NP-C problems & reductions redux

Sides based on material by Kani, Erickson, Chekuri, et. al.

All mistakes are my own! - Ivan Abraham (Fall 2024)

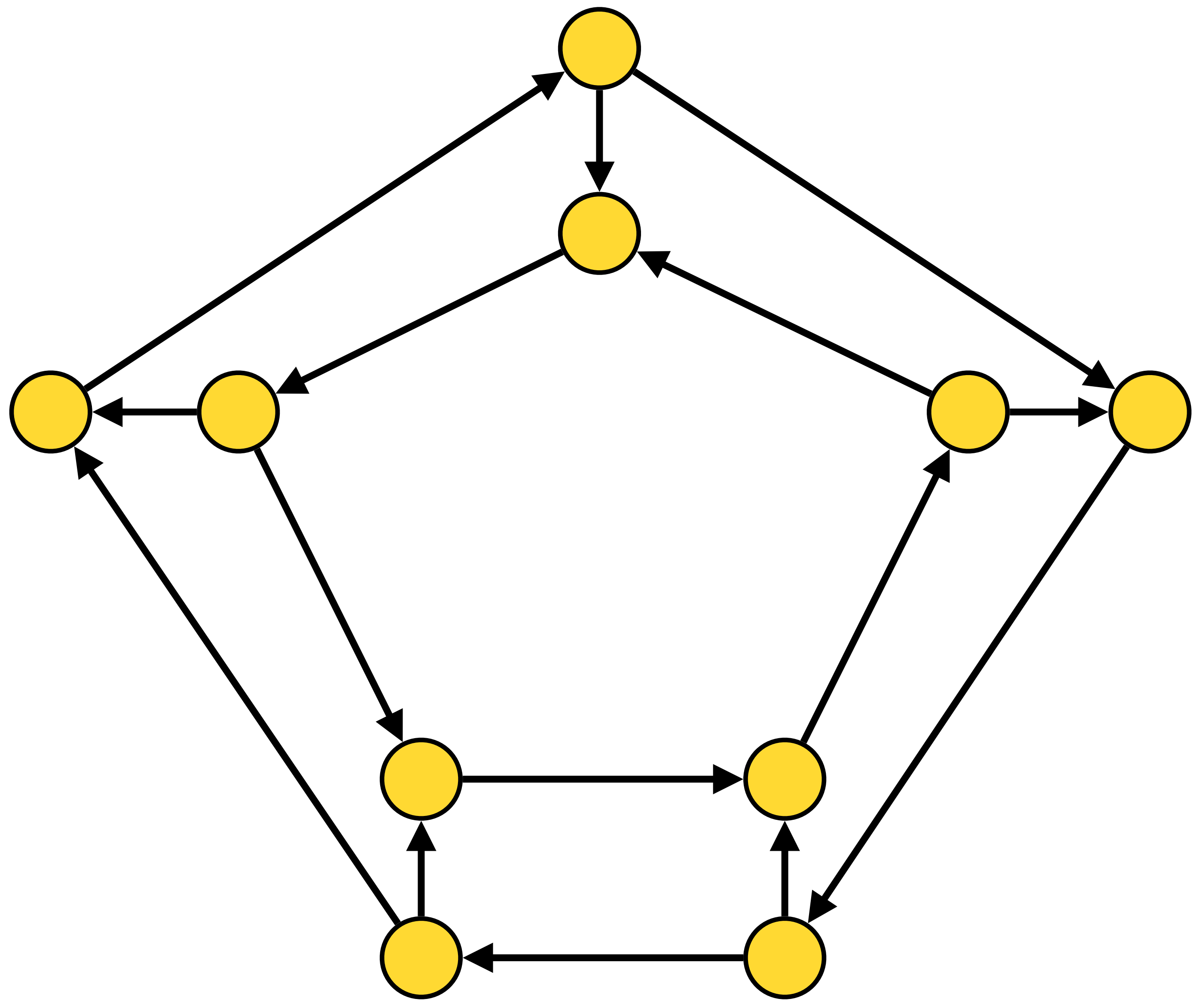
Image by ChatGPT (probably collaborated with DALL-E)

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?

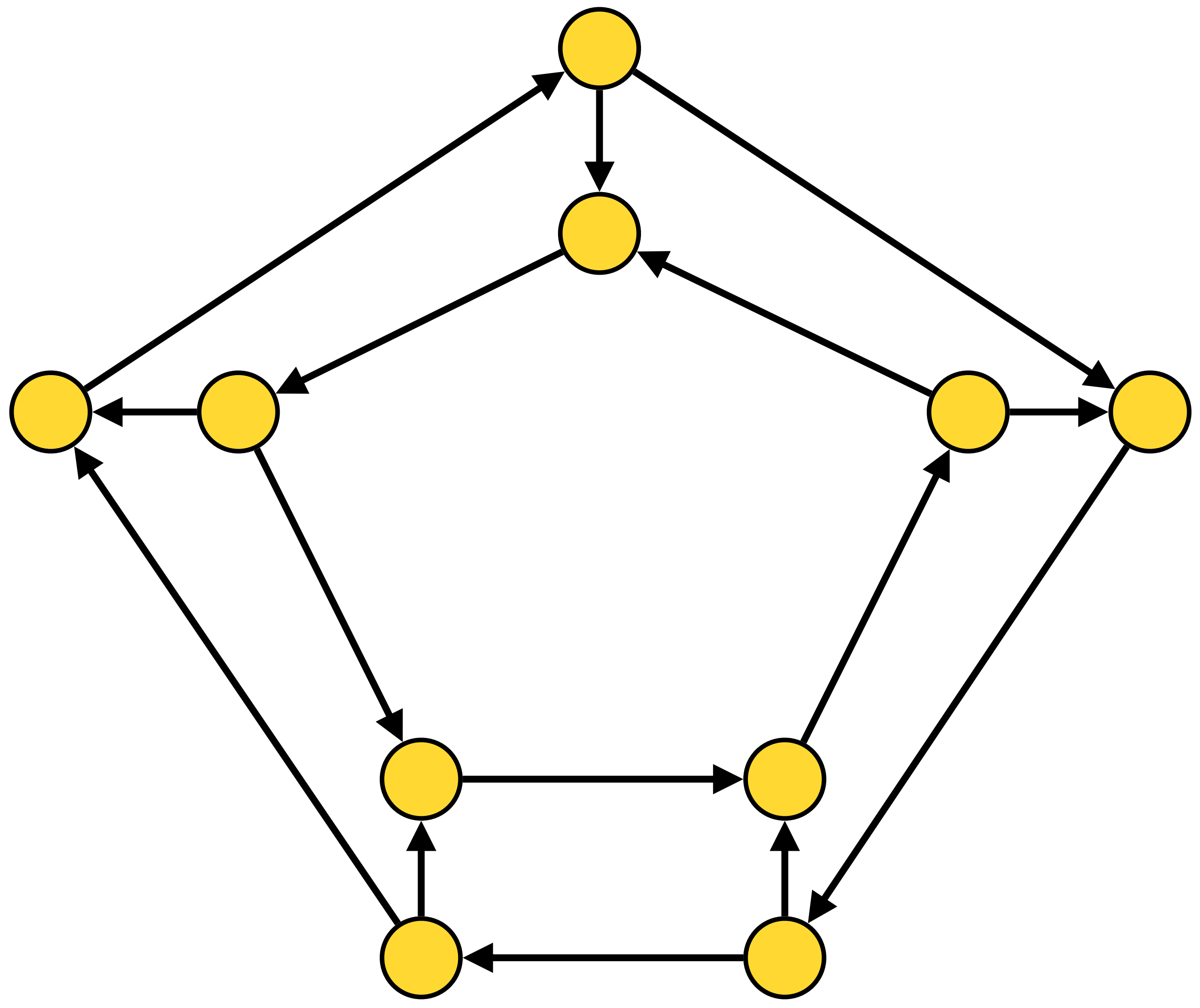


Directed Hamiltonian cycle

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

Goal: Does G have a Hamiltonian cycle?

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

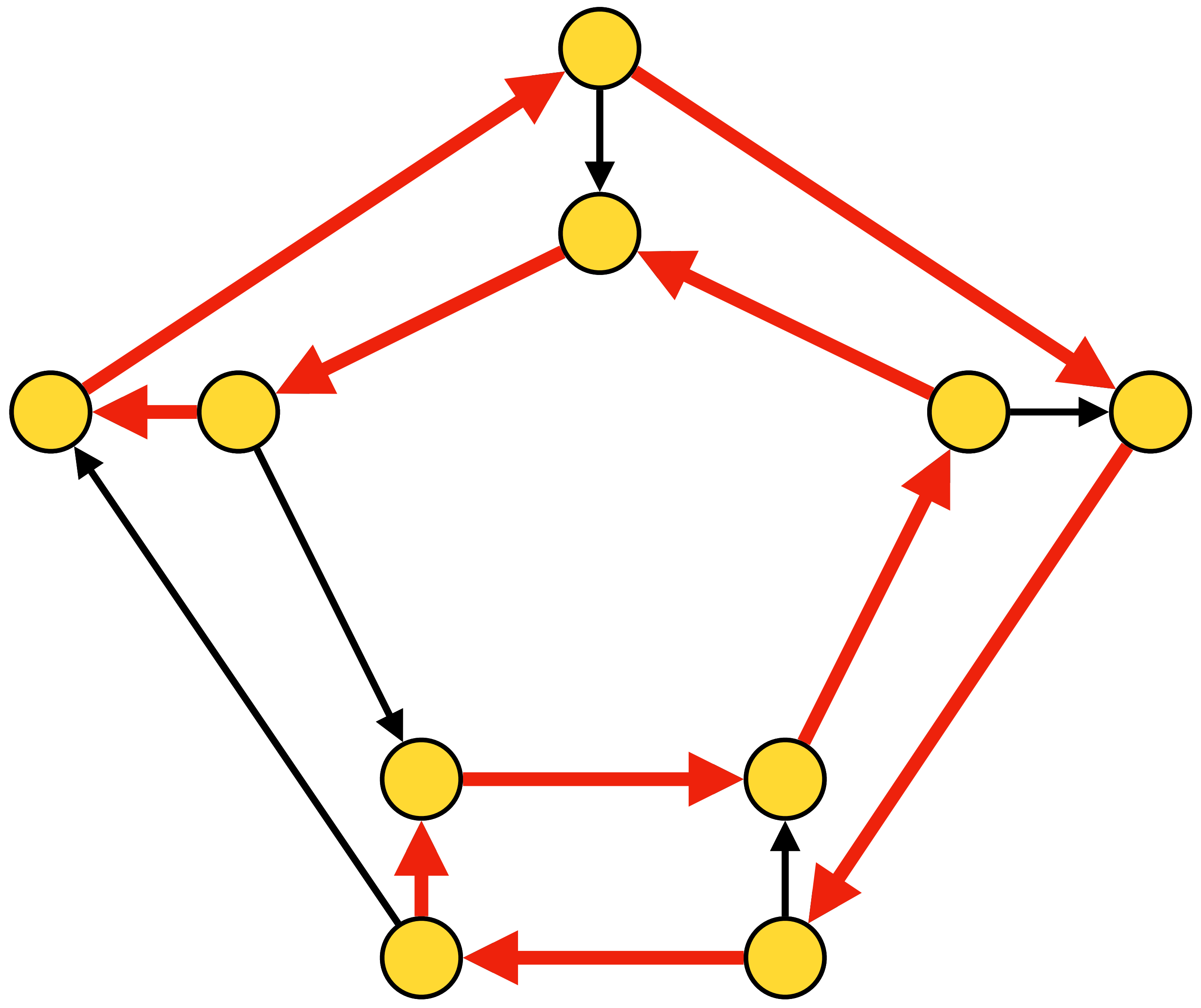


Directed Hamiltonian cycle

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

Goal: Does G have a Hamiltonian cycle?

