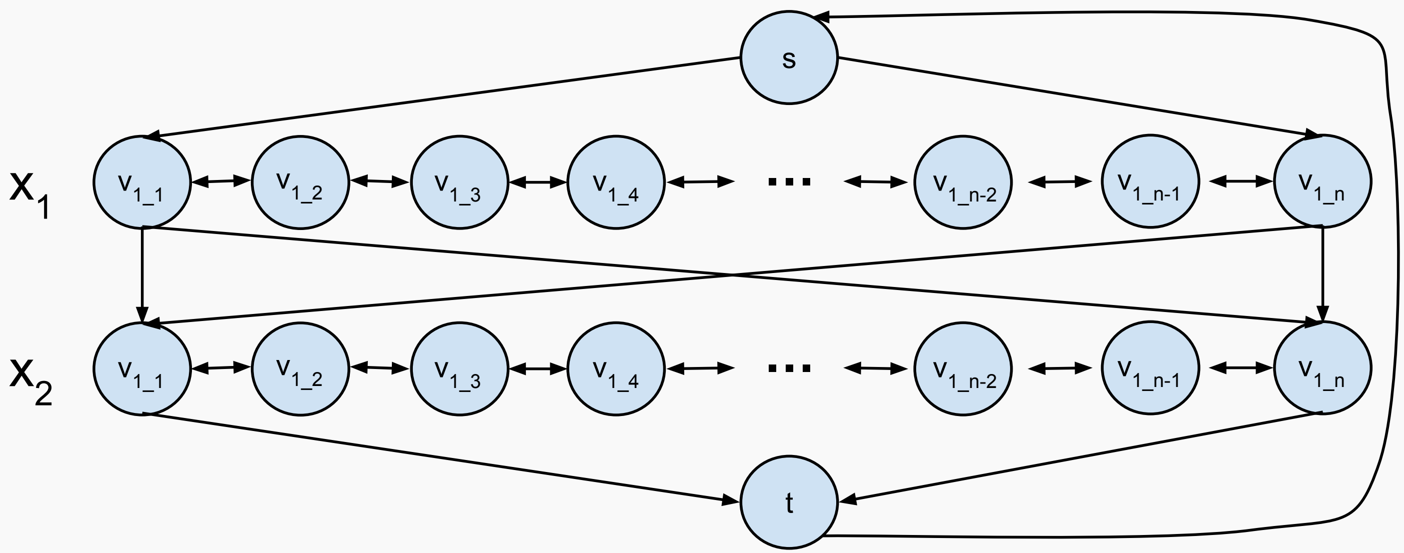


Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{7}$$

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



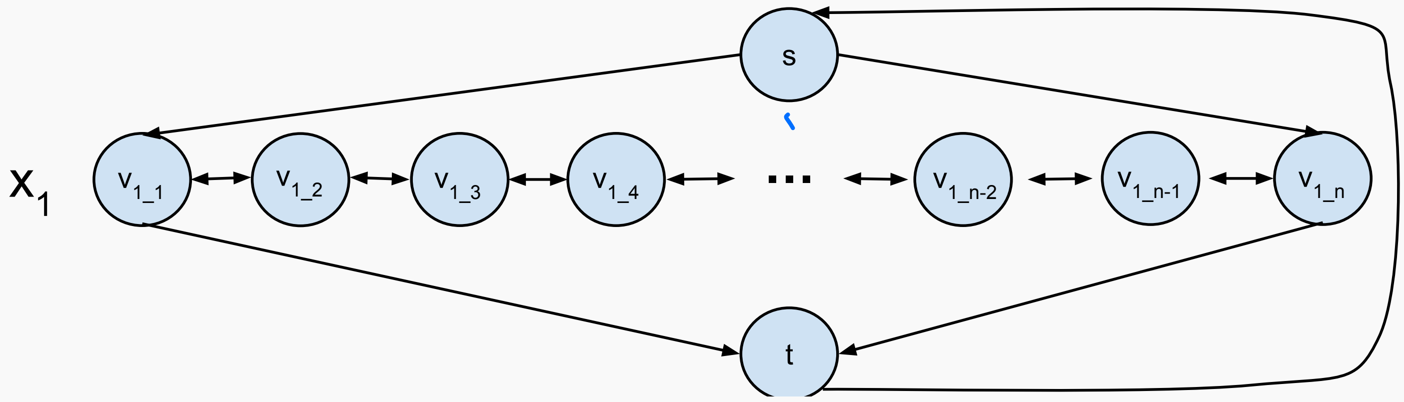
Reduction: Encoding idea III

How do we handle clauses?

$$f(x_1) = (x_1) \tag{8}$$

C_1

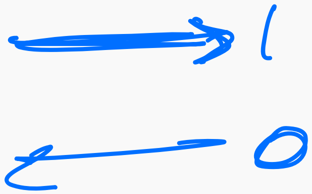
Lets go back to our one variable graph:



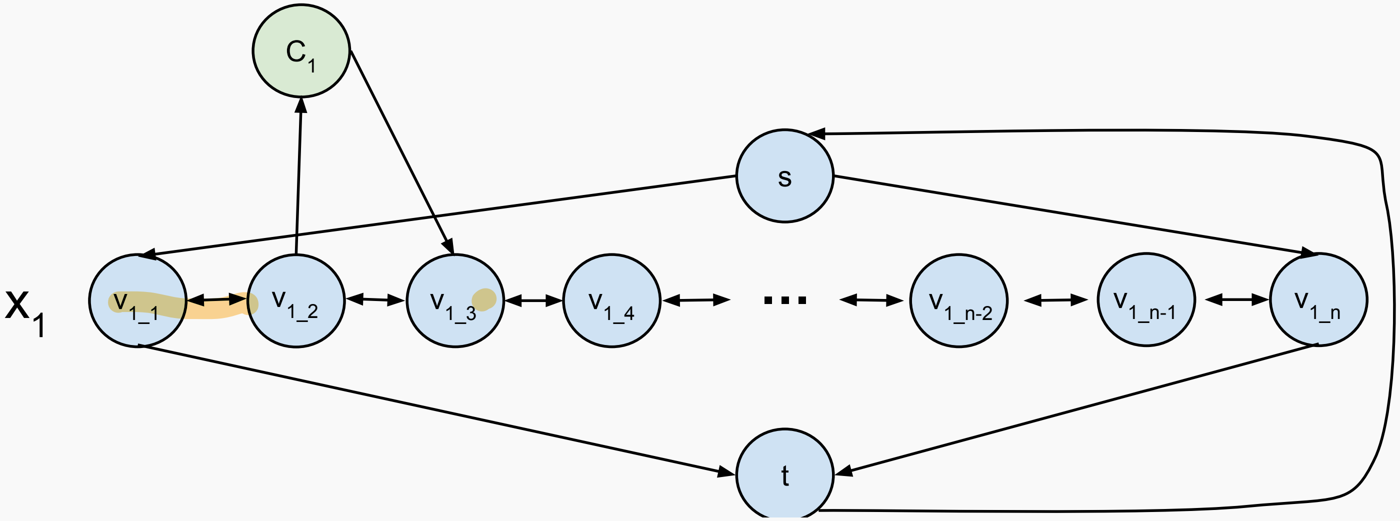
Reduction: Encoding idea III

How do we handle clauses?

$$f(x_1) = x_1 \tag{9}$$



Add node for clause:

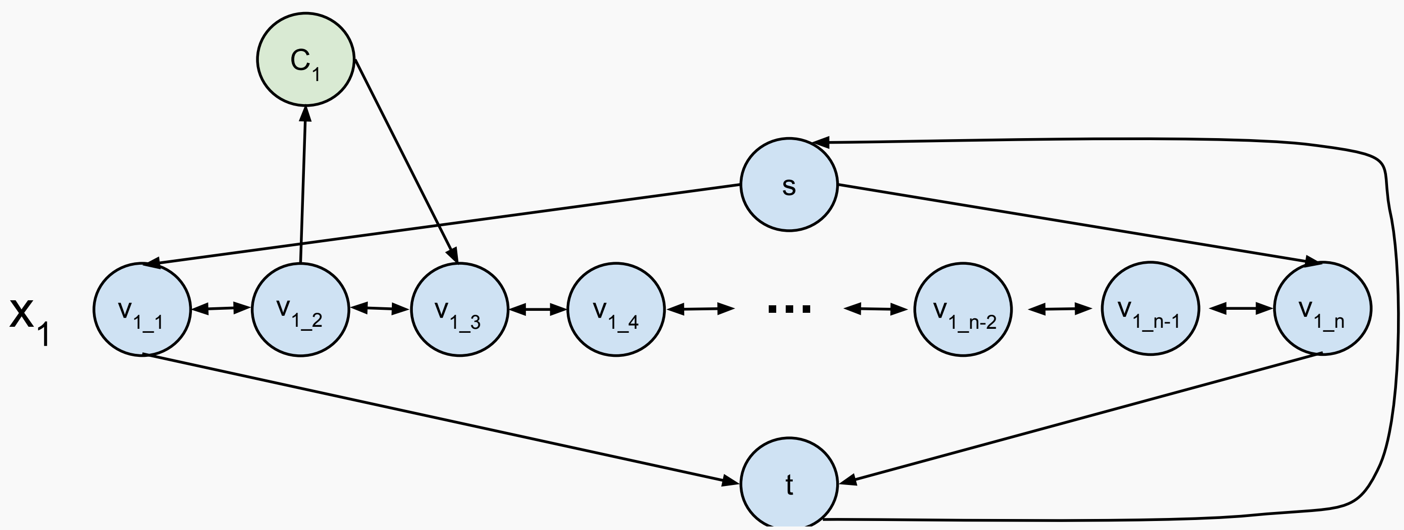


Reduction: Encoding idea III

How do we handle clauses?

$$f(x_1, x_2) = (x_1 \vee \overline{x_2}) \tag{10}$$

What do we do if the clause has two literals:

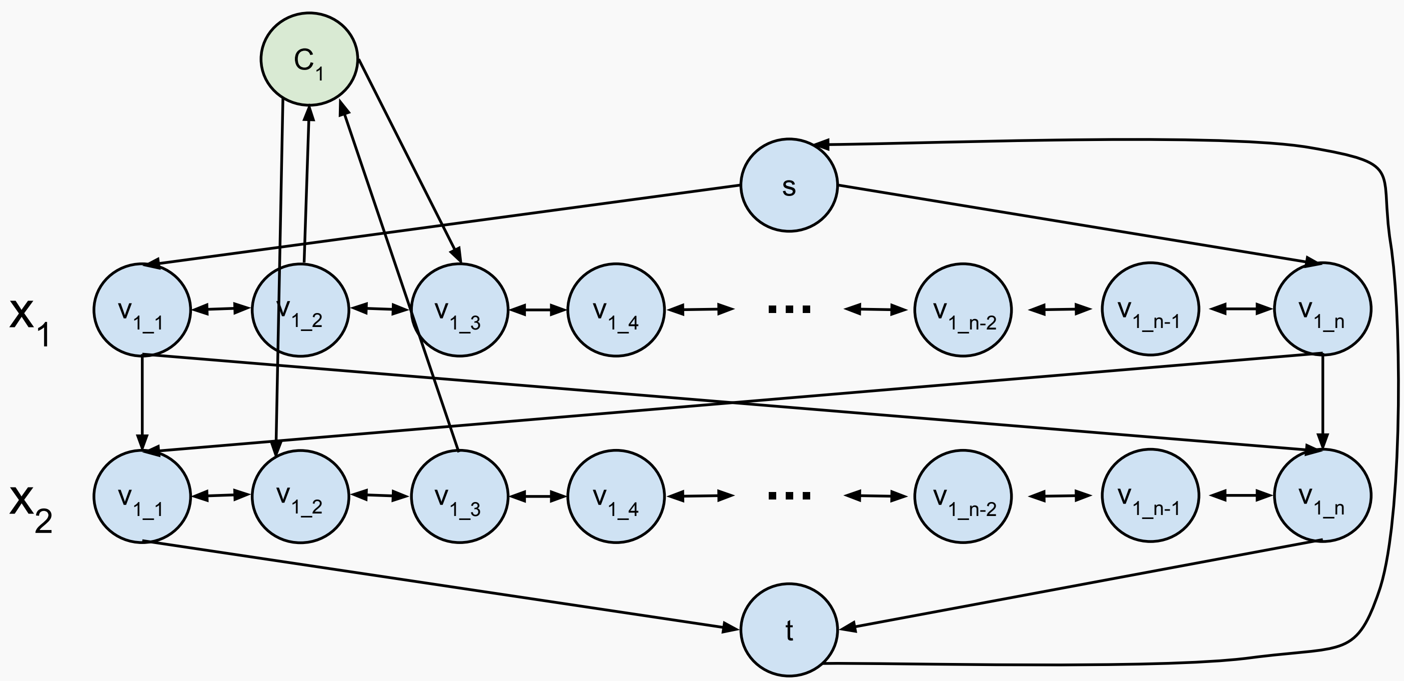


Reduction: Encoding idea III

How do we handle clauses?

$$f(x_1, x_2) = \underbrace{(x_1 \vee \overline{x_2})}_{C_1} \tag{10}$$

What do we do if the clause has two literals:

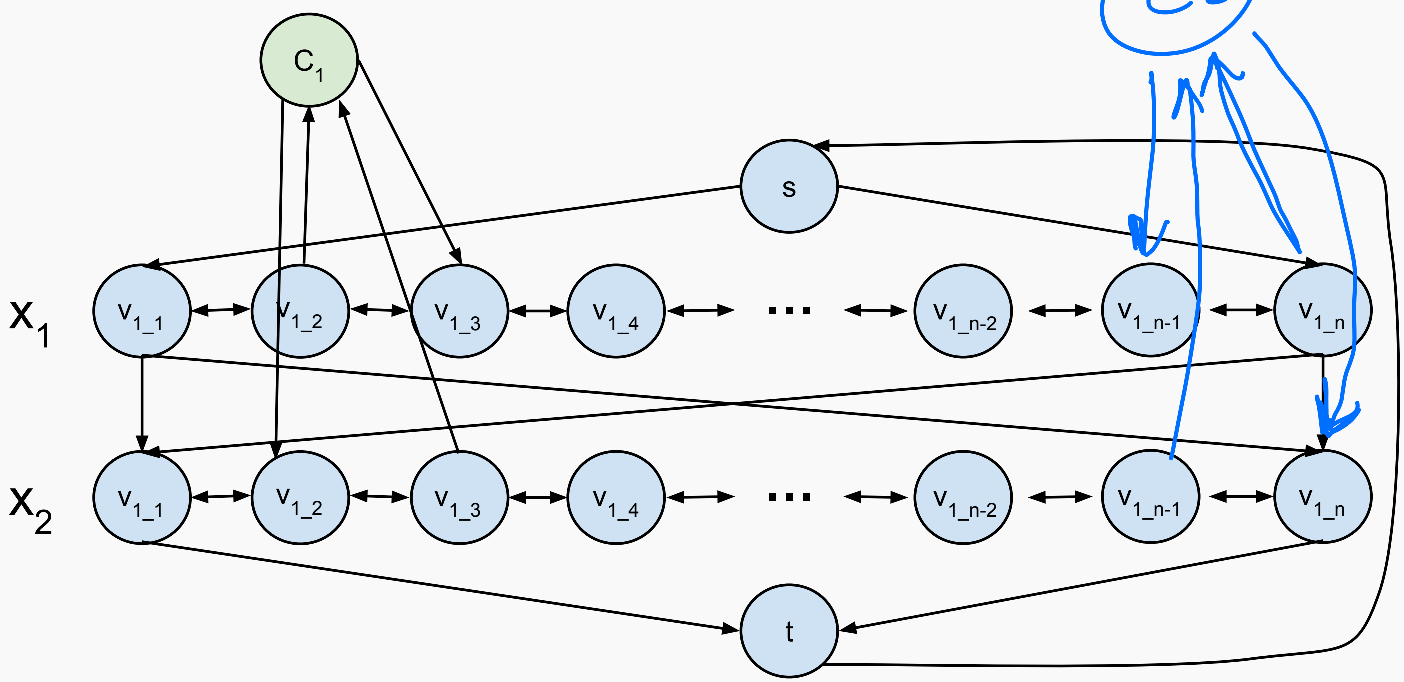


Reduction: Encoding idea III

How do we handle clauses?