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

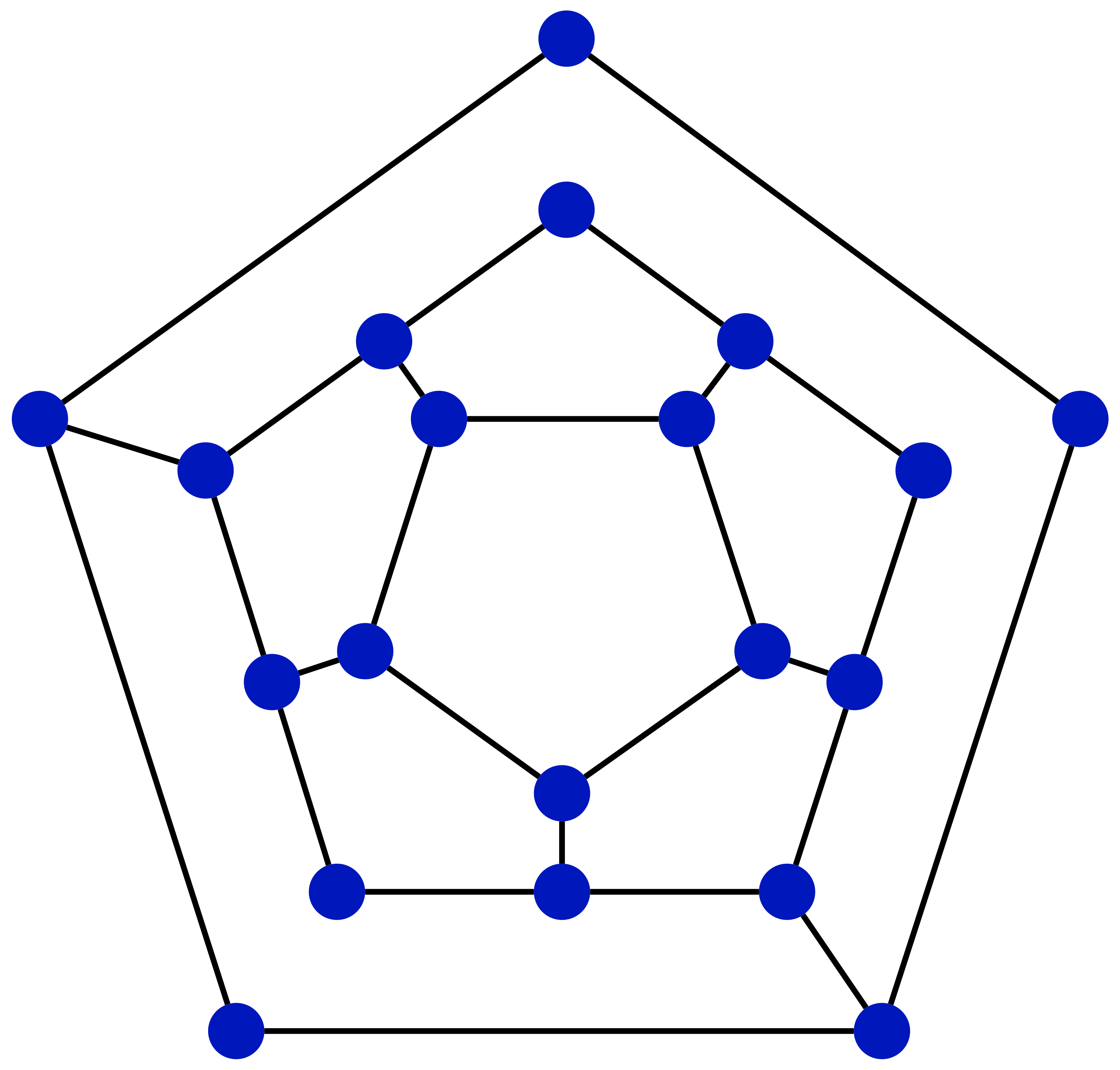


Question

Is the following graph Hamiltonian?

Same as asking if the graph has a cycle that visits each vertex exactly once.

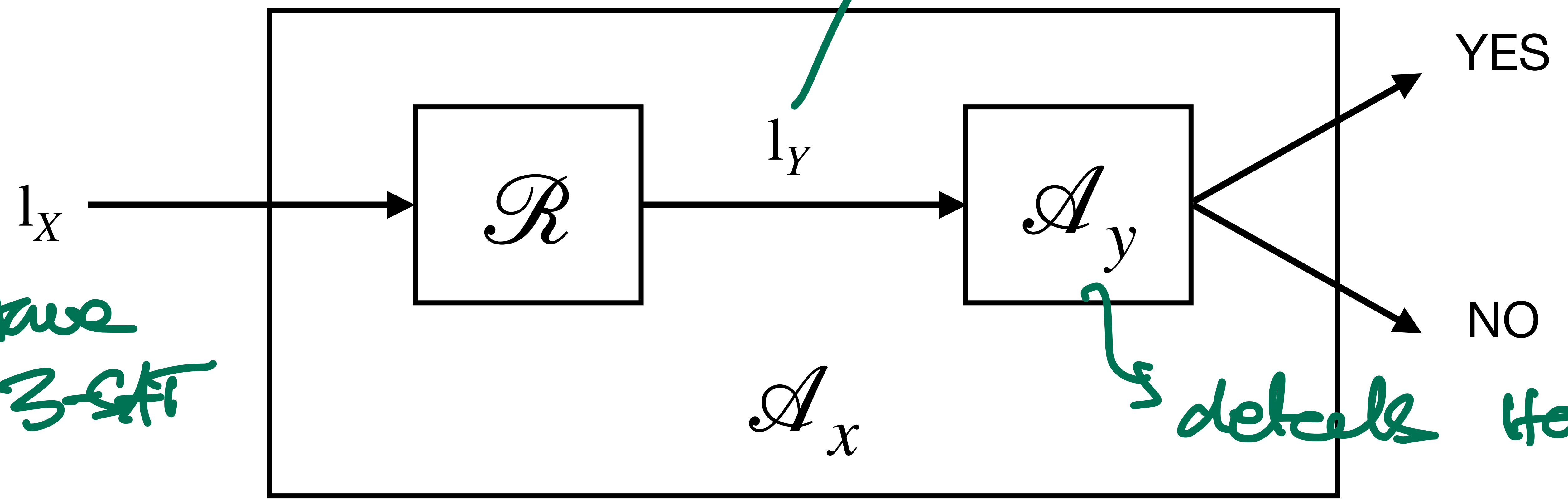
· Does not



Directed Hamiltonian cycle is NP-C

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

is a directed graph.



instance of 3-SAT

detects Hamiltonian cycle.

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

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

ϕ: phi φ: psi

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

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

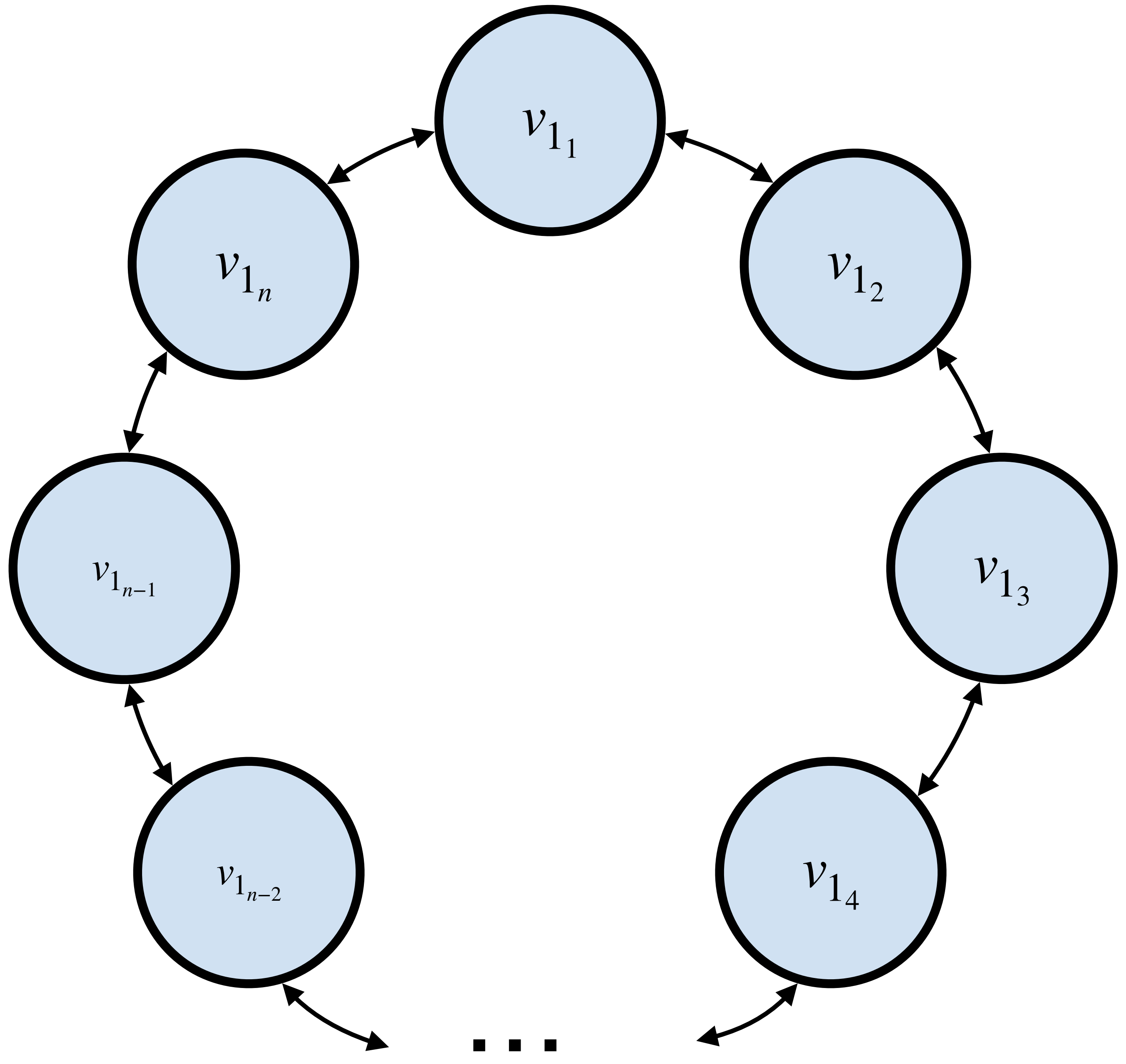
Reduction

Encoding idea I

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

$$f(X_1) = 1$$

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



Reduction

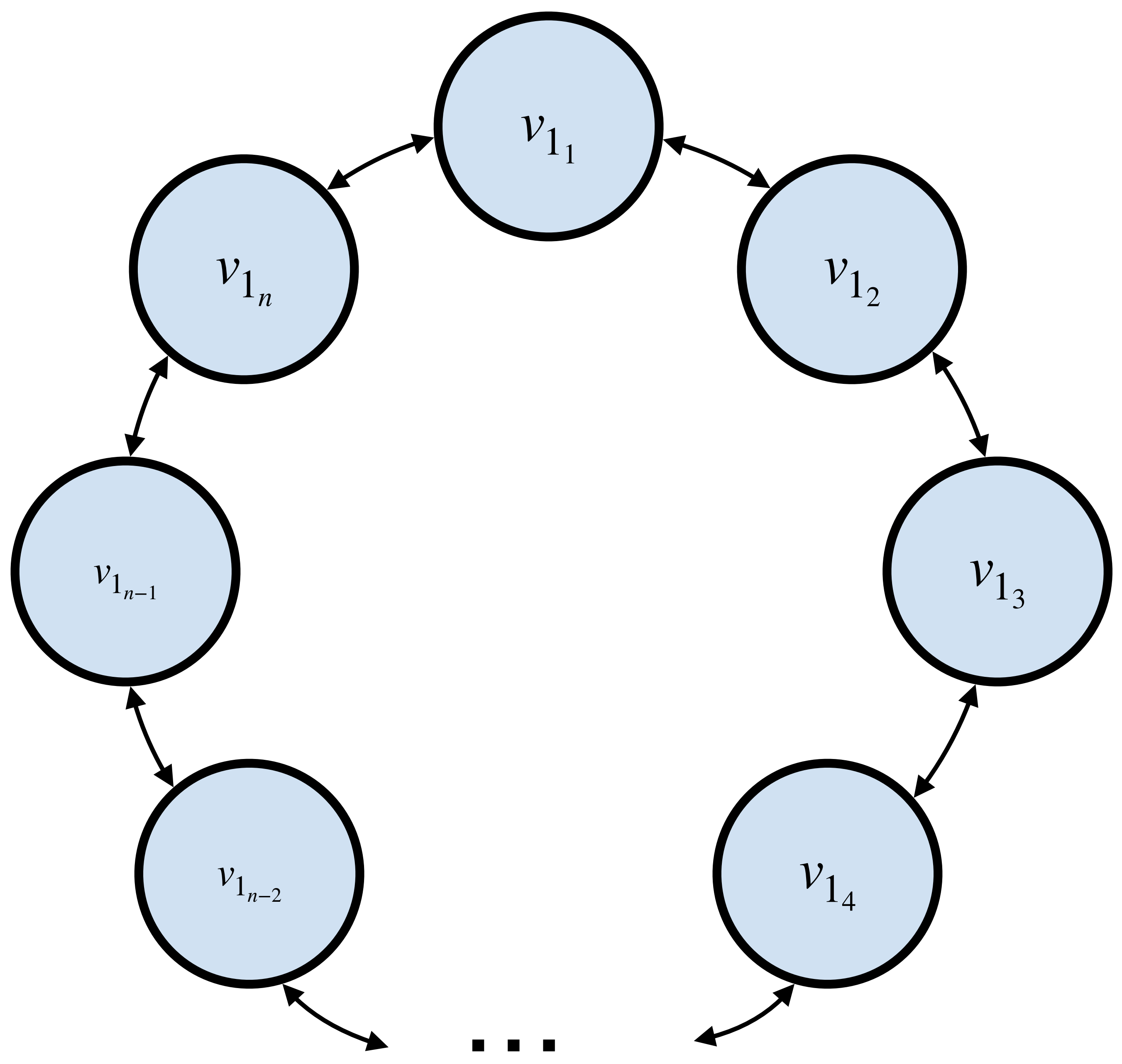
Encoding idea I

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

$$f(X_1) = 1$$

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

But how do we encode the variable?



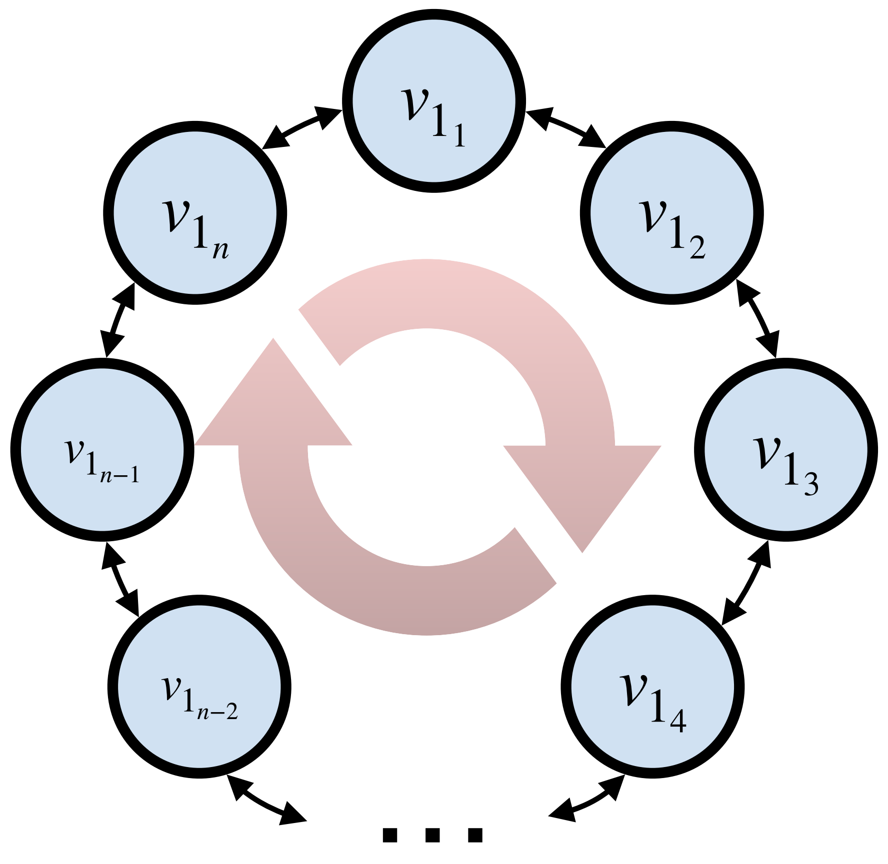
Reduction

Encoding idea I

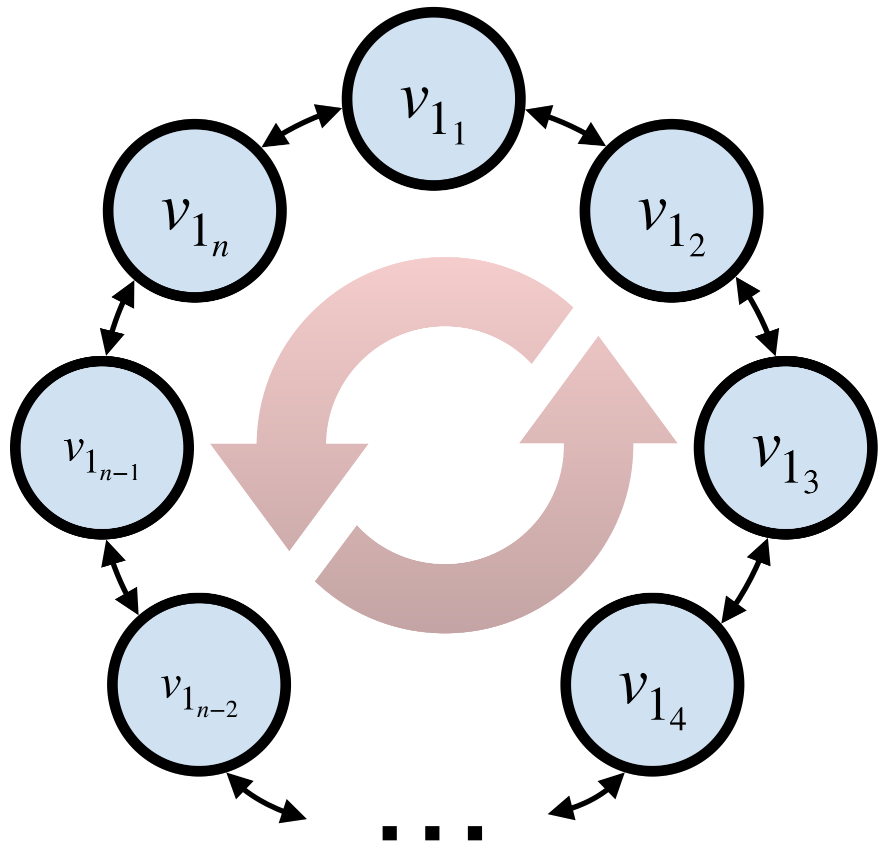
Need to create a graph from any arbitrary boolean assignment. Consider the expression:

$$f(X_1) = 1$$

Maybe we can encode the variable X_1 in terms of the cycle direction.