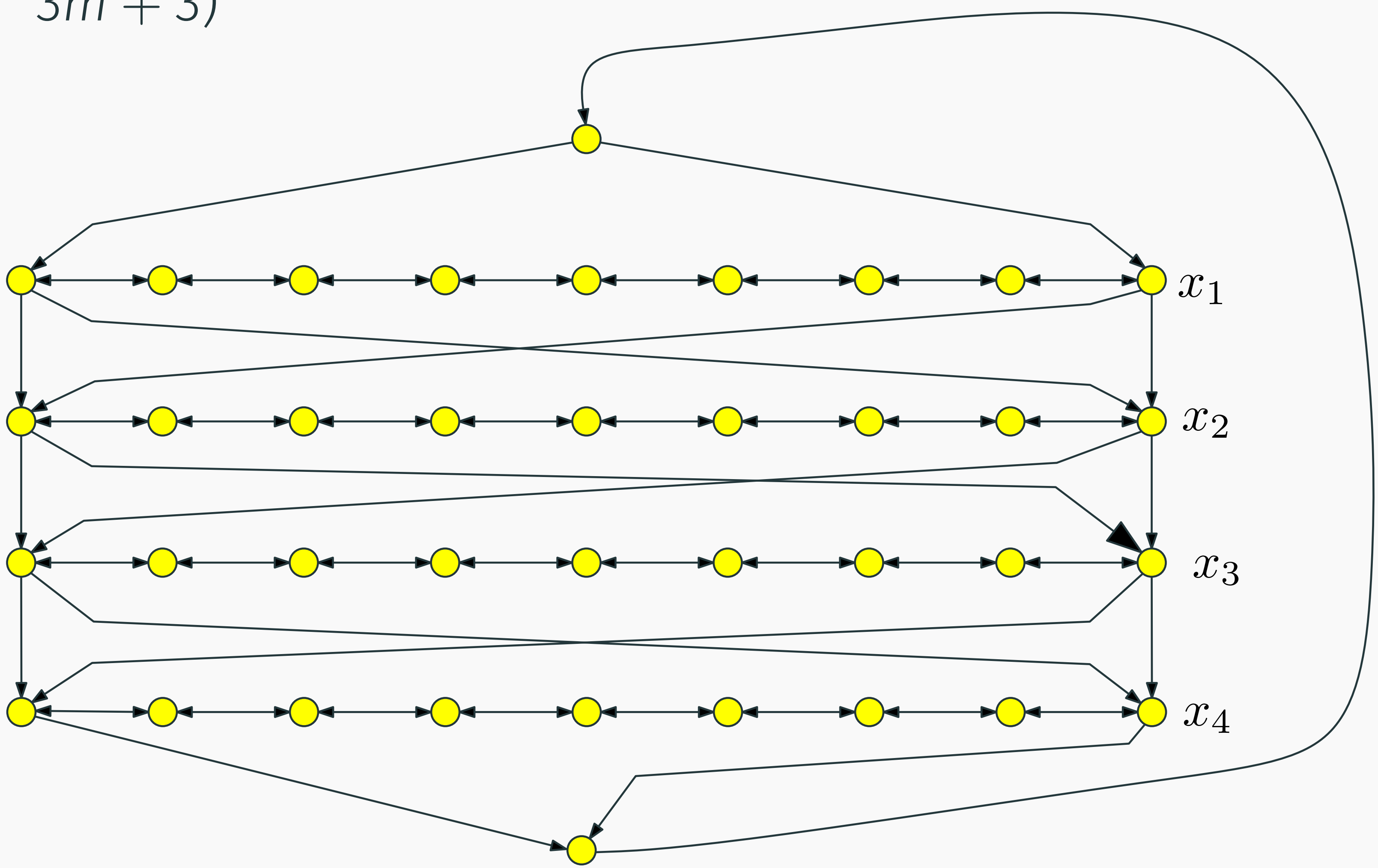
$$f(x_1, x_2) = (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \tag{11}$$

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 φ; nodes numbered from left to right (1 to $3m+3$)

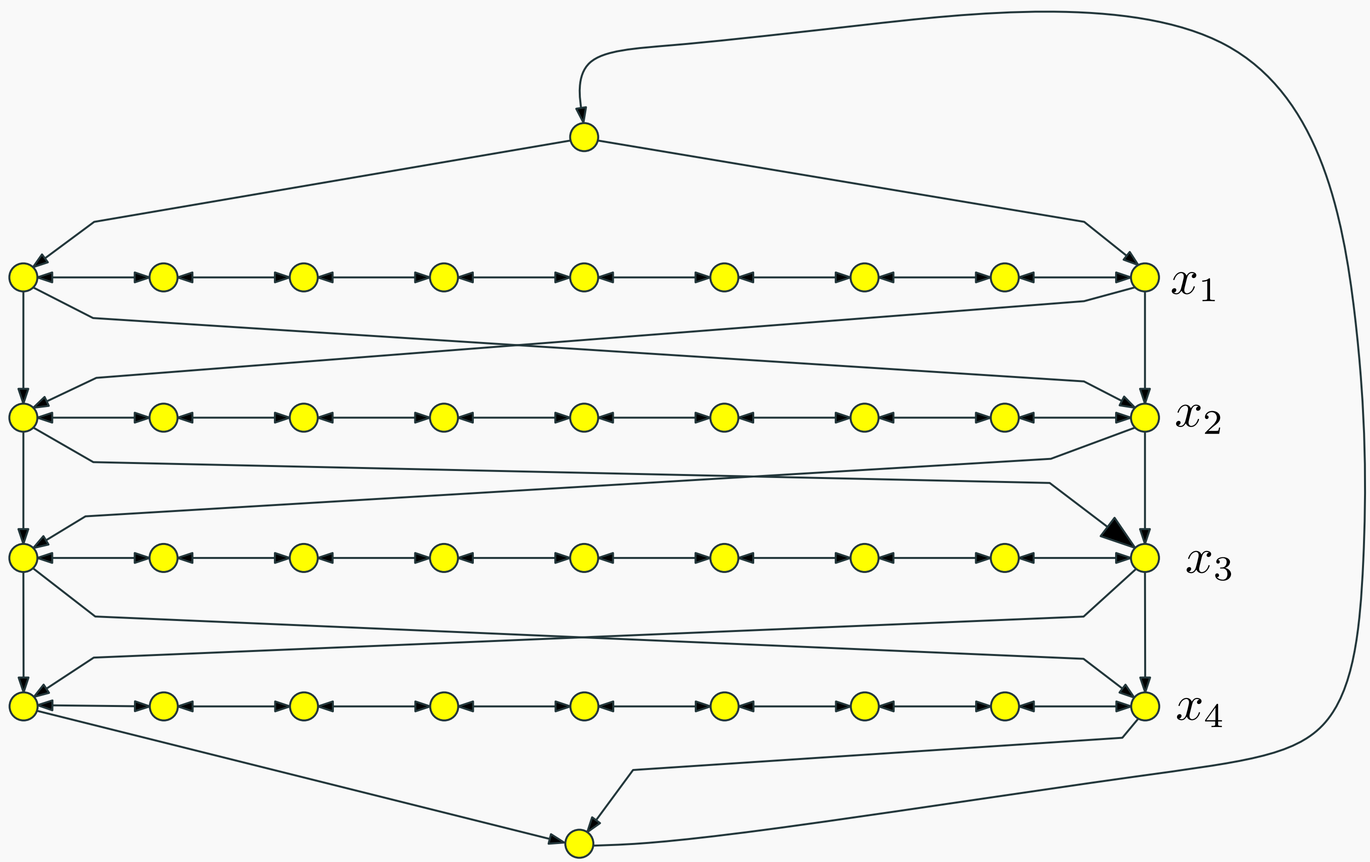


The Reduction algorithm: Review II

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.

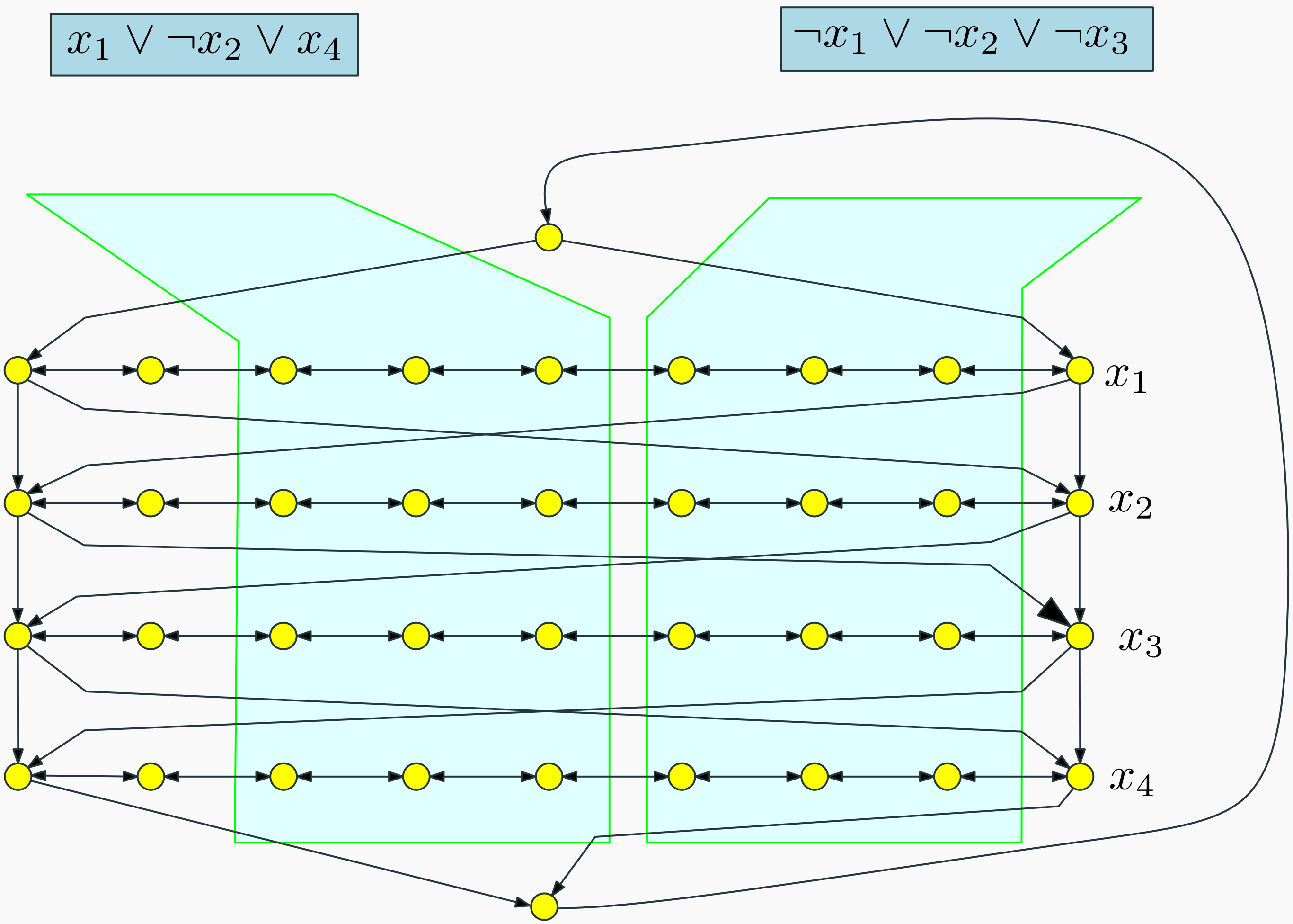
$x_1 \vee \neg x_2 \vee x_4$

$\neg x_1 \vee \neg x_2 \vee \neg x_3$



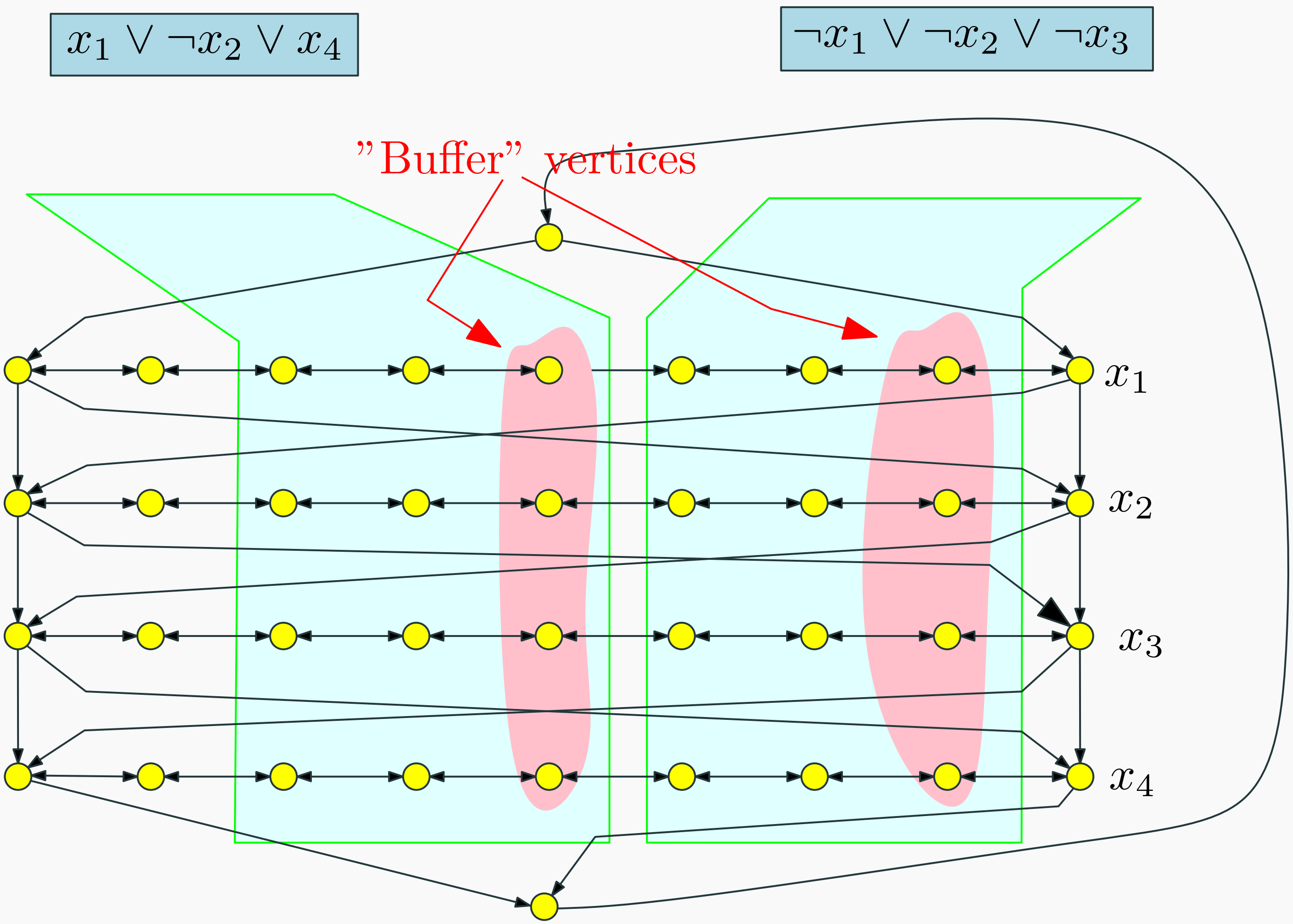
The Reduction algorithm: Review II

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.



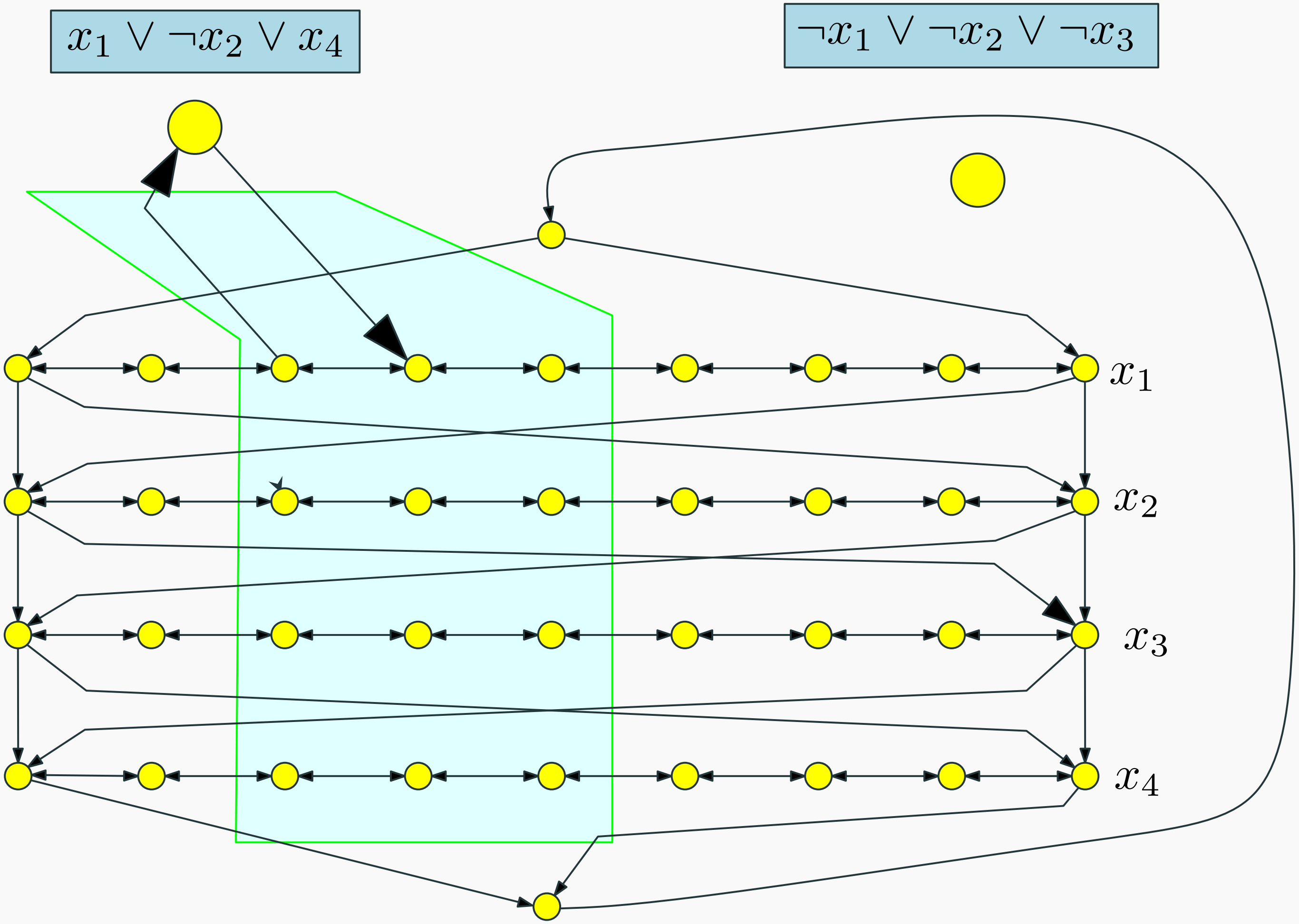
The Reduction algorithm: Review II

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.