If $X_1 = 1$



If $X_1 = 0$

Reduction

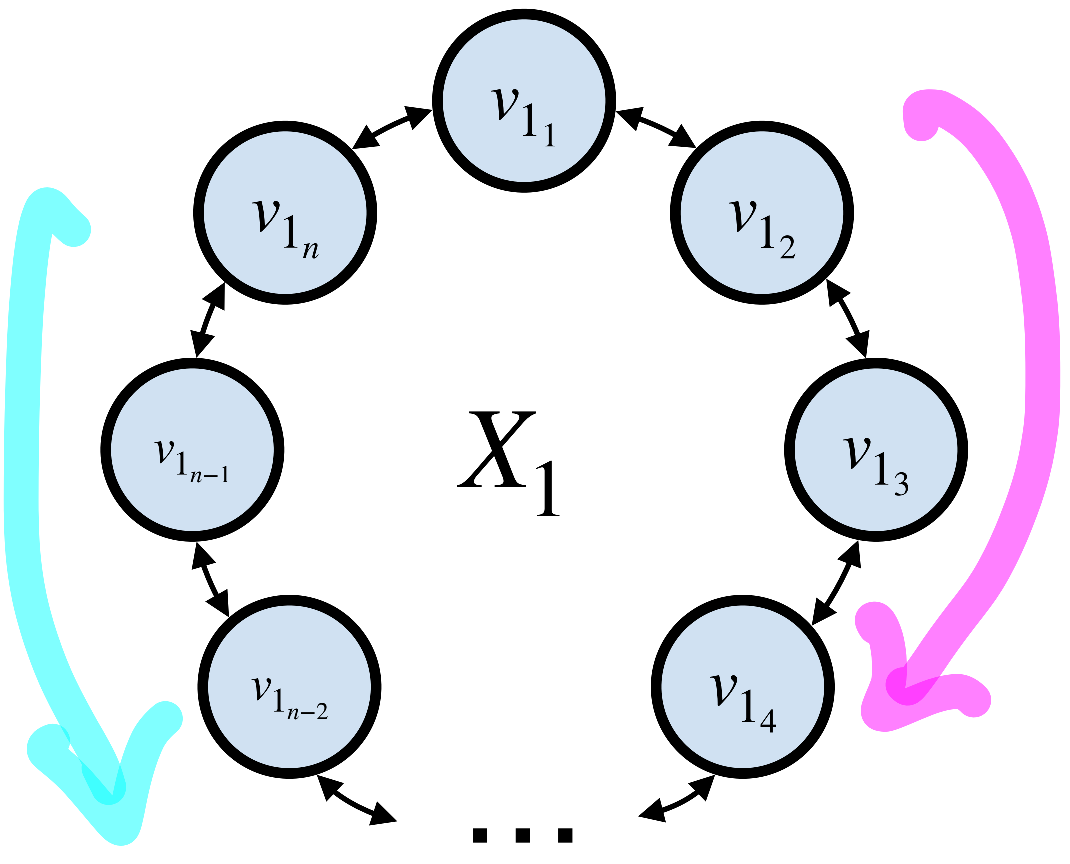
Encoding idea II

How: $f(x_1, x_2) \longleftrightarrow$ to a graph w/ a Hamiltonian cycle if 1 or no such cycle if 0.

How do we encode multiple variables?

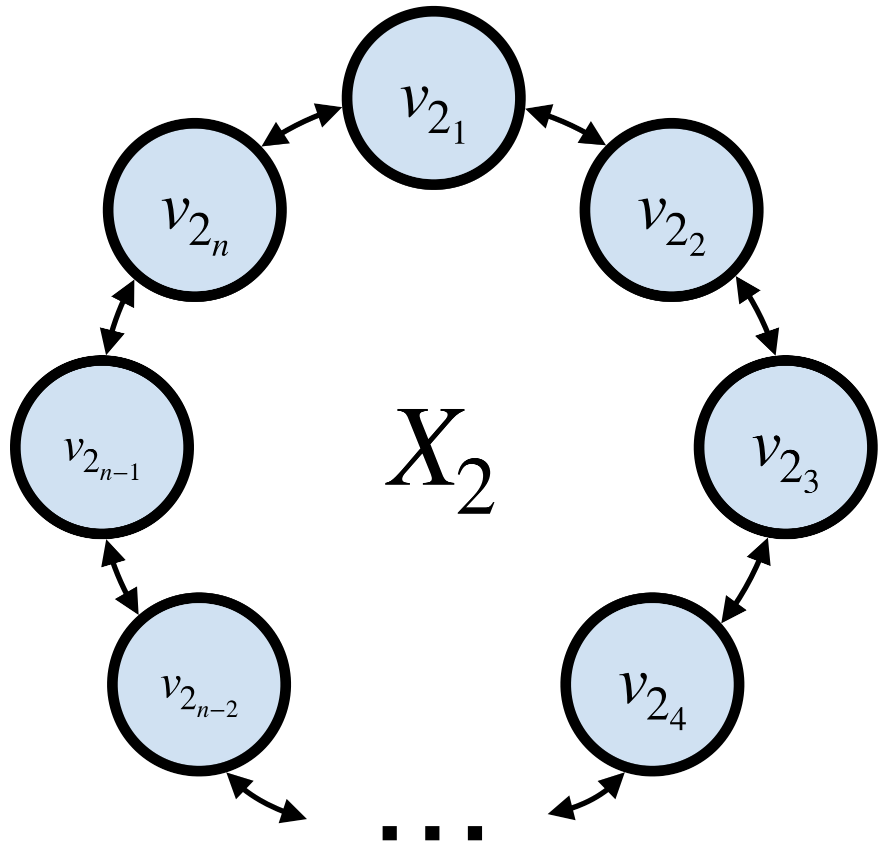
$$f(X_1, X_2) = 1$$

Maybe two circles?



if $x_1 = 1$

if $x_1 = 0$



Similarly

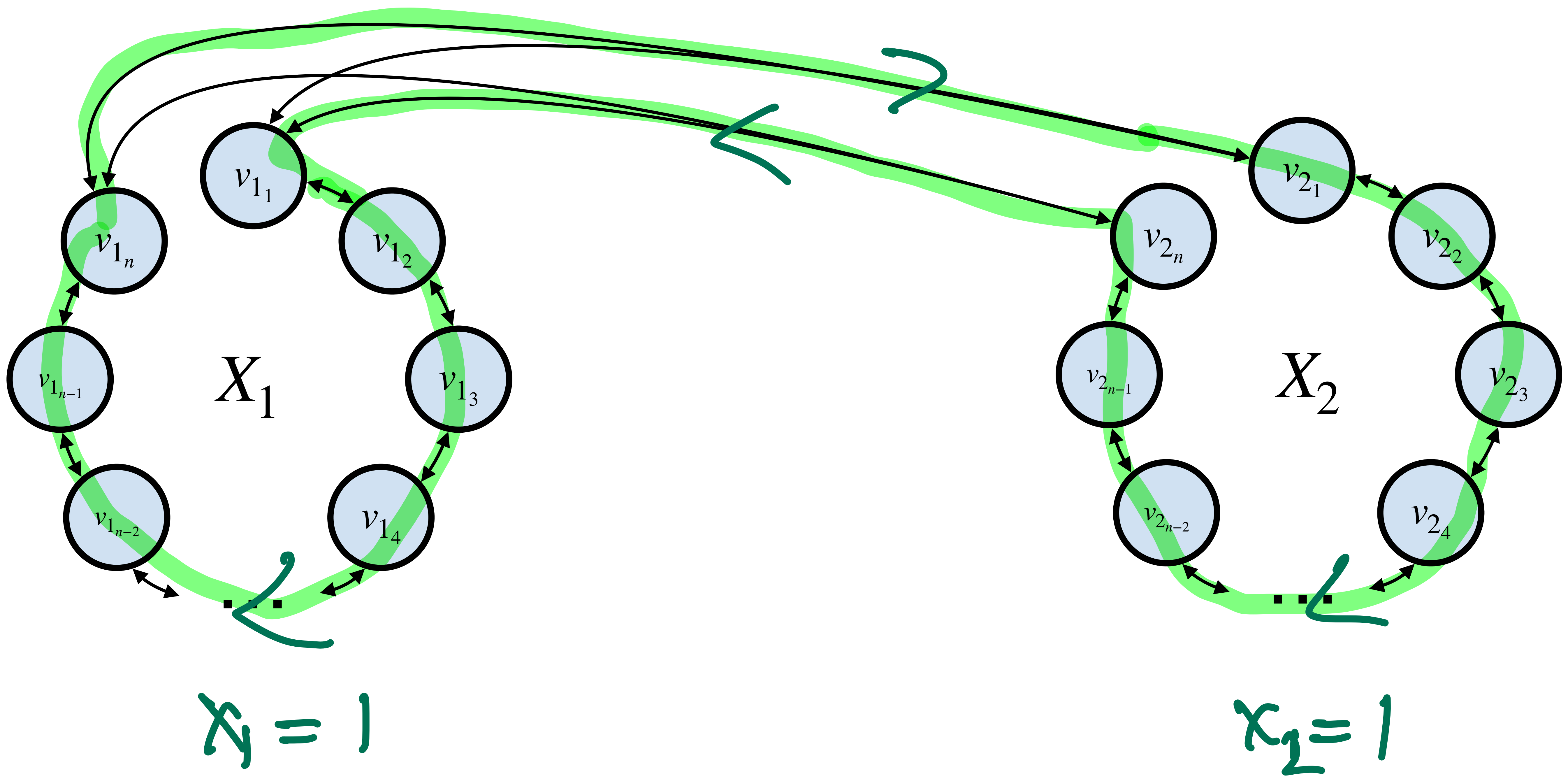
Reduction

Encoding idea II

How do we encode multiple variables?

$$f(X_1, X_2) = 1$$

Need to connect them so that we have a single hamiltonian path for each possible variable assignment.