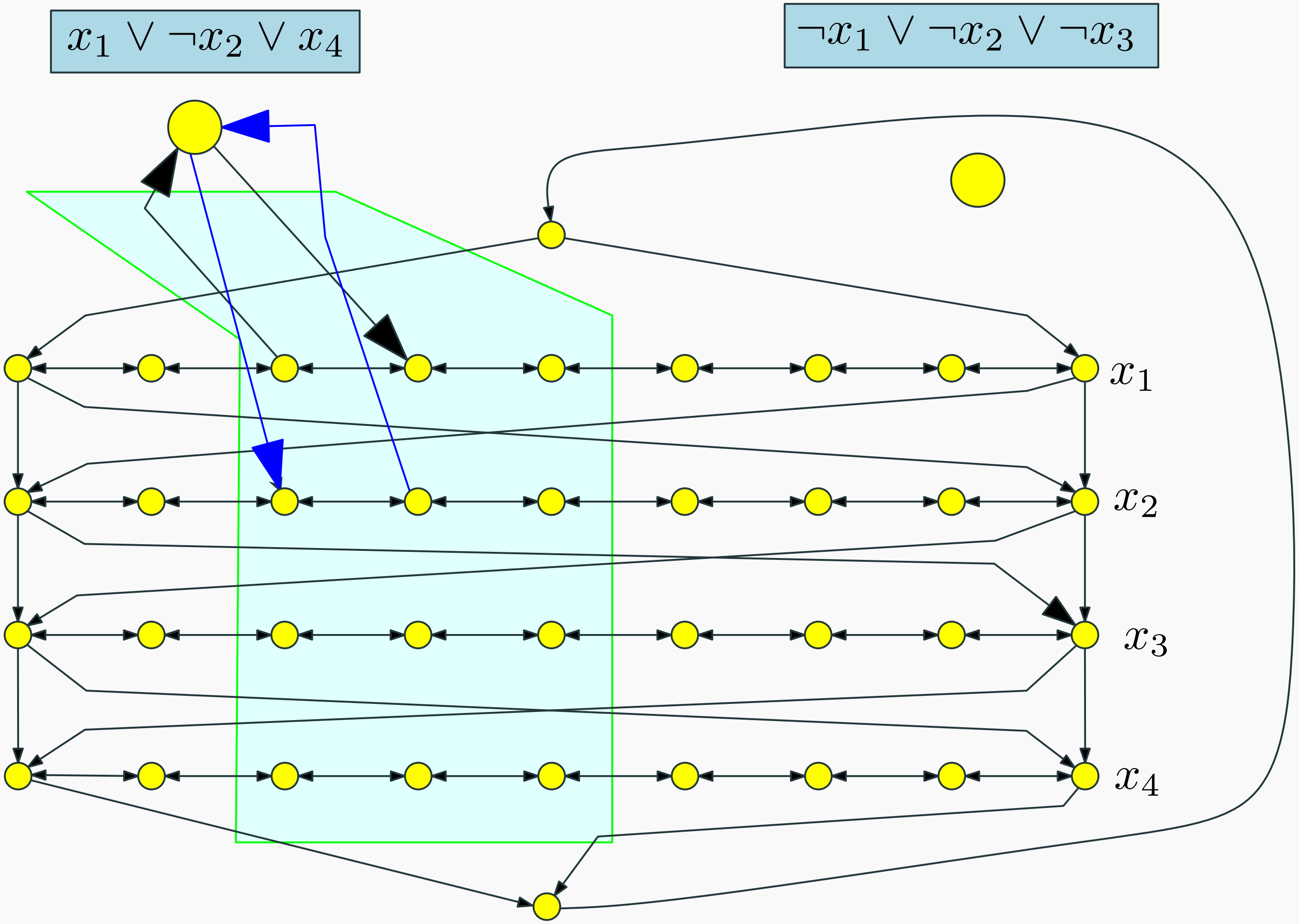
The Reduction algorithm: Review II

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.

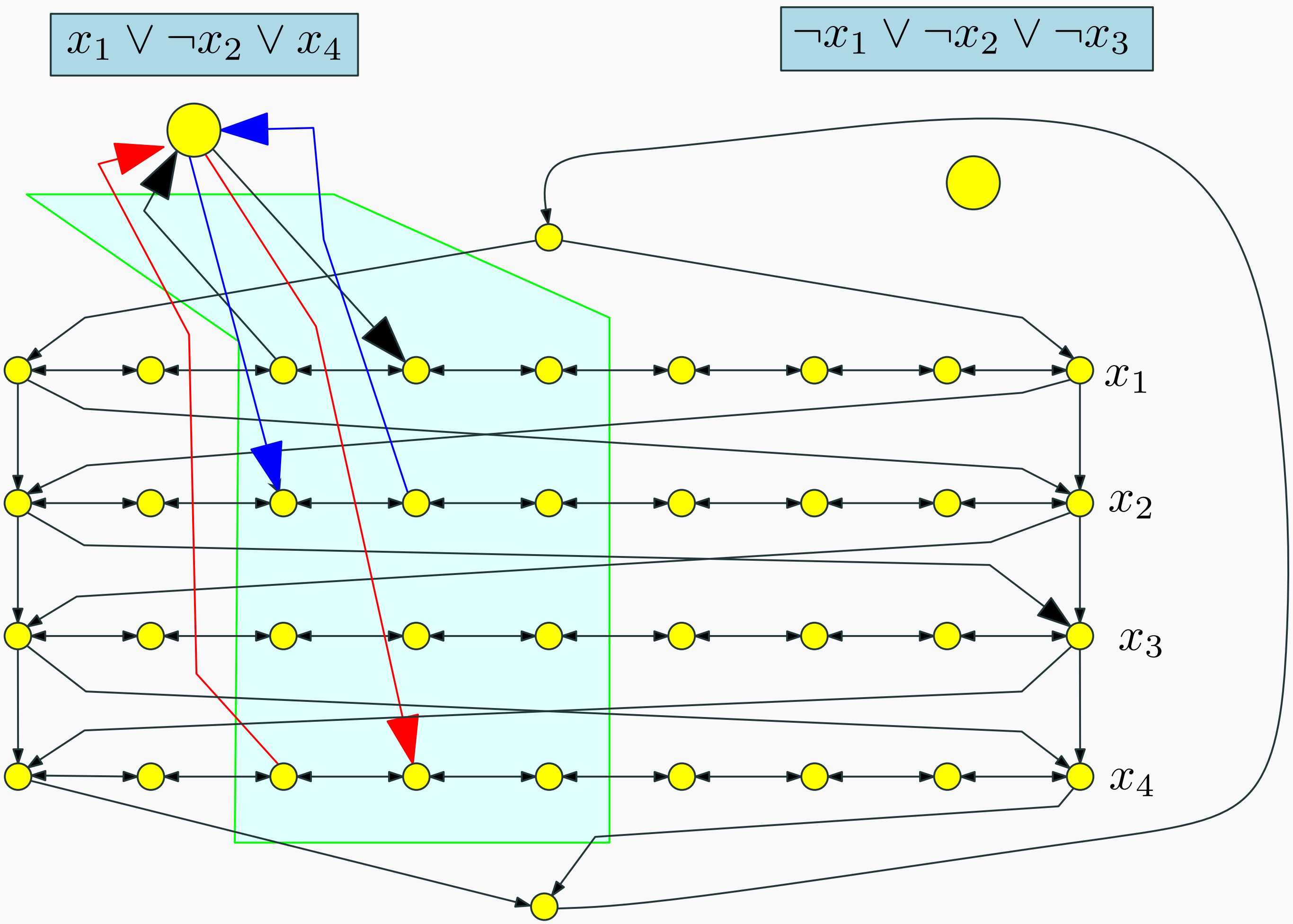


The Reduction algorithm: Review II

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.

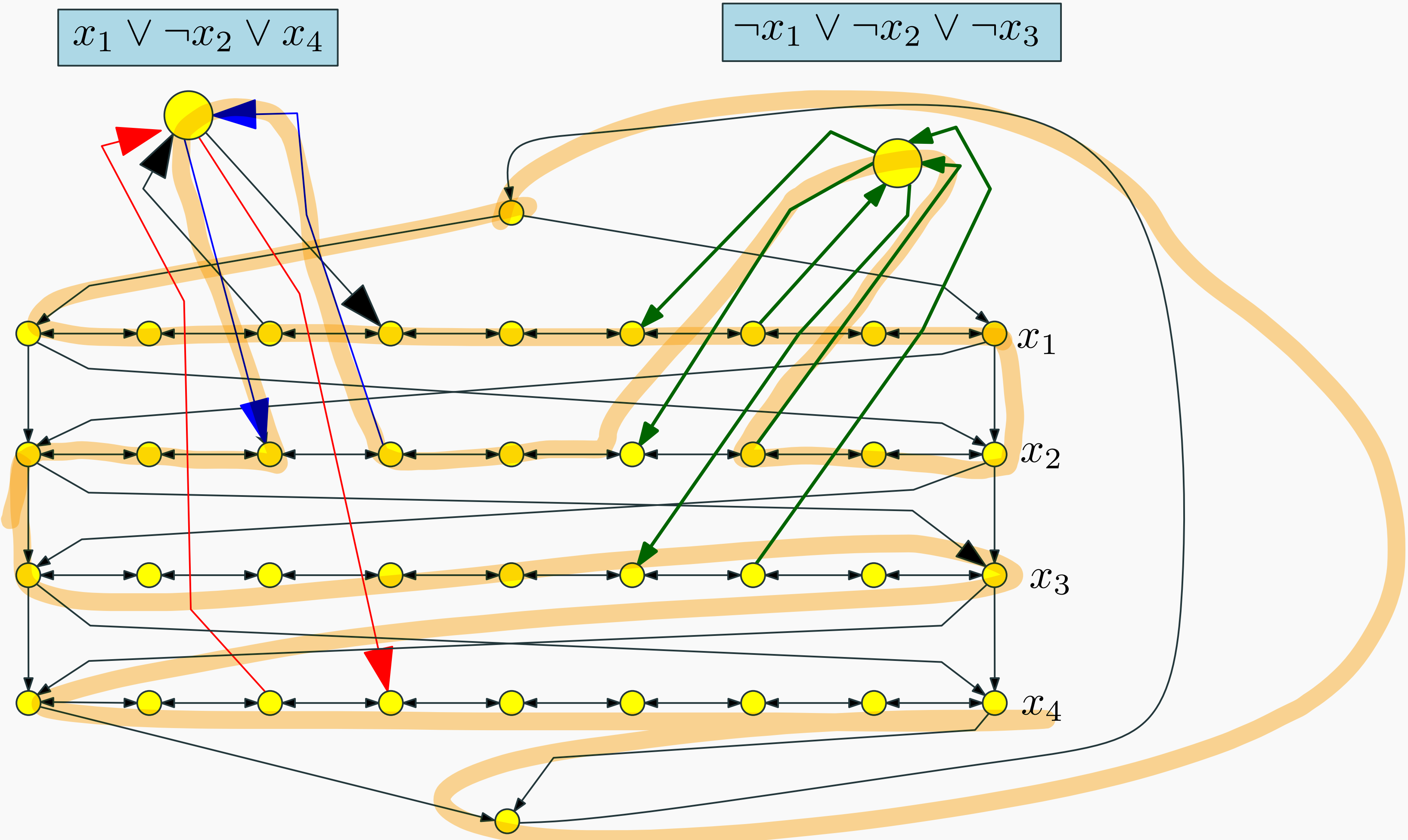


Add vertex c_j for clause C_j. c_j has edge from vertex $3j$ and to vertex $3j + 1$ on path i if x_i appears in clause C_j, and has edge from vertex $3j + 1$ and to vertex $3j$ if $\neg x_i$ appears in C_j.



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
φ has a satisfying assignment iff G_φ has a Hamiltonian cycle.

Based on proving following two lemmas.

Lemma
If φ has a satisfying assignment then G_φ has a Hamilton cycle.

Lemma
If G_φ has a Hamilton cycle then φ has a satisfying assignment.

Satisfying assignment $\rightarrow$ Hamiltonian Cycle

Lemma
If φ has a satisfying assignment then G_φ has a Hamilton cycle.

Proof.

$\Rightarrow$ Let a be the satisfying assignment for φ. 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 Π is a Hamiltonian cycle in G_φ

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

Hamiltonian Cycle $\rightarrow$ Satisfying assignment

Suppose Π is a Hamiltonian cycle in G_φ

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

Lemma
Every Hamilton cycle in G_φ is canonical.

Proof of Lemma

Lemma
Every Hamilton cycle in G_φ is canonical.

- If Π 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 Π 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 $\Longrightarrow$ Satisfying assignment (contd)

Lemma
Any canonical Hamilton cycle in G_φ corresponds to a satisfying truth assignment to φ.

Consider a canonical Hamilton cycle Π.

- 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 Π in $G - \{c_1, \ldots c_m\}$ traverses each path in only one direction, which determines truth assignment
- Easy to verify that this truth assignment satisfies φ

Hamiltonian cycle in undirected graph

Hamiltonian Cycle in Undirected Graphs

Problem

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

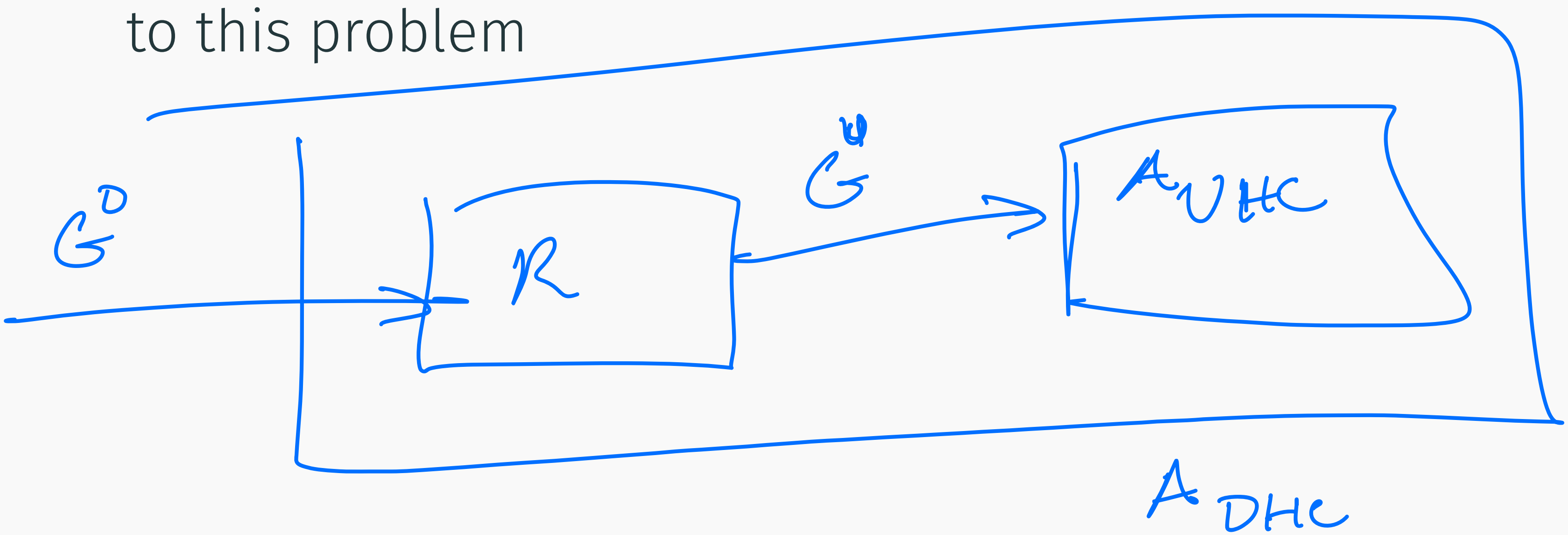
Goal *Does G have a Hamiltonian cycle? That is, is there a cycle that visits every vertex exactly one (except start and end vertex)?*

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

$DHC \le_p UHC$

Proof.