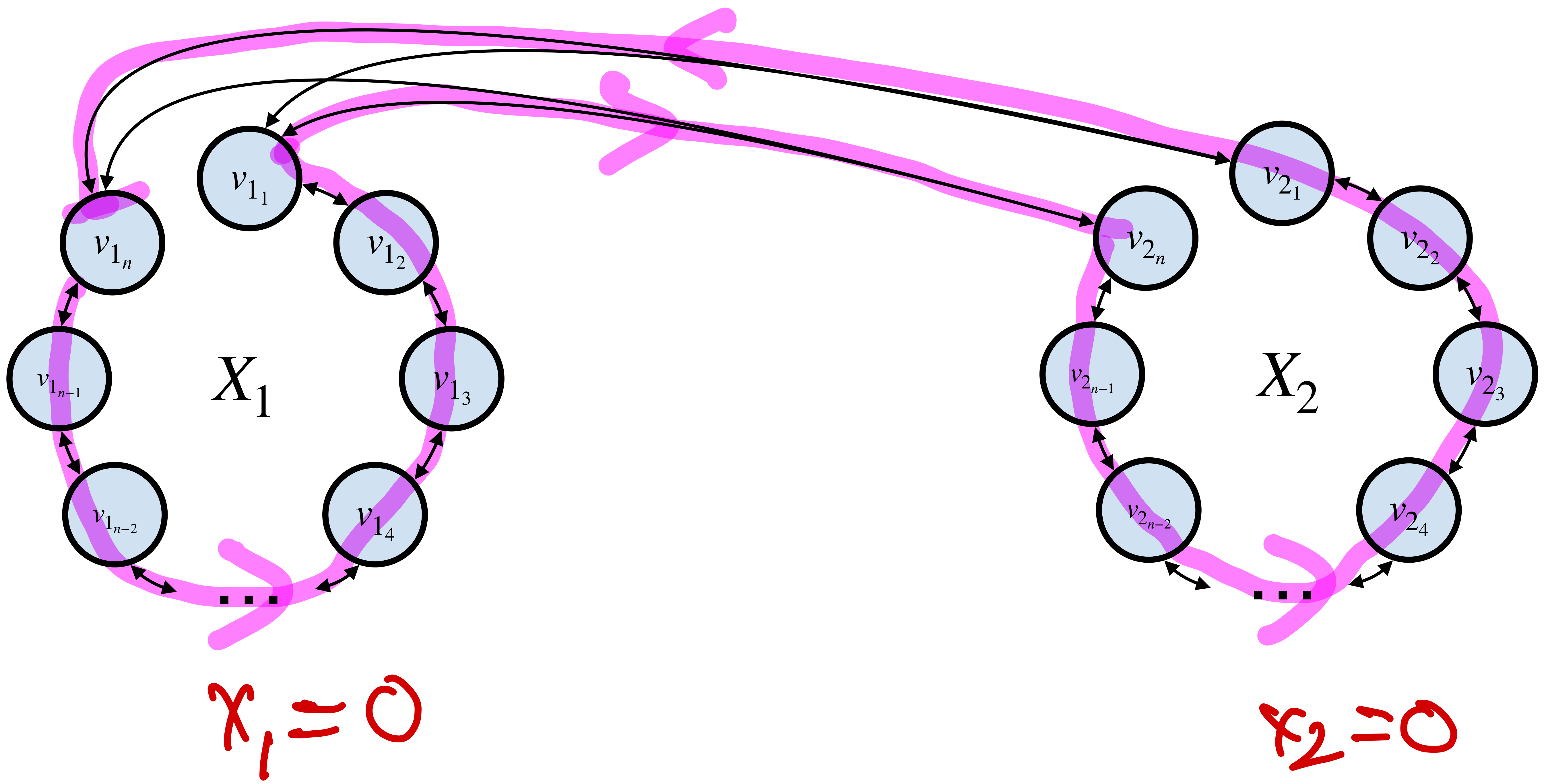
Reduction

Encoding idea II

How do we encode multiple variables?

$$f(X_1, X_2) = 1$$

Need to connect them so that we have a single hamiltonian path for each possible variable assignment.



Reduction
Encoding idea II

How do we encode multiple variables?

$$f(X_1, X_2) = 1$$

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

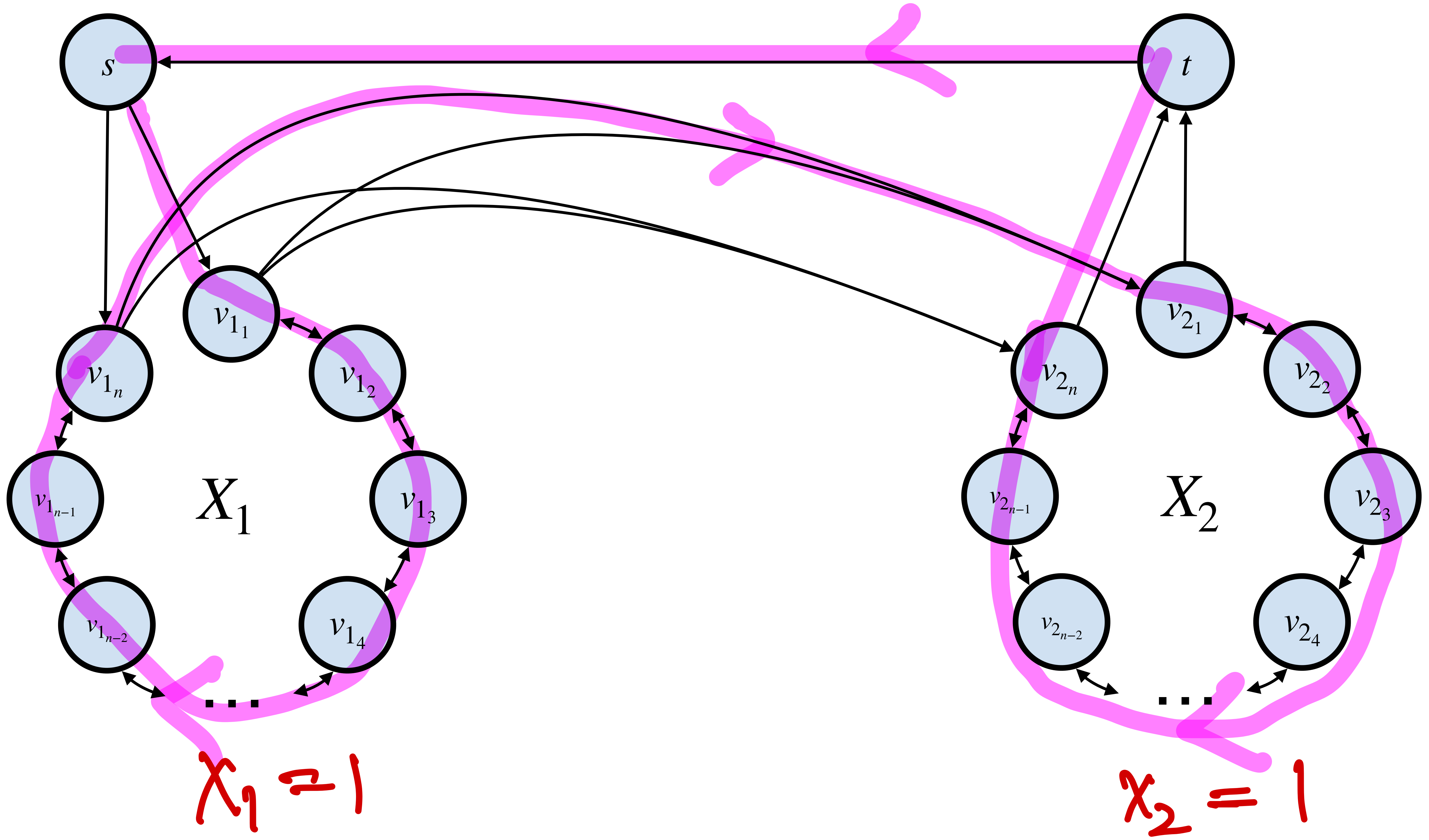
Reduction

Encoding idea II

How do we encode multiple variables?

$$f(X_1, X_2) = 1$$

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



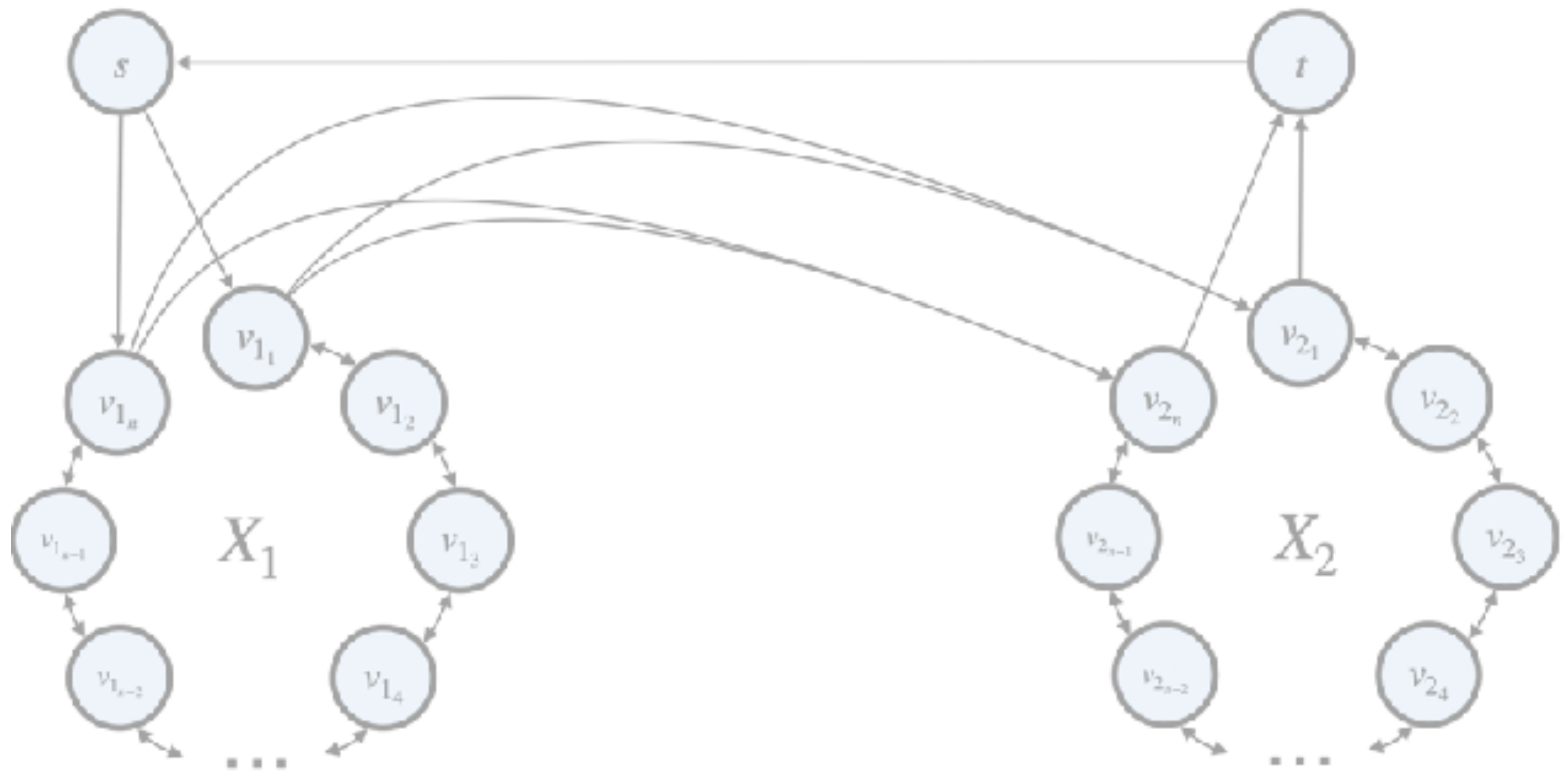
Reduction
Encoding idea II

Getting a bit messy. Let's reorganize:

Reduction

Encoding idea II

Getting a bit messy. Let's reorganize:

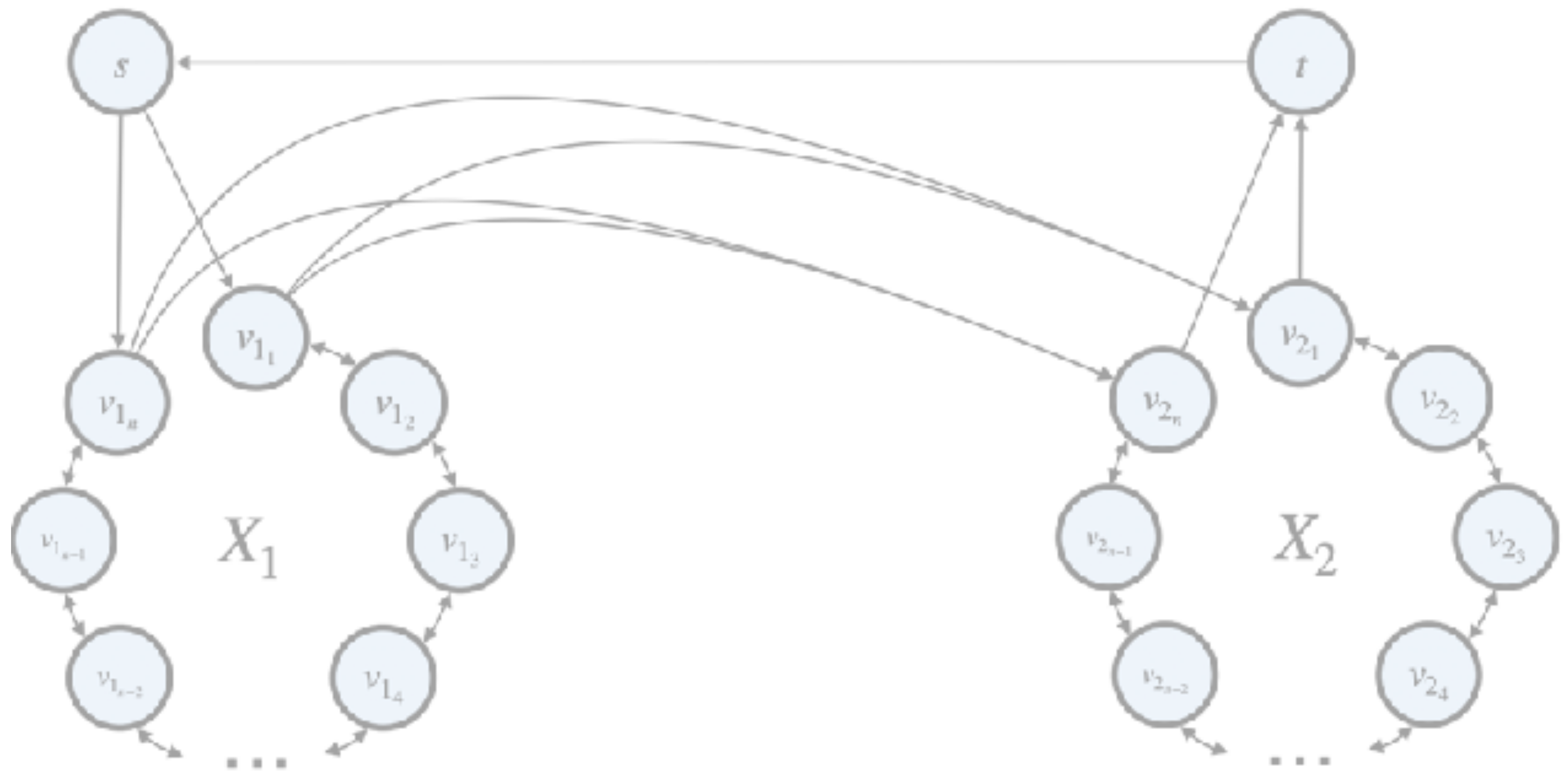


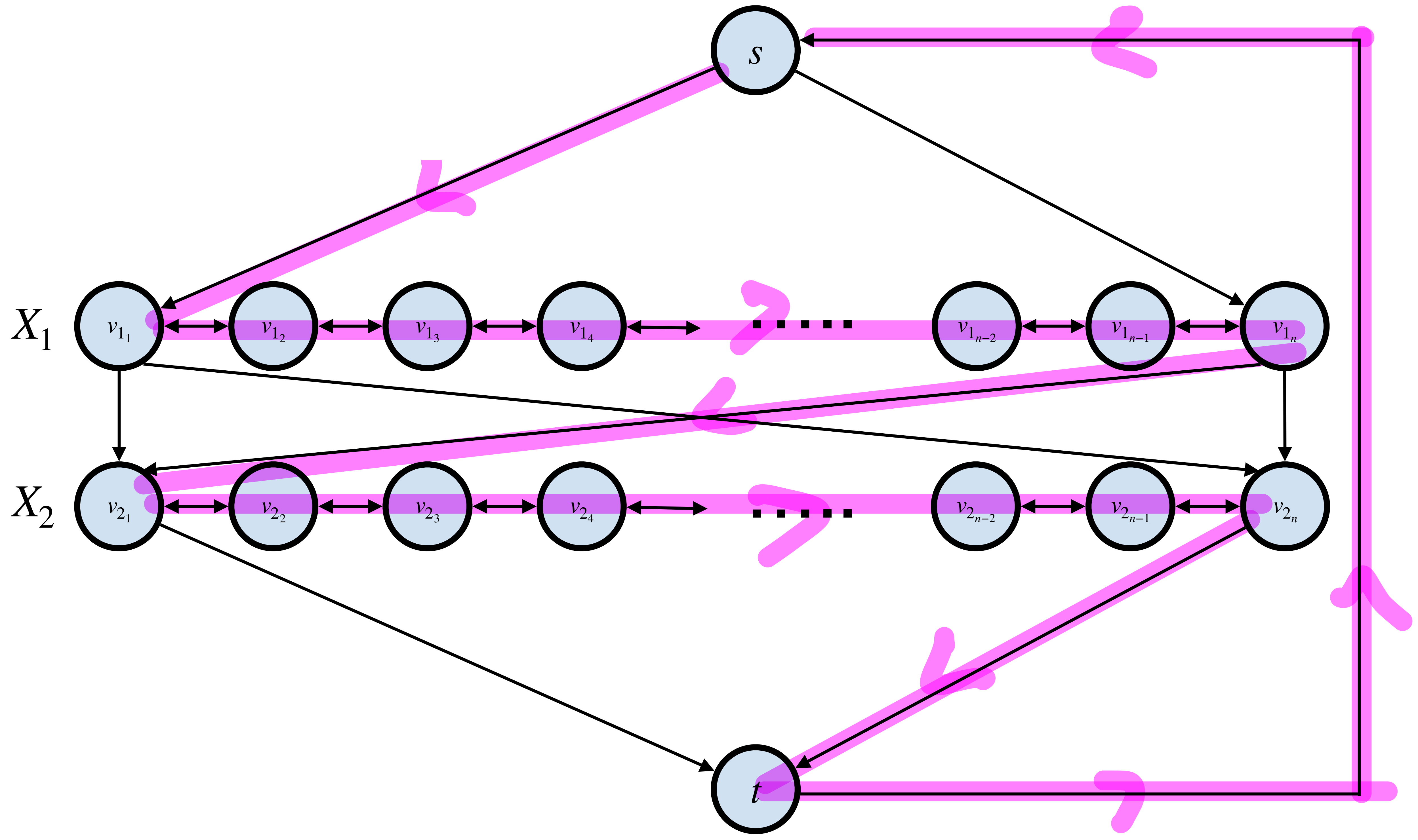
Reduction

Encoding idea II

1) CW → Left to right
2) CCW → Right to left

Getting a bit messy. Let's reorganize:





$x_1 = 1$, $x_2 = 1$

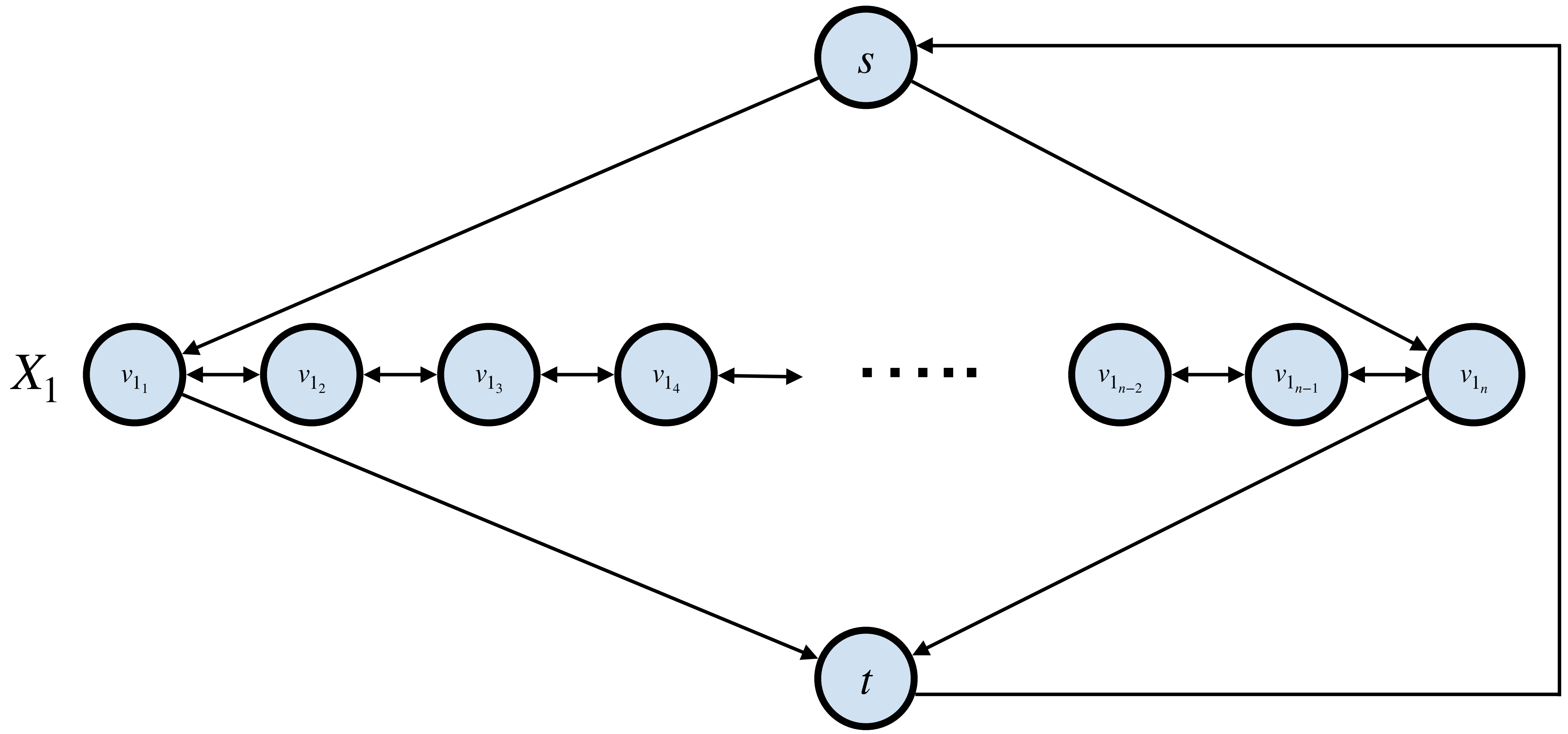
Reduction

Encoding idea III

$(- - -) \wedge (- - - -) \ldots$

How do we handle clauses ?