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

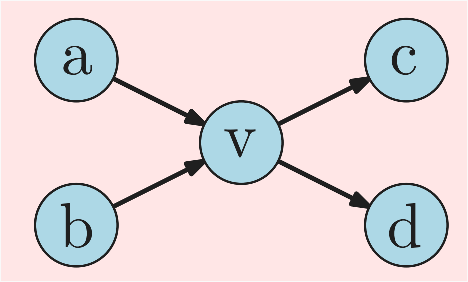


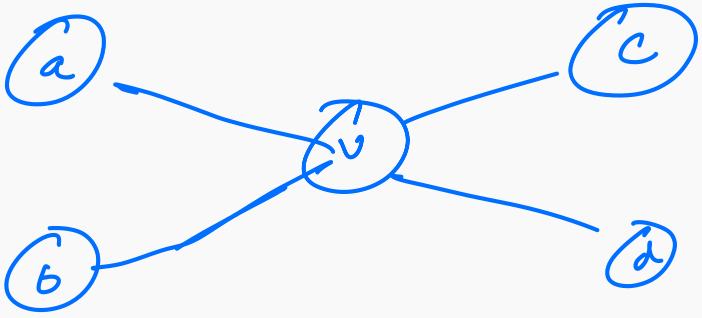
Reduction Sketch

Goal: Given directed graph G, need to construct undirected graph G' such that G has Hamiltonian Path iff G' has Hamiltonian path

Reduction

-
-



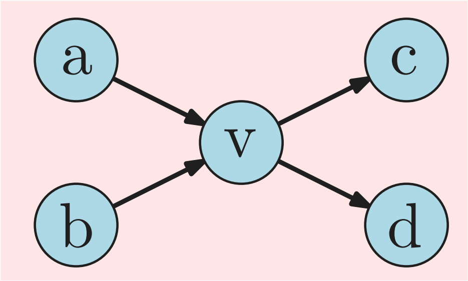


Reduction Sketch

Goal: Given directed graph G, need to construct undirected graph G' such that G has Hamiltonian Path iff G' has Hamiltonian path

Reduction

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

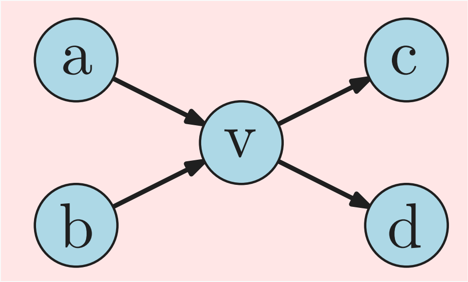


Reduction Sketch

Goal: Given directed graph G, need to construct undirected graph G' such that G has Hamiltonian Path 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})

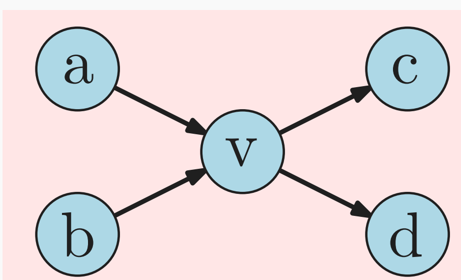


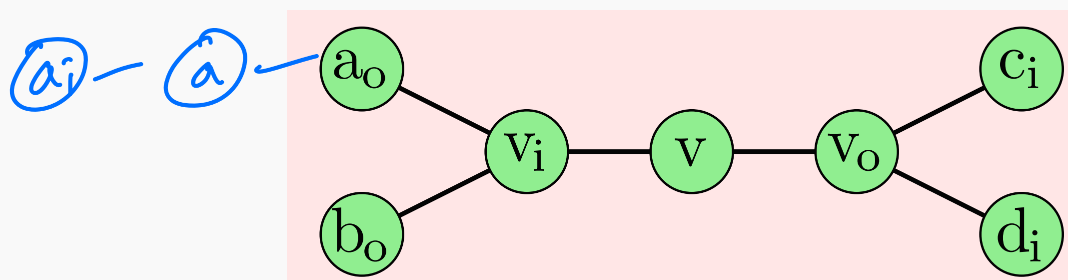
Reduction Sketch

Goal: Given directed graph G, need to construct undirected graph G' such that G has Hamiltonian Path iff G' has Hamiltonian path

Reduction

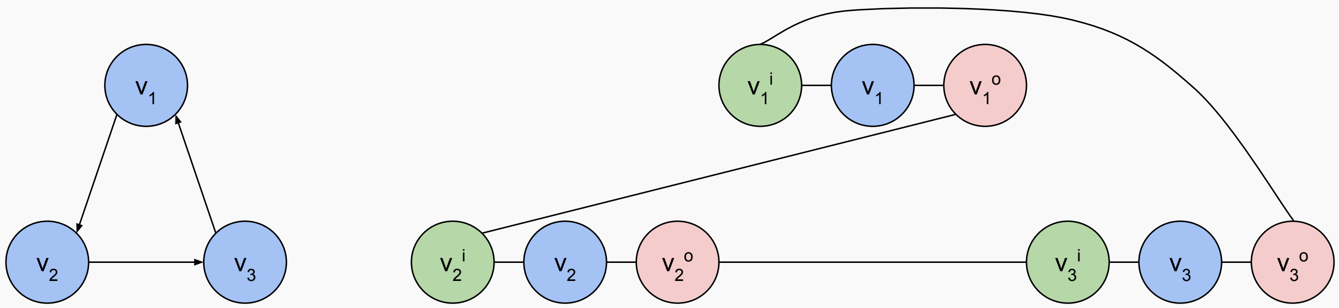
- Replace each vertex v by 3 vertices: v_{in}, v, and v_{out}
- A directed edge (a, b) is replaced by edge (a_{out}, b_{in})





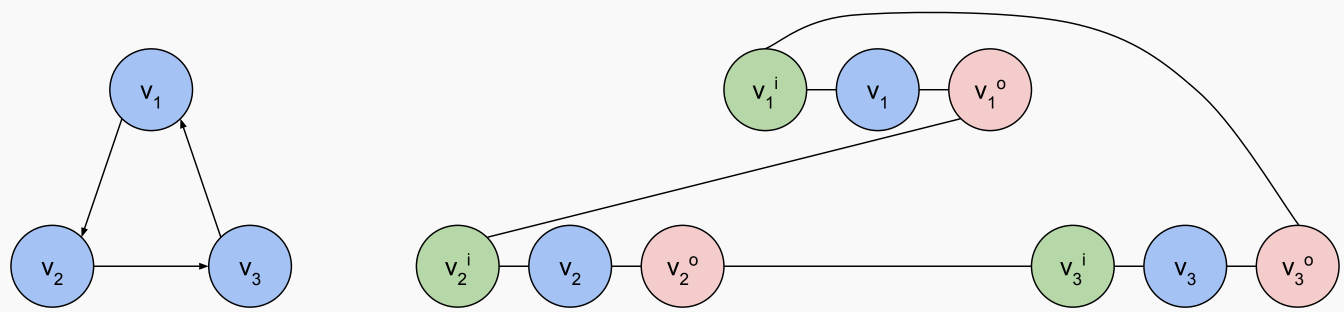
Reduction Sketch Example

Graph with cycle:

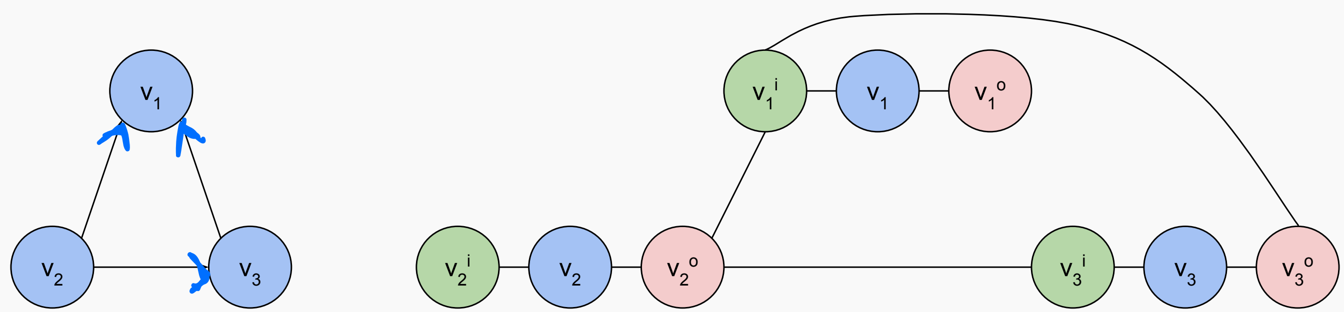


Reduction Sketch Example

Graph <u>with</u> cycle:



Graph <u>without</u> cycle:



Reduction: Wrapup

- The reduction is polynomial time (exercise)
- The reduction is correct (exercise)

$3SAT \leq_p DHC \leq_p UHC$

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 **Halitonian Cycle** or do a reduction from **Halitonian 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 **Halitonian Cycle** or do a reduction from **Halitonian Cycle**

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

NP-Completeness of Graph Coloring

Graph Coloring

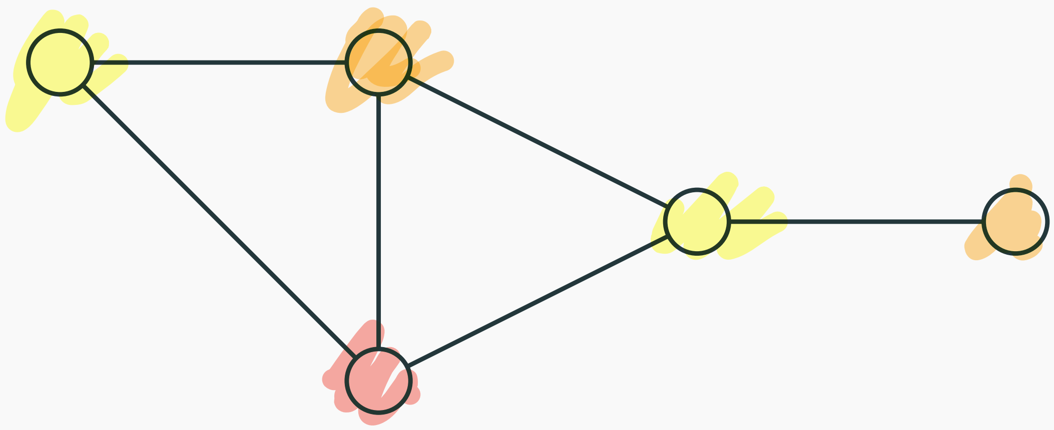
Problem: Graph Coloring

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?

k = 3

Problem: 3 Coloring

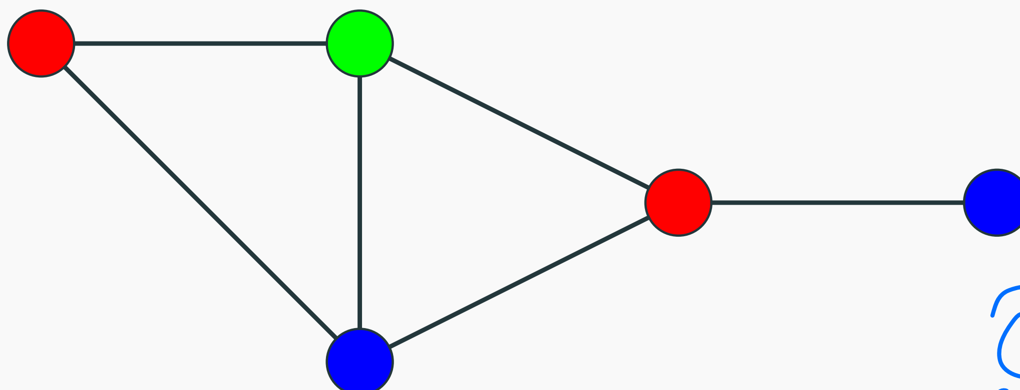
Instance: $G = (V, E)$: Undirected graph.
Question: Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?

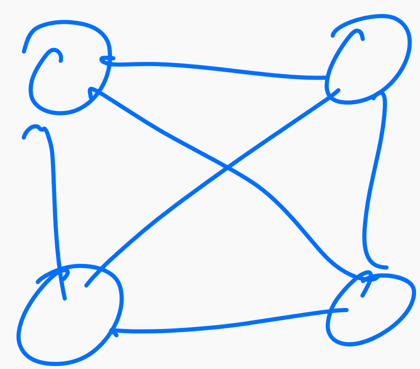


Graph 3-Coloring

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?





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-COLOR $\leq_P k -$ 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 $[a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]$
- 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 $[a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]$
- Each cell phone tower (simplifying) gets one band
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference

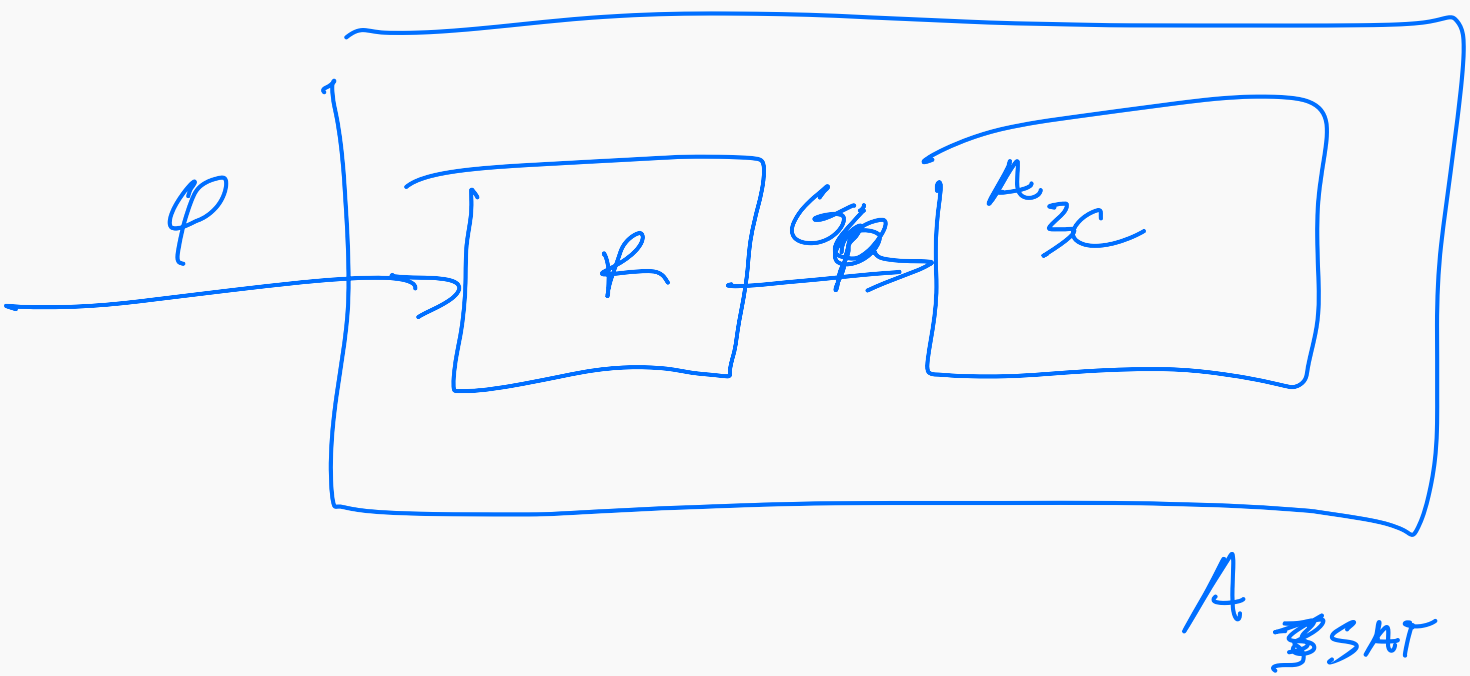
Problem: given k bands and some region with n towers, is there a way to assign the bands to avoid interference?

Can reduce to k-coloring by creating intereference/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-SAT $\leq_P$ 3-Coloring.



Reduction Idea

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

- need to establish truth assignment for $x_1, \ldots, x_n$ via colors for some nodes in G_φ.
- 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 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

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:

$$f(x_1, x_2) = (x_1 \vee x_2) \tag{12}$$

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

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

$$f(x_1, x_2) = (x_1 \vee x_2) \tag{12}$$

Assume green=true and red=false,

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

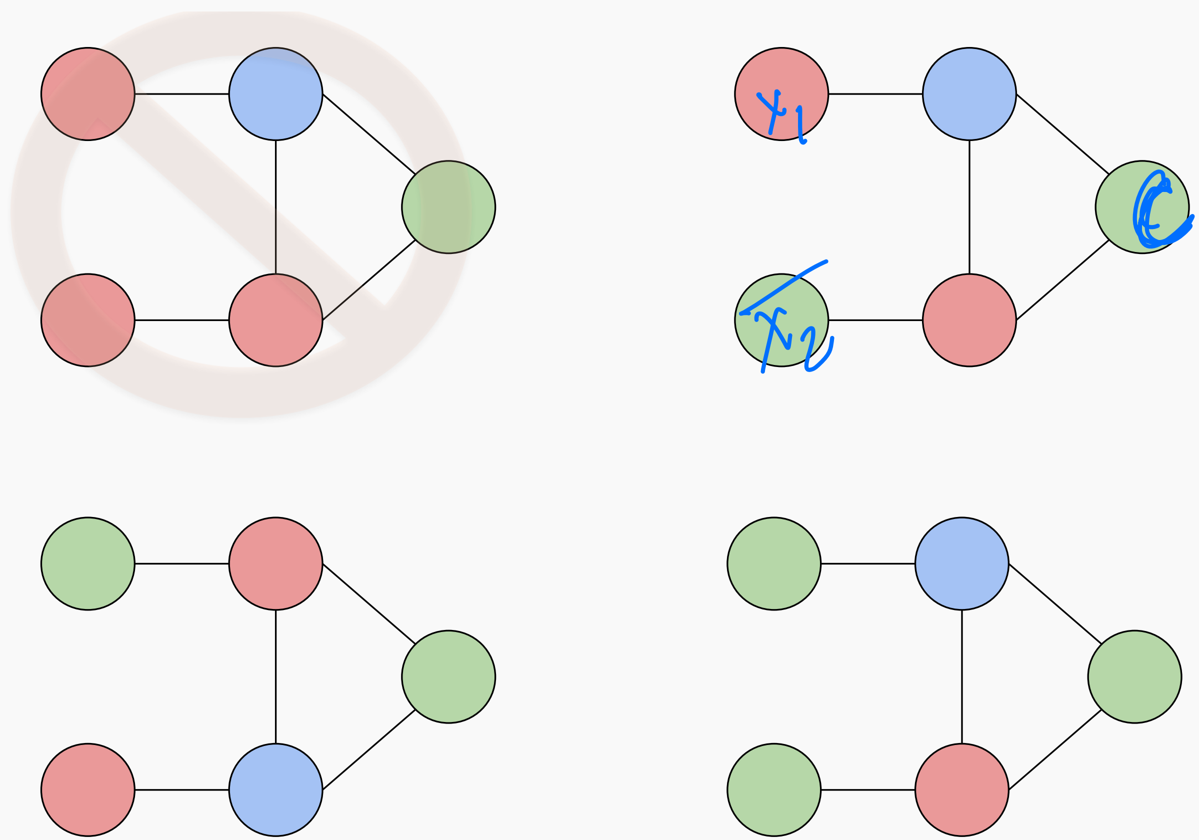
Let's try some stuff:

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 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

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?:

$$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \tag{13}$$

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

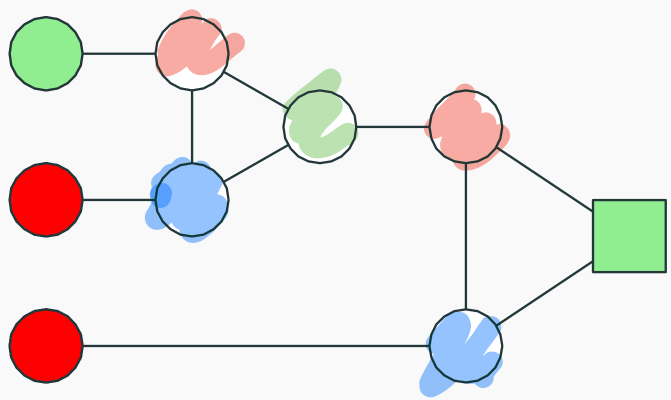
How do we do the same thing for 3 variables?:

$$f(x_1, x_2, x_3) = (x_1 \lor x_2 \lor x_3) \tag{13}$$

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).

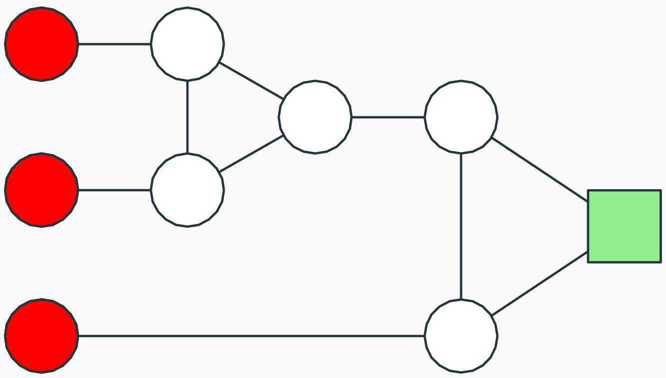


a Yes.

b No.

3 color this gadget.

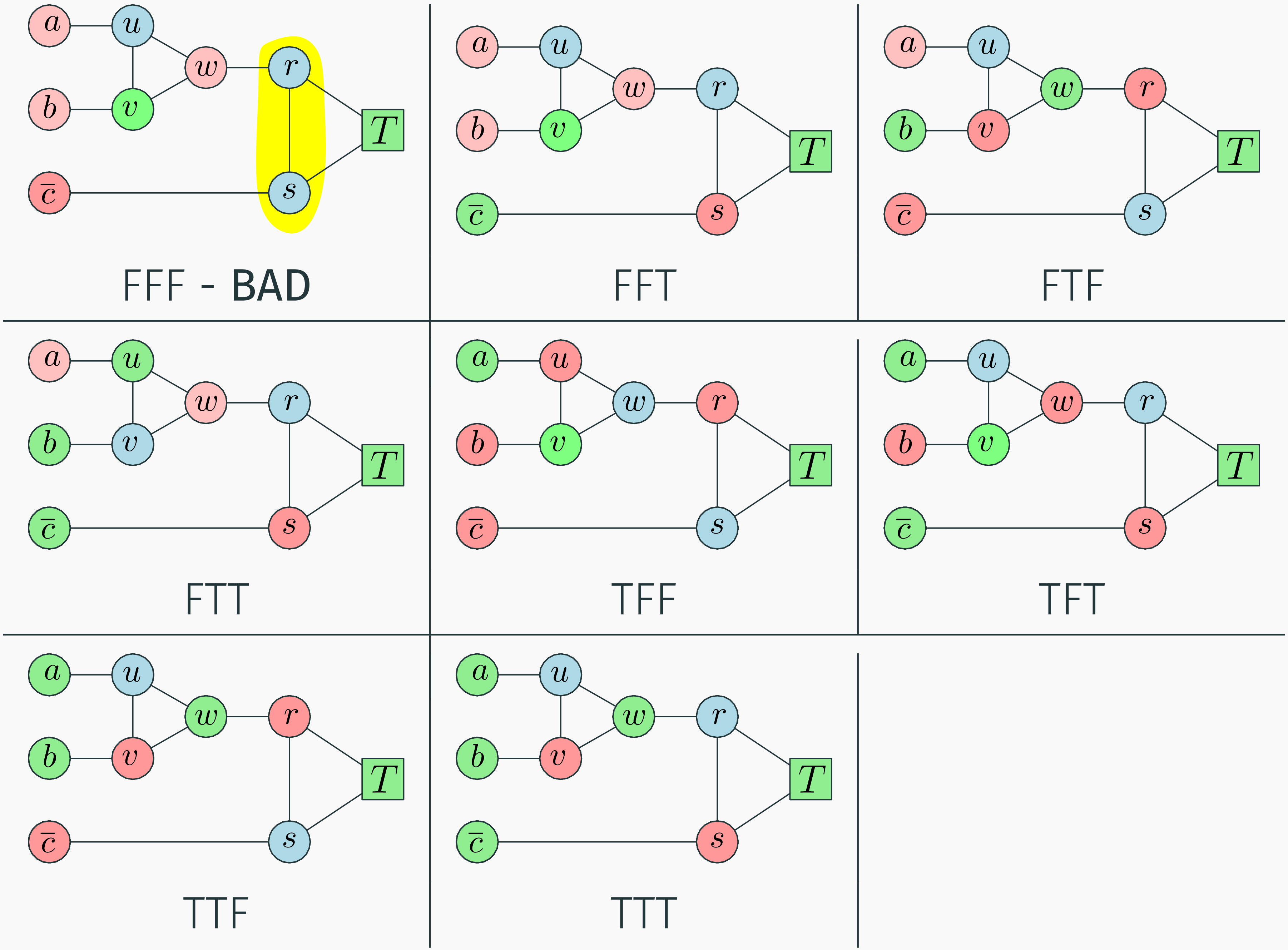
You are given three colors: red, green and blue. Can the following graph be three colored in a valid way (assuming that some of the nodes are already colored as indicated).



a Yes.

b No.

3-coloring of the clause gadget

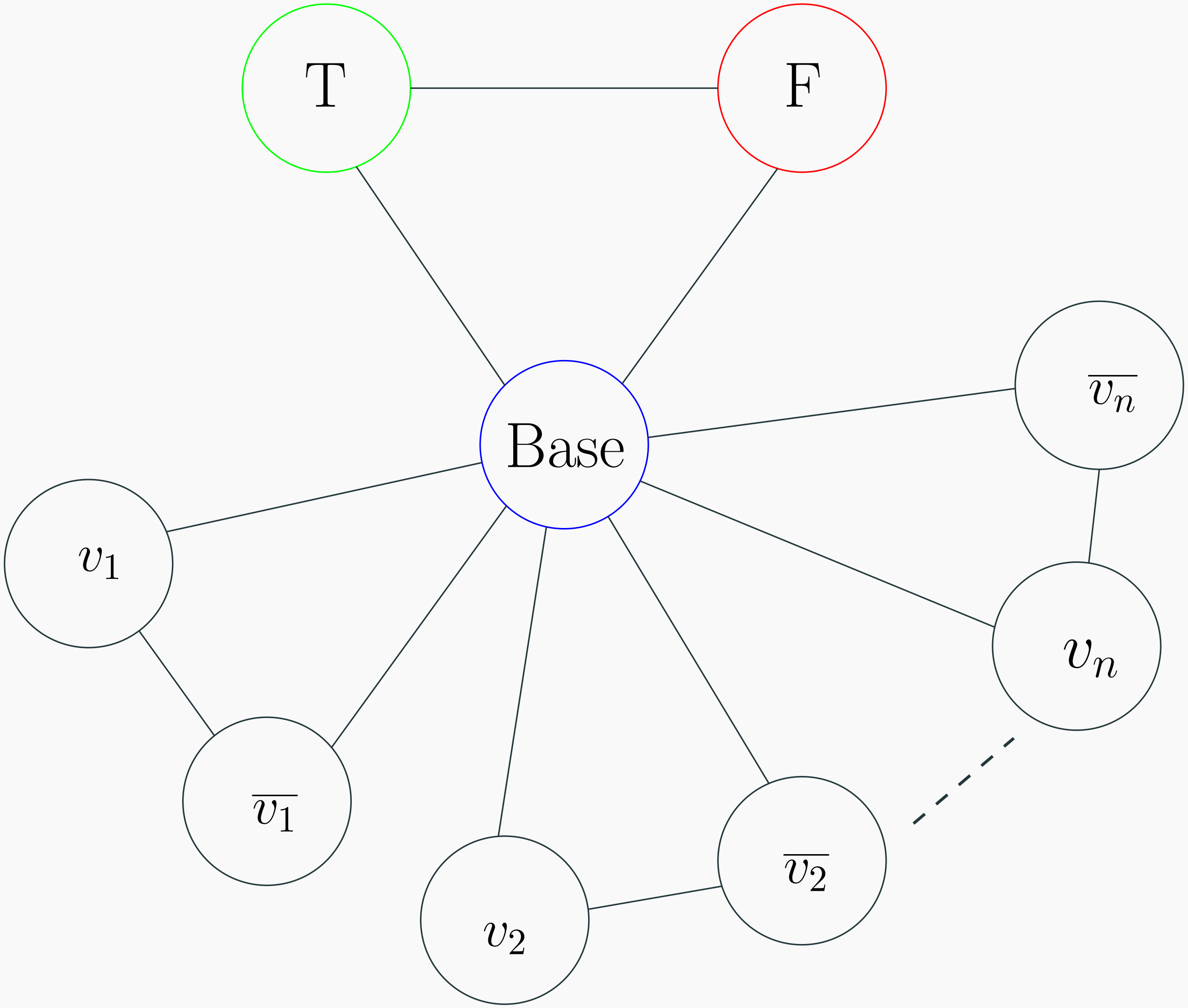


Reduction Idea II - Literal Assignment I

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

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

Reduction Idea II - Literal Assignment II

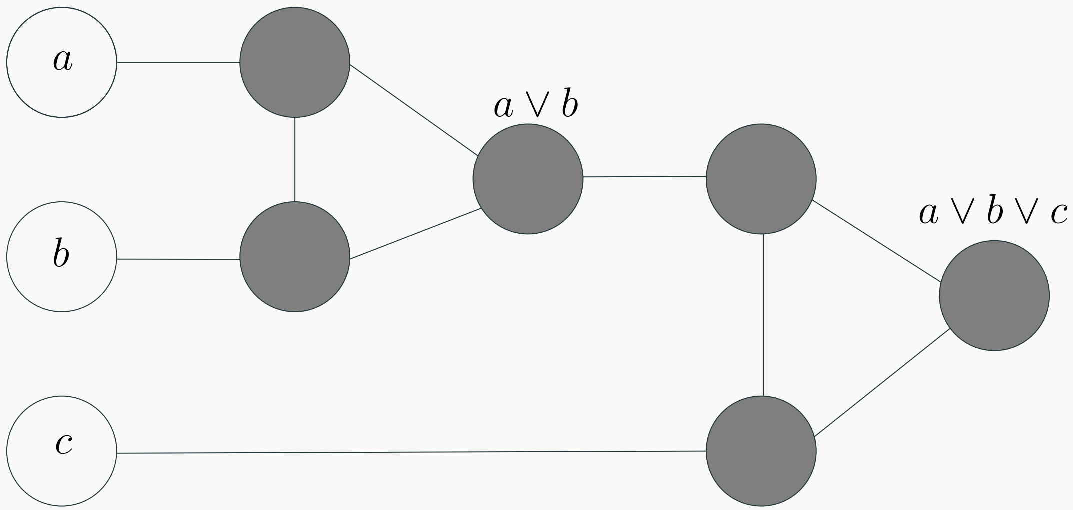


Review Clause Satisfiability Gadget

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:



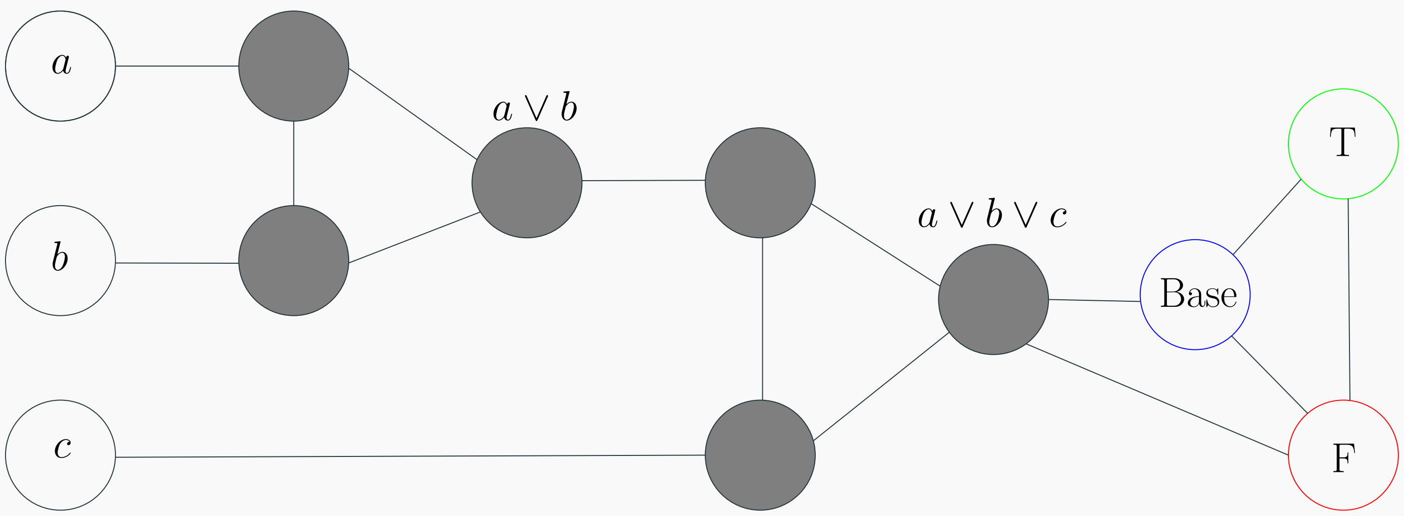
OR-Gadget Graph

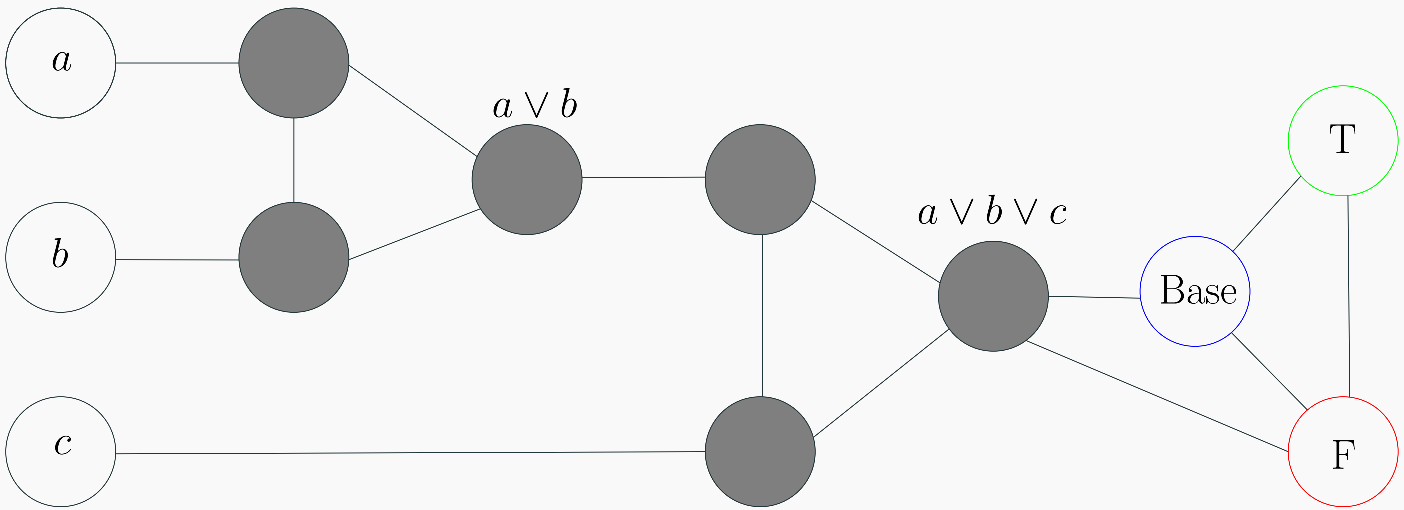
Property: if a, b, c are colored False in a 3-coloring then output node of OR-gadget has to be colored False.

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



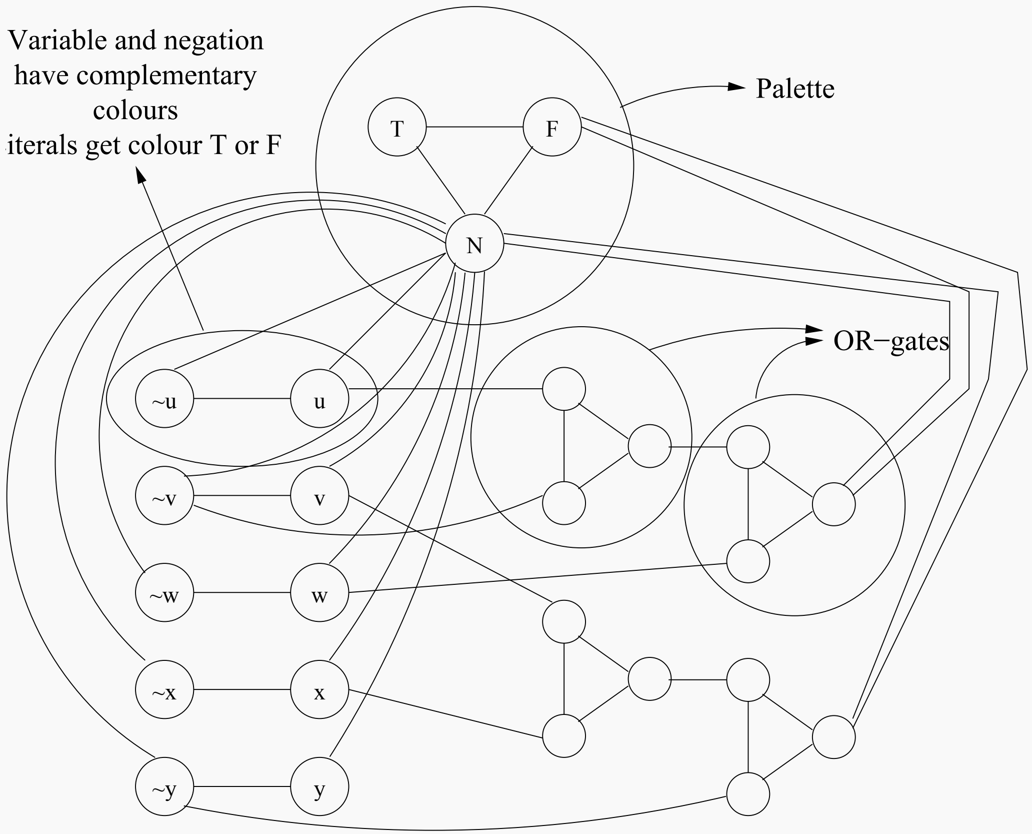


Lemma

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.

Example

$\varphi = (u \lor \neg v \lor w) \land (v \lor x \lor \neg y)$



Correctness of Reduction

φ is satisfiable implies G_φ is 3-colorable

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

Correctness of Reduction

φ is satisfiable implies G_φ 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

φ is satisfiable implies G_φ 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

φ is satisfiable implies G_φ 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_φ is 3-colorable implies φ is satisfiable

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

Correctness of Reduction

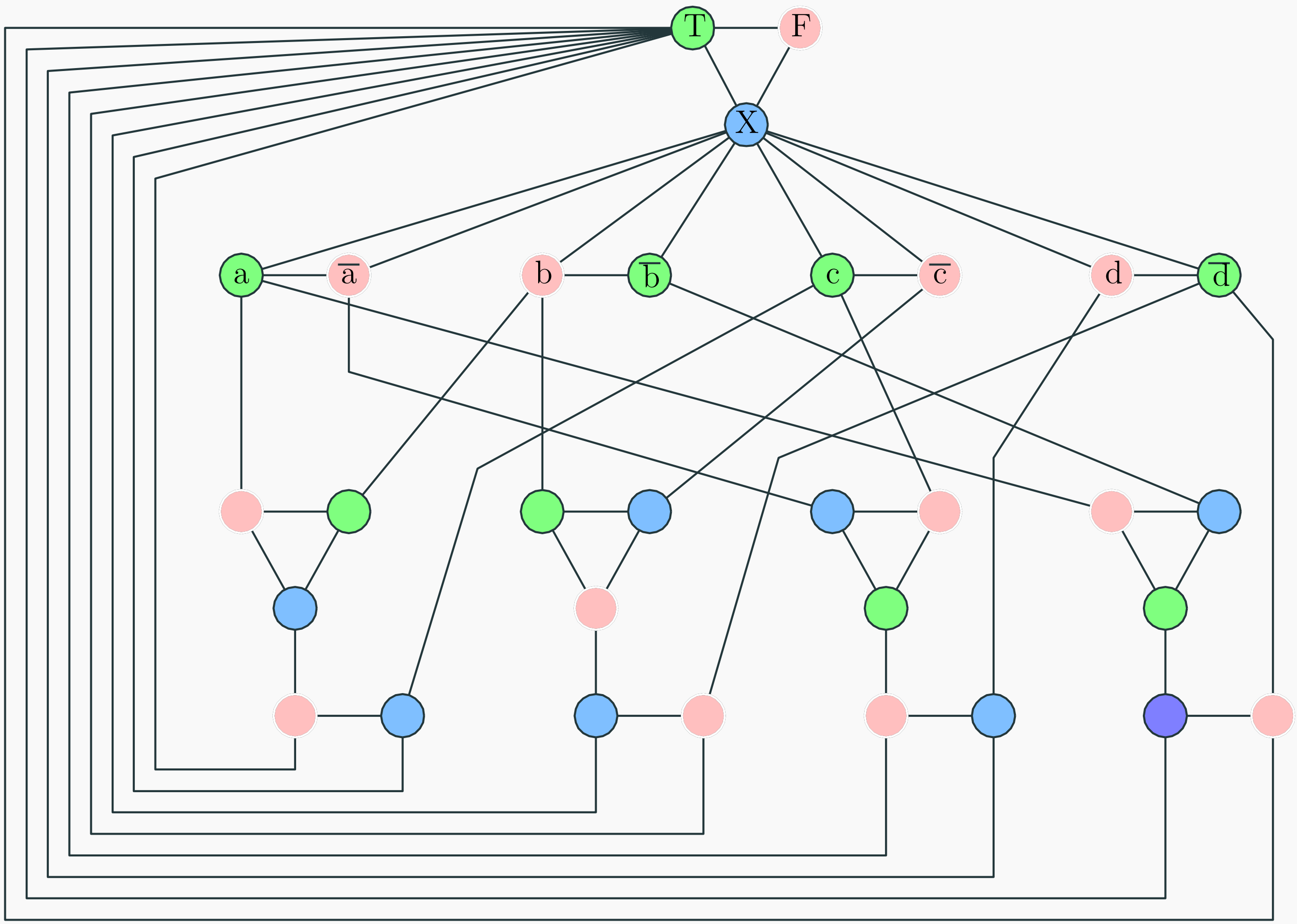
φ is satisfiable implies G_φ is 3-colorable

- 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_φ is 3-colorable implies φ 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

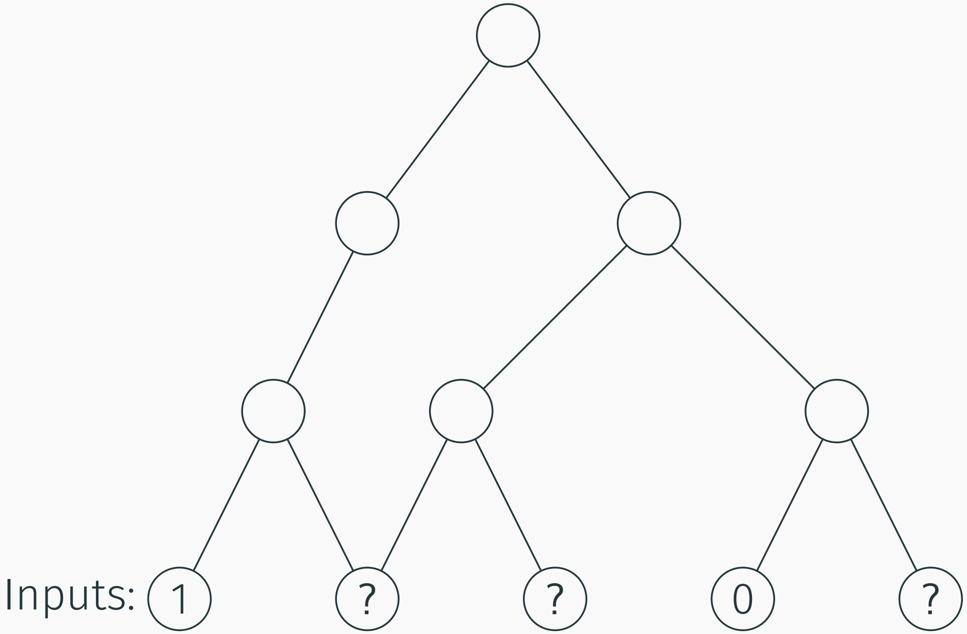


$P_{NP} \leq P_{NP\text{-}Hard}$

Circuit-Sat Problem

Circuits

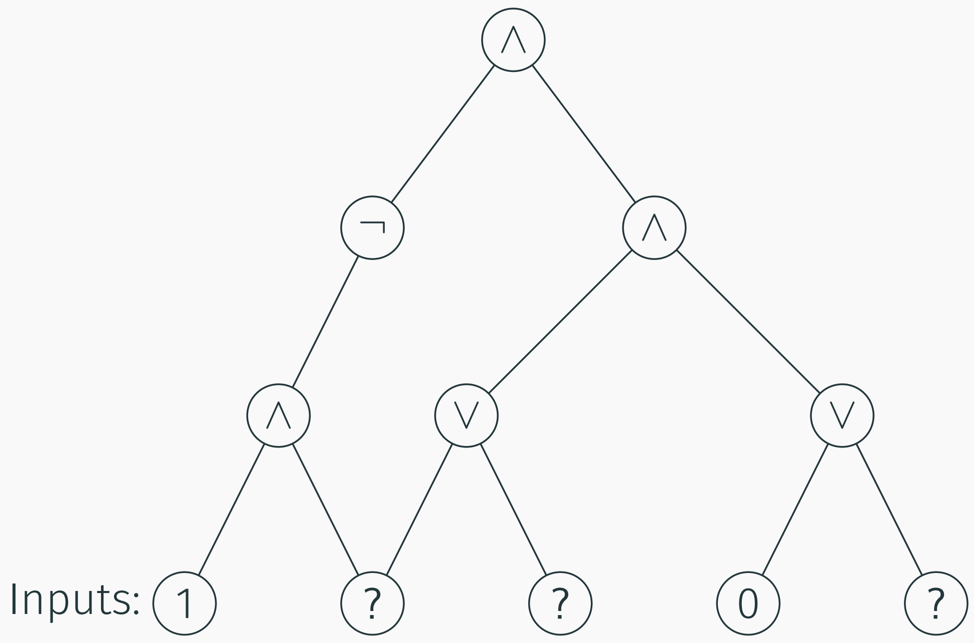
A circuit is a directed acyclic graph with



- 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

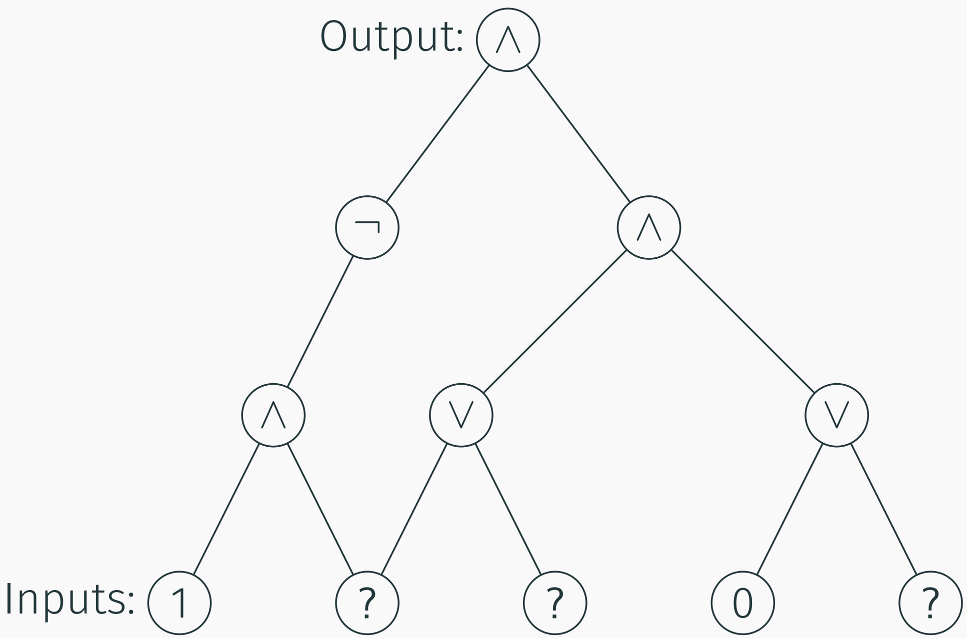
A circuit is a directed acyclic graph with



- Input vertices (without incoming edges) labeled with 0, 1 or a distinct variable.
- Every other vertex is labeled ∨, ∧ or ¬.
- 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 ∨, ∧ or ¬.
- 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 φ with n variables and m clauses, create a Circuit C.