Lets go back to our one variable graph.

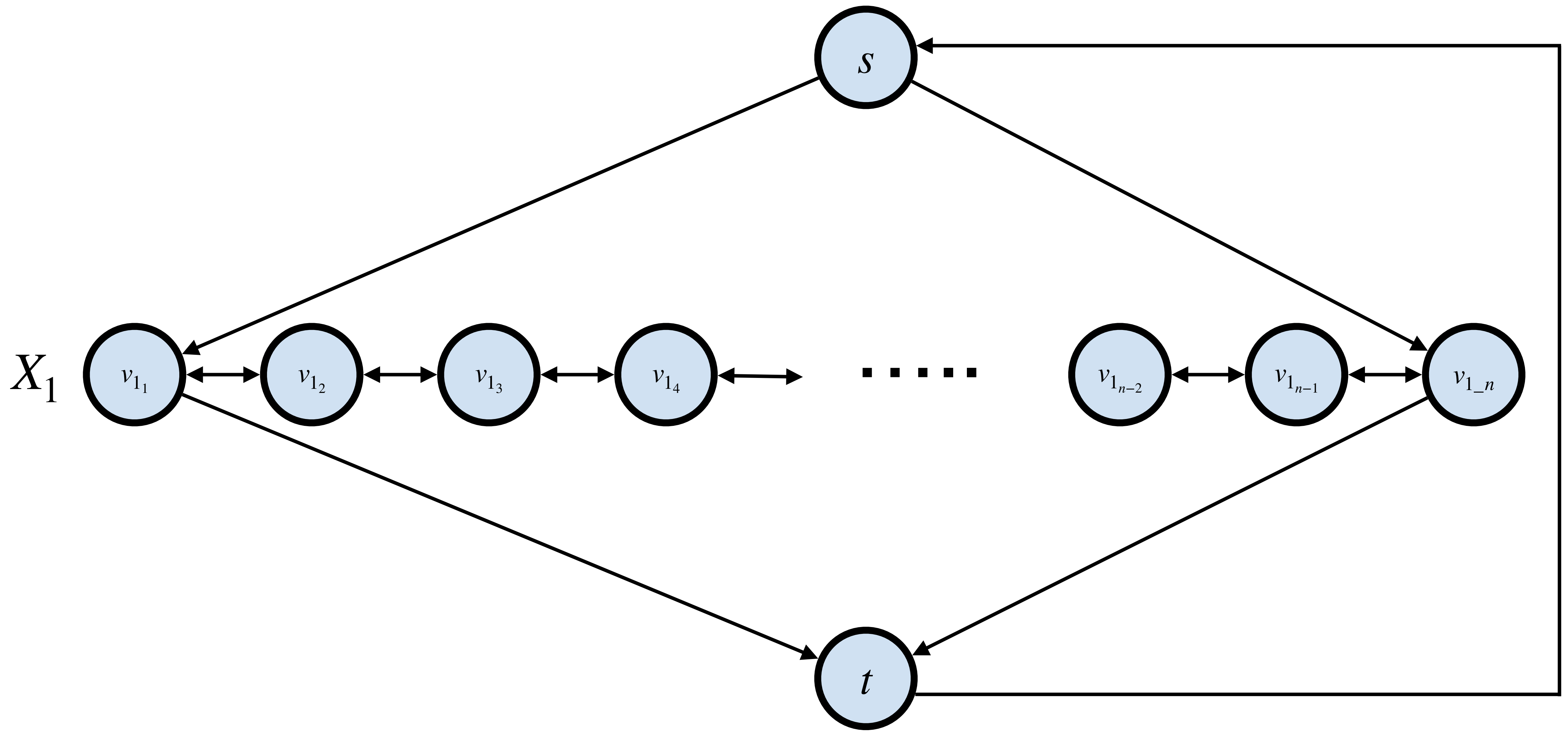


Reduction

Encoding idea III

How do we handle a clause ?

$$f(X_1) = X_1$$



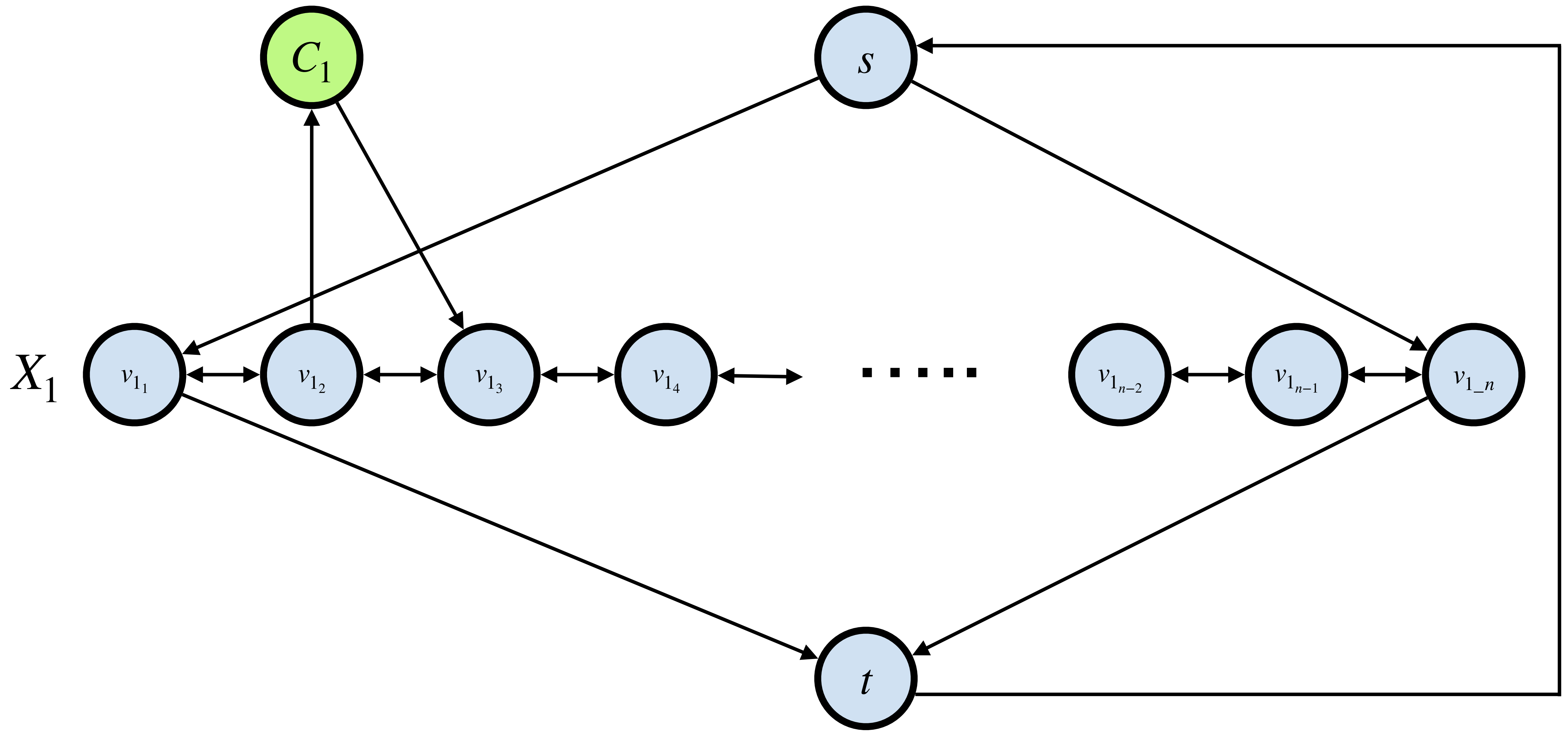
Reduction

Encoding idea III

How do we handle a clause ?

$$f(X_1) = X_1$$

Add node for clause.