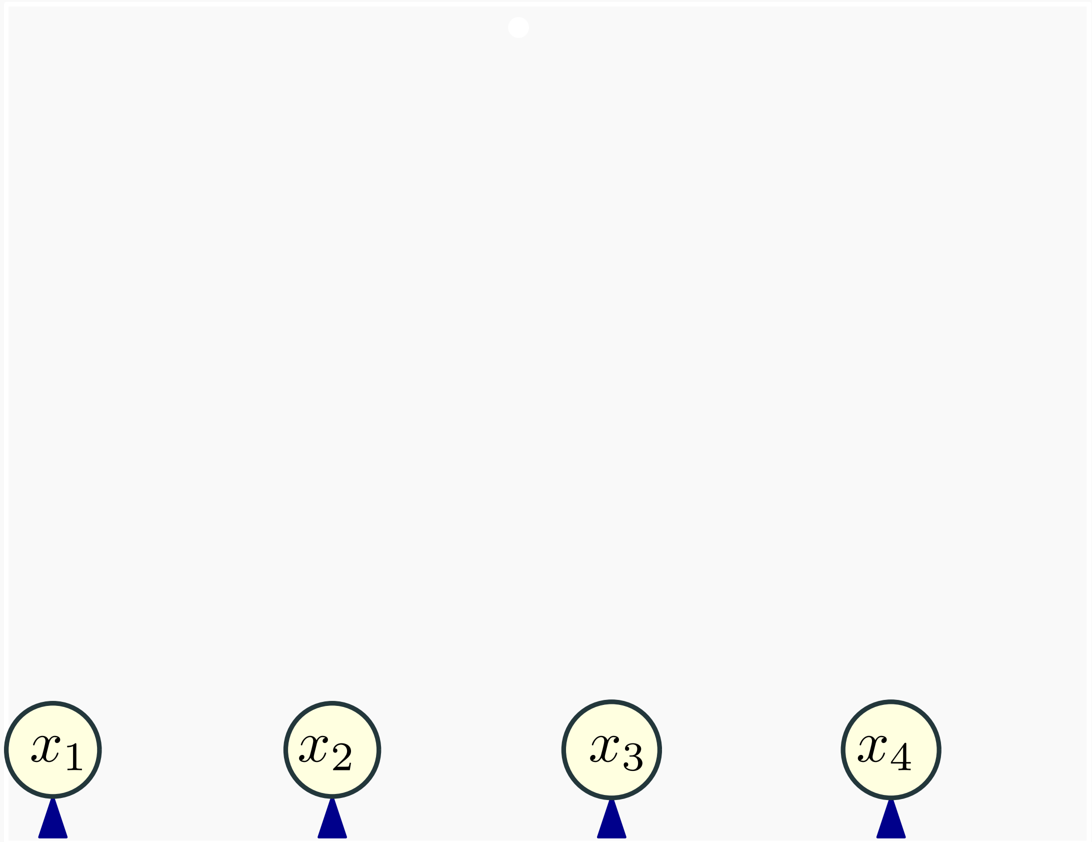
- Inputs to C are the n boolean variables $x_1, x_2, \ldots, x_n$
- Use NOT gate to generate literal $\neg x_i$ for each variable x_i
- For each clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ use two OR gates to mimic formula
- Combine the outputs for the clauses using AND gates to obtain the final output

Example: 3SAT $\leq_P$ CSAT

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

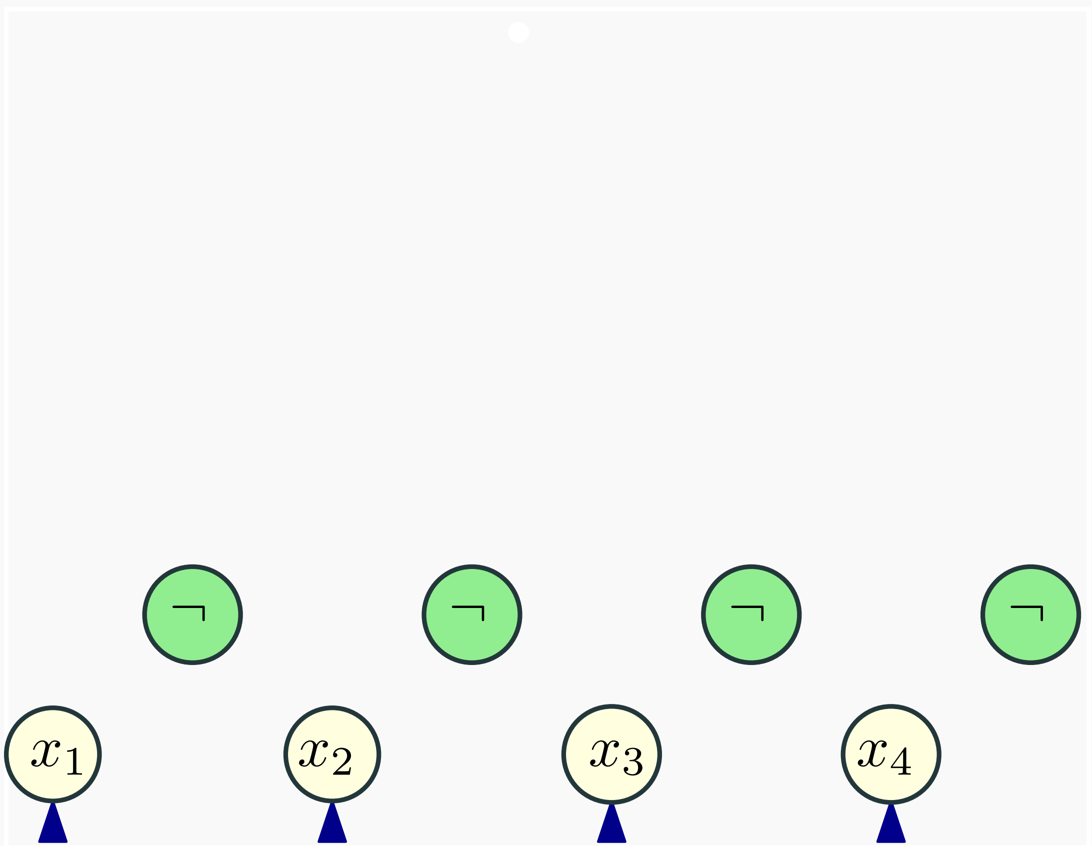
Example: 3SAT $\leq_P$ CSAT

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



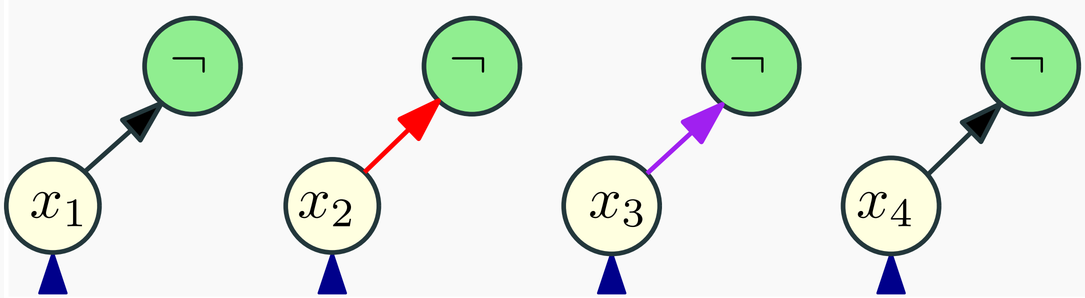
Example: 3SAT $\leq_P$ CSAT

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



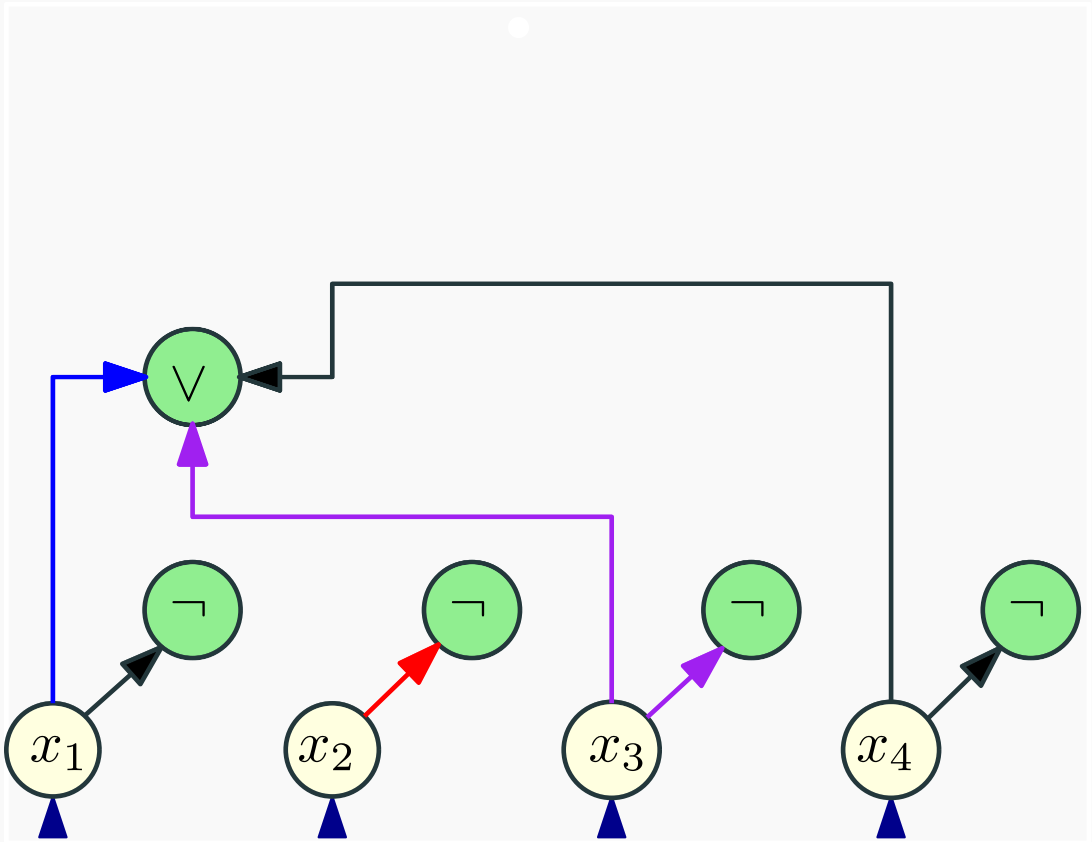
Example: 3SAT $\leq_P$ CSAT

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



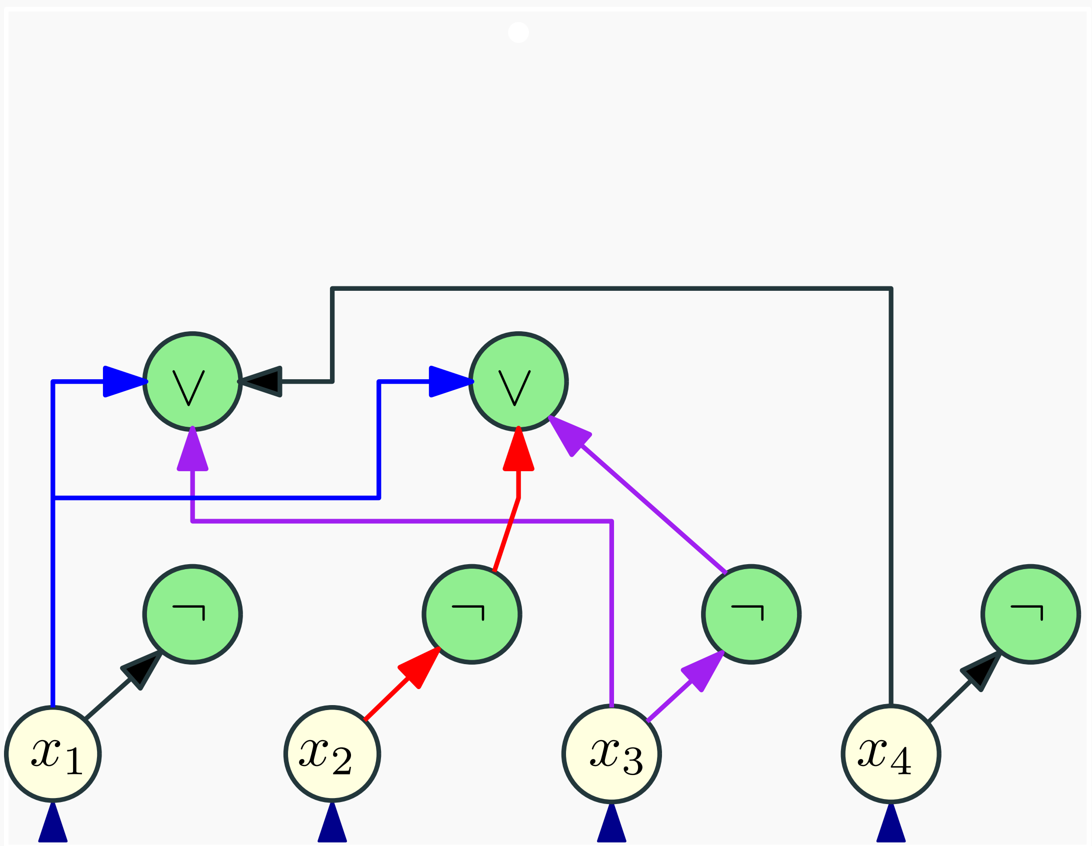
Example: 3SAT $\leq_P$ CSAT

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



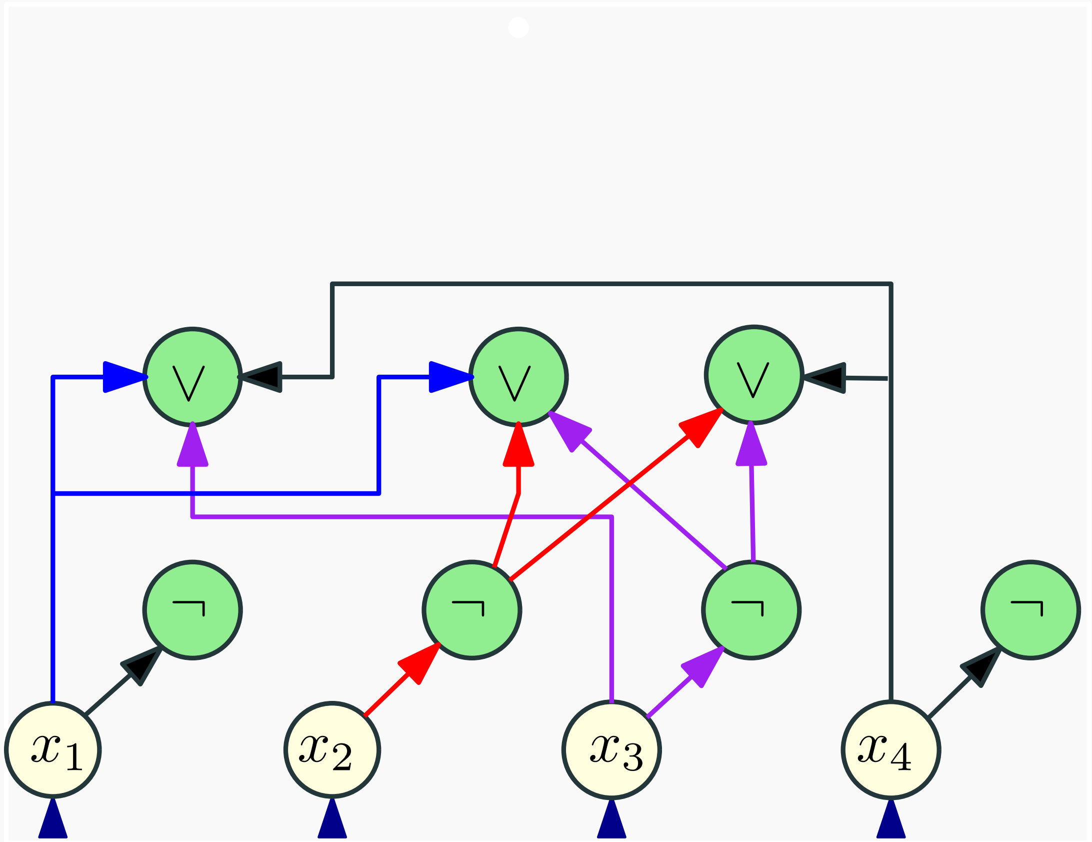
Example: 3SAT $\leq_P$ CSAT

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



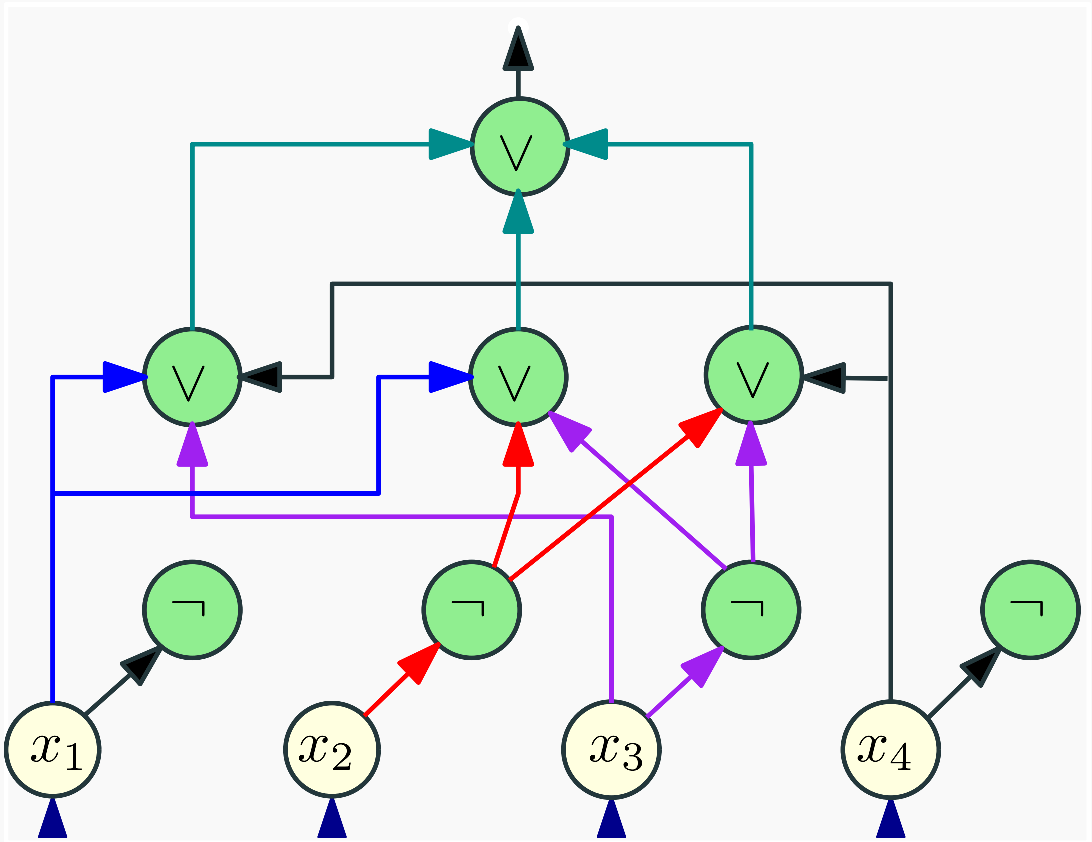
Example: 3SAT $\leq_P$ CSAT

$$\varphi = \Big(x_1 \lor \lor x_3 \lor x_4\Big) \land \Big(x_1 \lor \neg x_2 \lor \neg x_3\Big) \land \Big(\neg x_2 \lor \neg x_3 \lor x_4\Big)$$



Example: 3SAT $\leq_P$ CSAT

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



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

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

$$\equiv$$

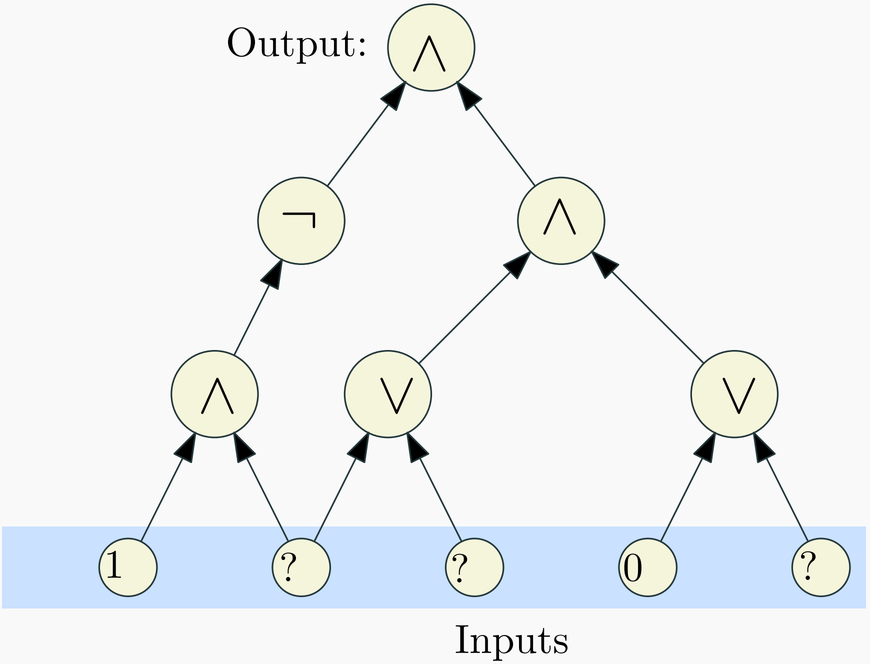
$$(z \vee \bar{x} \vee \bar{y}) \wedge (\bar{z} \vee x \vee y) \wedge (\bar{z} \vee x \vee \bar{y}) \wedge (\bar{z} \vee \bar{x} \vee y)$$

Summary of formulas we derived

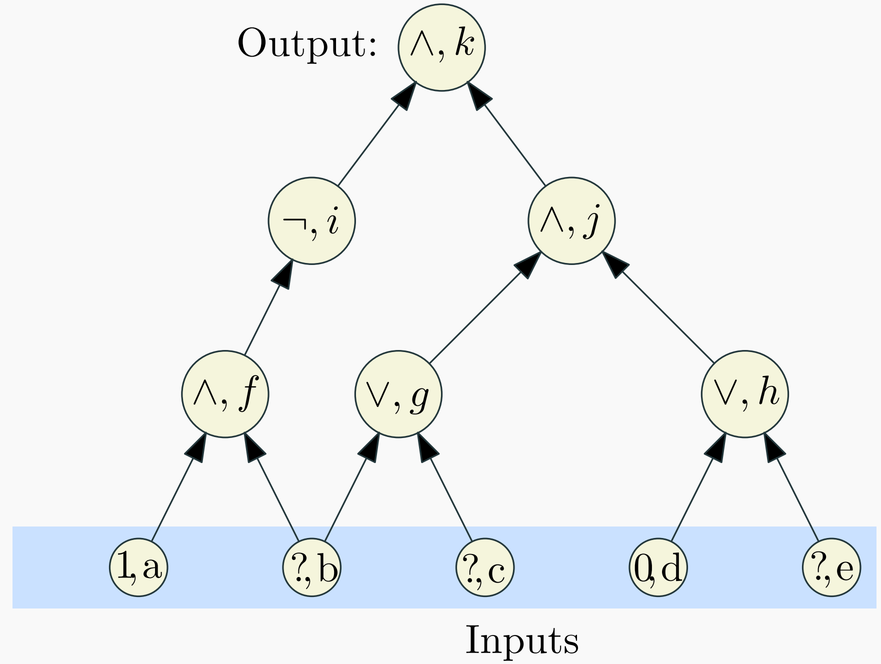
Lemma
The following identities hold:

- $z = \overline{x} \quad \equiv \quad (z \vee x) \wedge (\overline{z} \vee \overline{x})\,.$
- $\left(z = x \vee y\right) \quad \equiv \quad (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$
- $\left(z = x \wedge y\right) \quad \equiv \quad \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$

Converting a circuit into a CNF formula

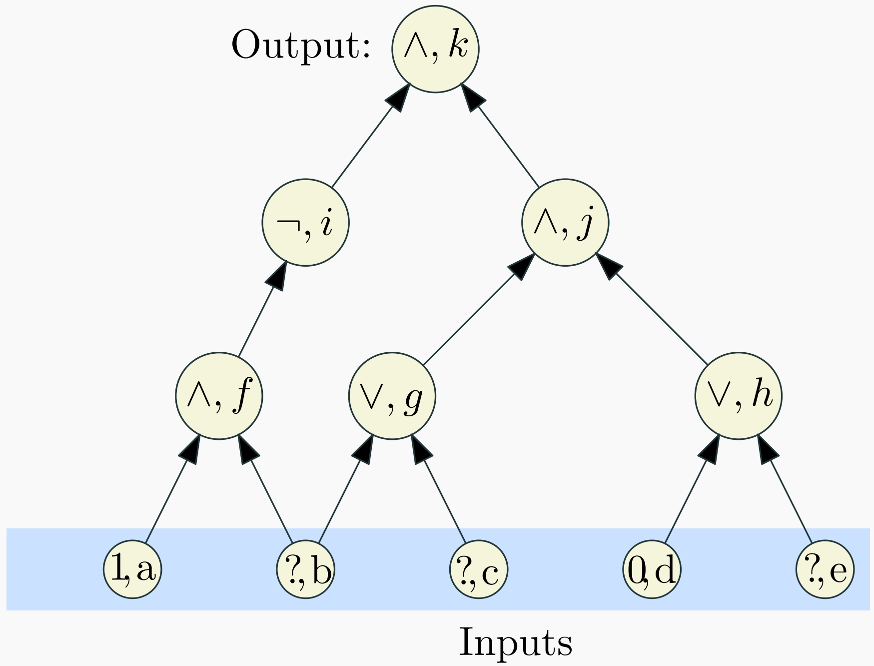


(A) Input circuit

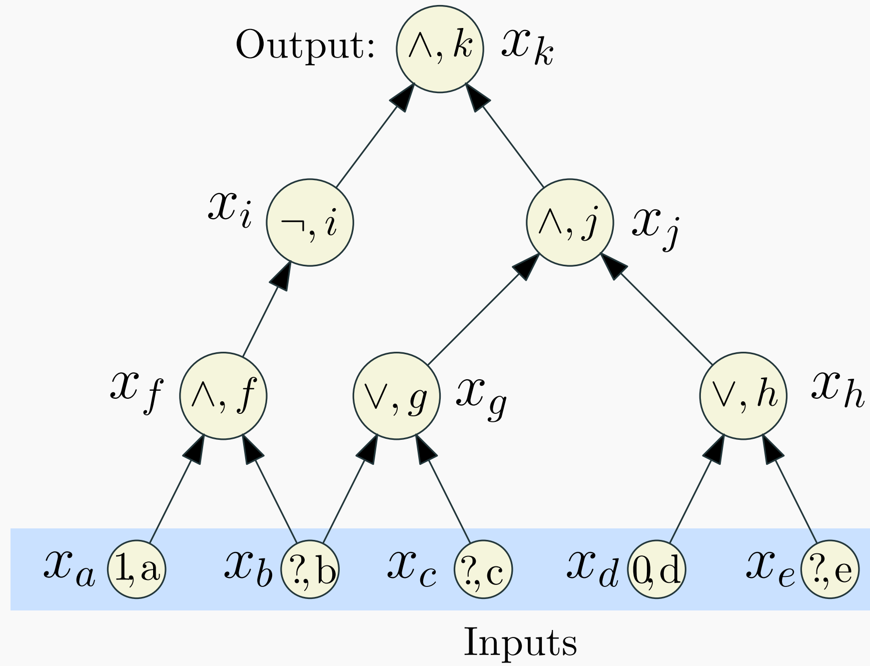


(B) Label the nodes.

Converting a circuit into a CNF formula

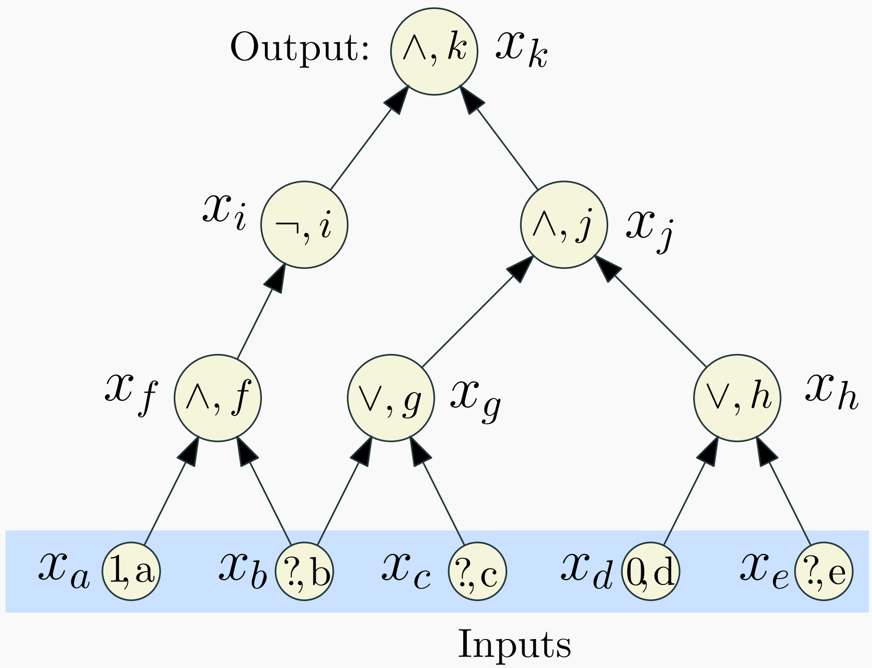


(B) Label the nodes.



(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!)

$x_k = x_i \wedge x_j$

$x_j = x_g \wedge x_h$

$x_i = \neg x_f$

$x_h = x_d \vee x_e$

$x_g = x_b \vee x_c$

$x_f = x_a \wedge x_b$

$x_d = 0$

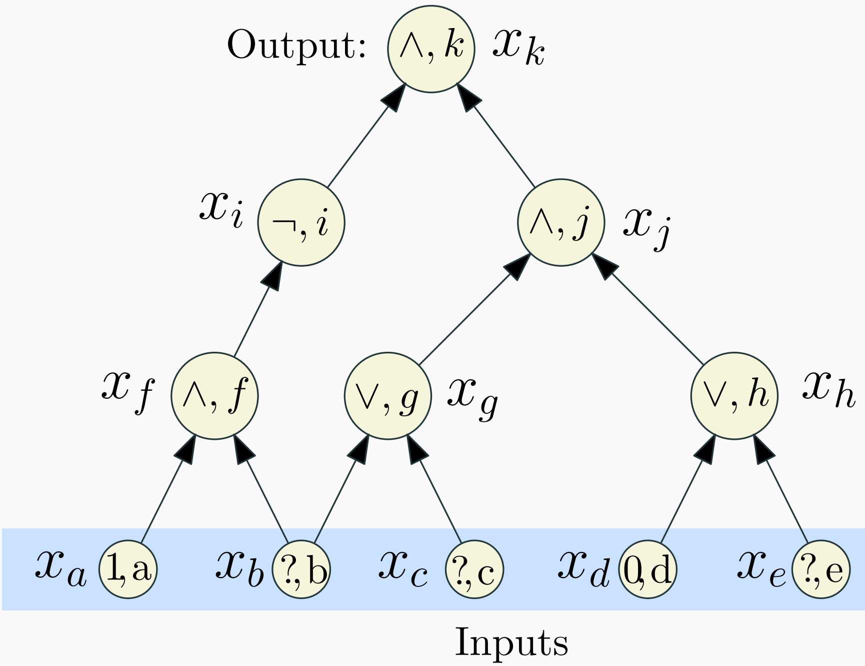
$x_a = 1$

(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



$$x_k \wedge (\neg x_k \vee x_i) \wedge (\neg x_k \vee x_j)$$
$$\wedge (x_k \vee \neg x_i \vee \neg x_j) \wedge (\neg x_j \vee x_g)$$
$$\wedge (\neg x_j \vee x_h) \wedge (x_j \vee \neg x_g \vee \neg x_h)$$
$$\wedge (x_i \vee x_f) \wedge (\neg x_i \vee \neg x_f)$$
$$\wedge (x_h \vee \neg x_d) \wedge (x_h \vee \neg x_e)$$
$$\wedge (\neg x_h \vee x_d \vee x_e) \wedge (x_g \vee \neg x_b)$$
$$\wedge (x_g \vee \neg x_c) \wedge (\neg x_g \vee x_b \vee x_c)$$
$$\wedge (\neg x_f \vee x_a) \wedge (\neg x_f \vee x_b)$$
$$\wedge (x_f \vee \neg x_a \vee \neg x_b) \wedge (\neg x_d) \wedge x_a$$

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 $(x_u \lor x_v)$, $(\neg x_u \lor \neg x_v)$. Observe that

$$x_v = \neg x_u \text{ is true} \quad \iff \quad \begin{matrix} (x_u \lor x_v) \\ (\neg x_u \lor \neg x_v) \end{matrix} \quad \text{both true.}$$

Reduction: $CSAT \leq_P SAT$

- Case $\vee$: 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

$$\left(x_v = x_u \vee x_w\right) \text{ is true} \quad \iff \quad \begin{matrix} (x_v \vee \neg x_u), \\ (x_v \vee \neg x_w), \\ (\neg x_v \vee x_u \vee x_w) \end{matrix} \quad \text{all true.}$$

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 φ_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 φ_C (exercise)

$\Leftarrow$ Consider a satisfying assignment a for φ_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