Reduction

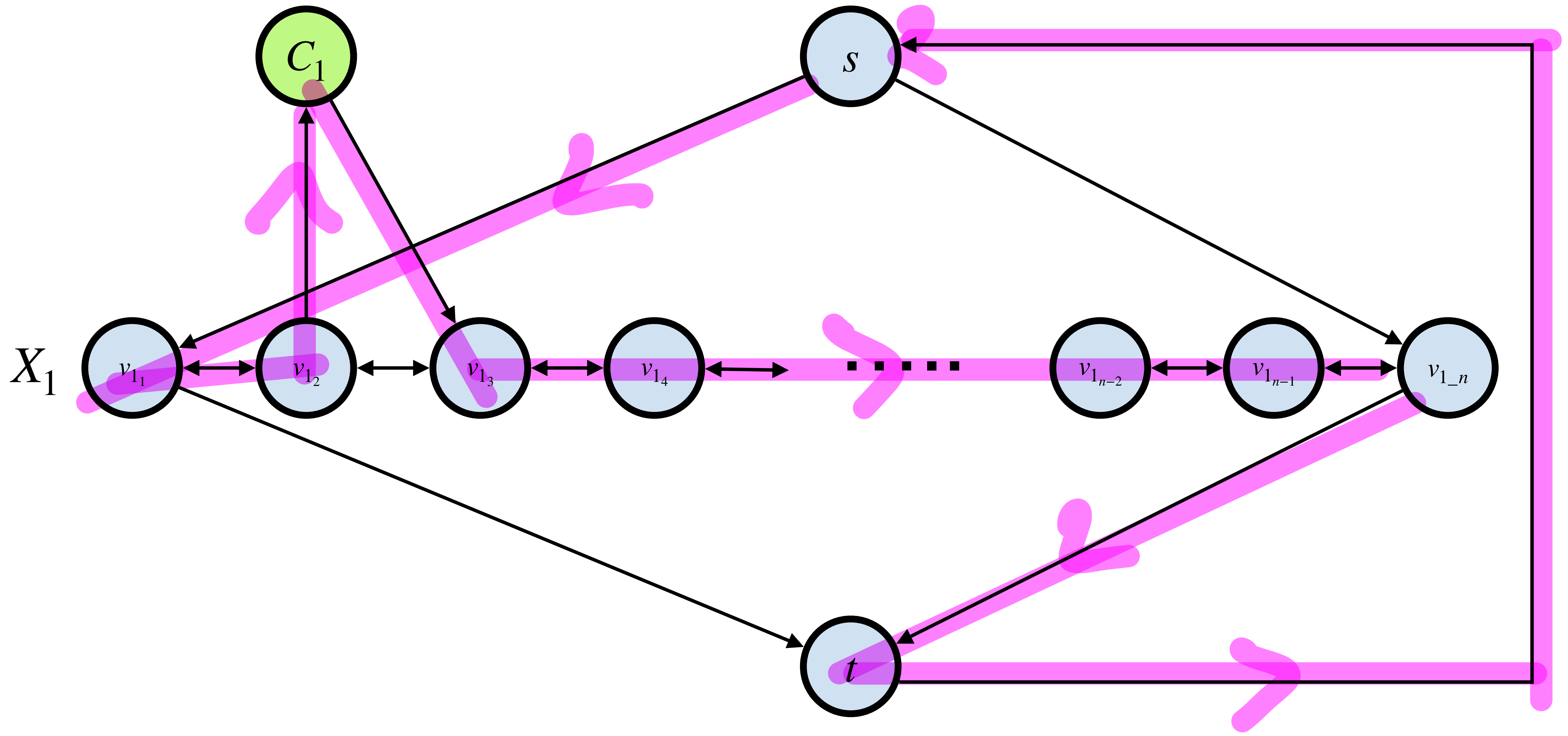
Encoding idea III

How do we handle a clause ?

$$f(X_1) = X_1$$

Add node for clause.

Enforces traversal in single direction.



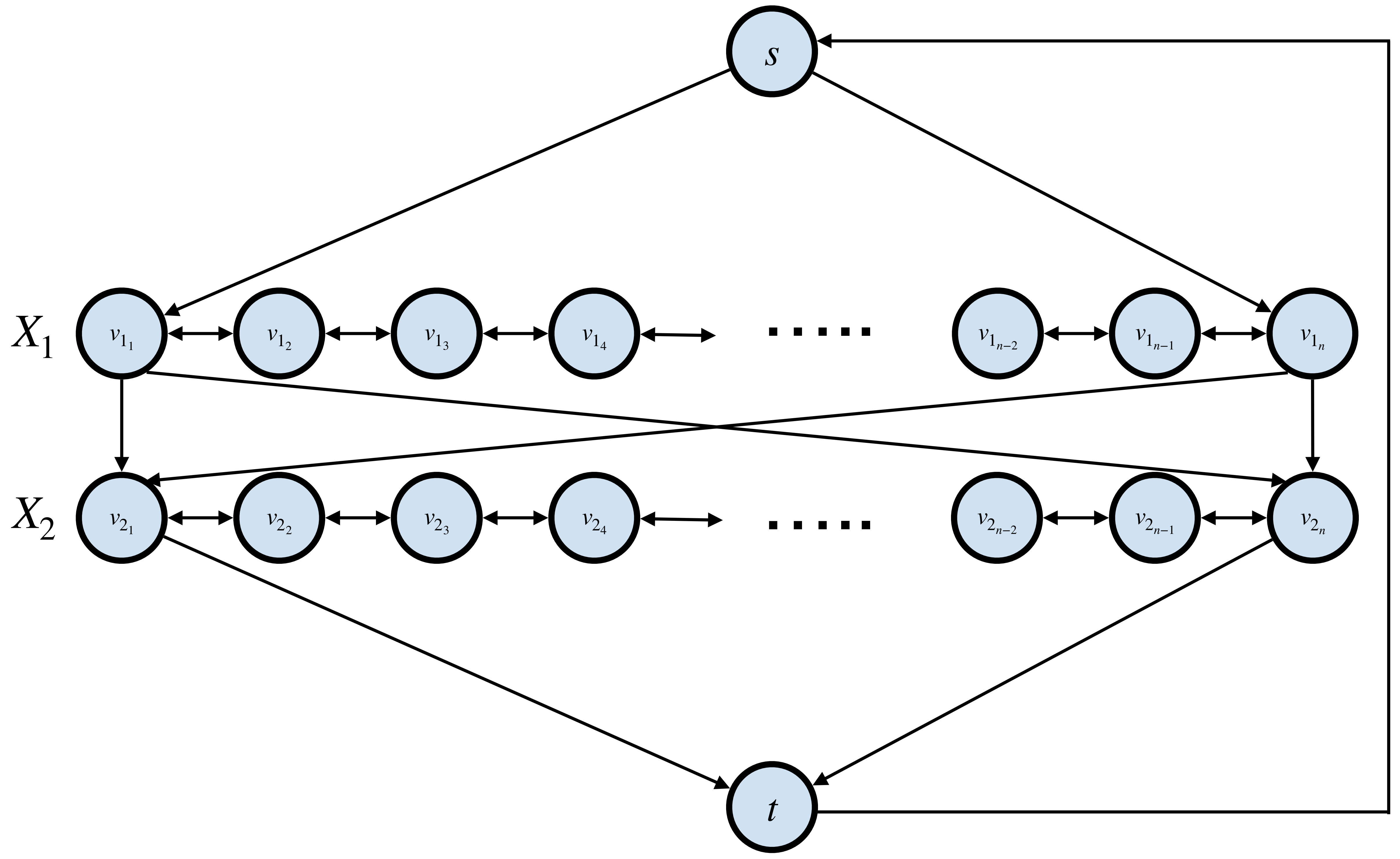
Reduction

Encoding idea III

How do we handle a clause ?

What do we do if the clause has **two literals**?

$$f(X_1, X_2) = (X_1 \vee \overline{X_2})$$



Reduction

Encoding idea III

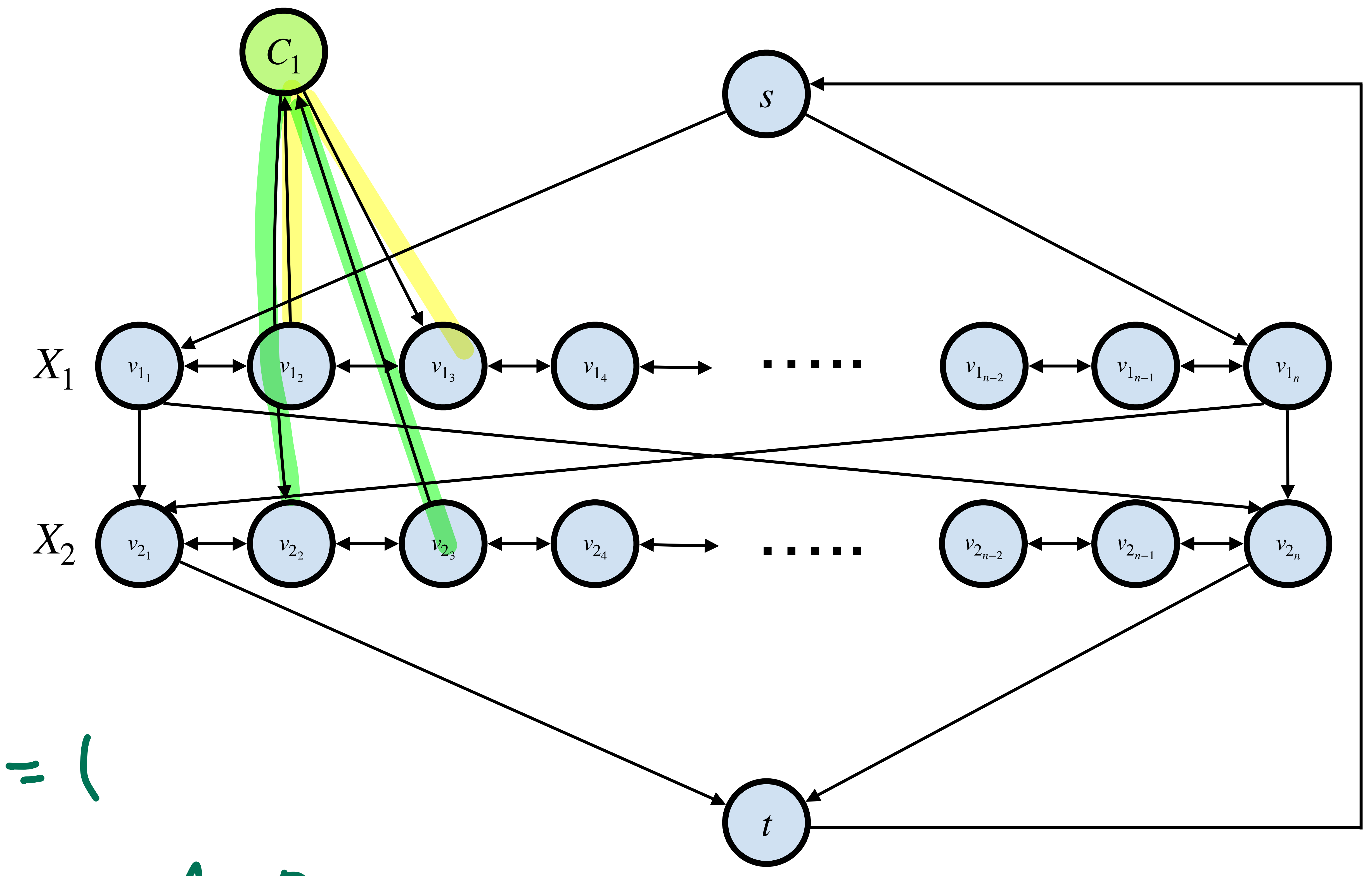
How do we handle a clause ?

What do we do if the clause has **two literals**?

$$f(X_1, X_2) = (X_1 \vee \overline{X_2})$$

Assume $X_1 = X_2 = 1$

X_2 now traveled $R \to L$



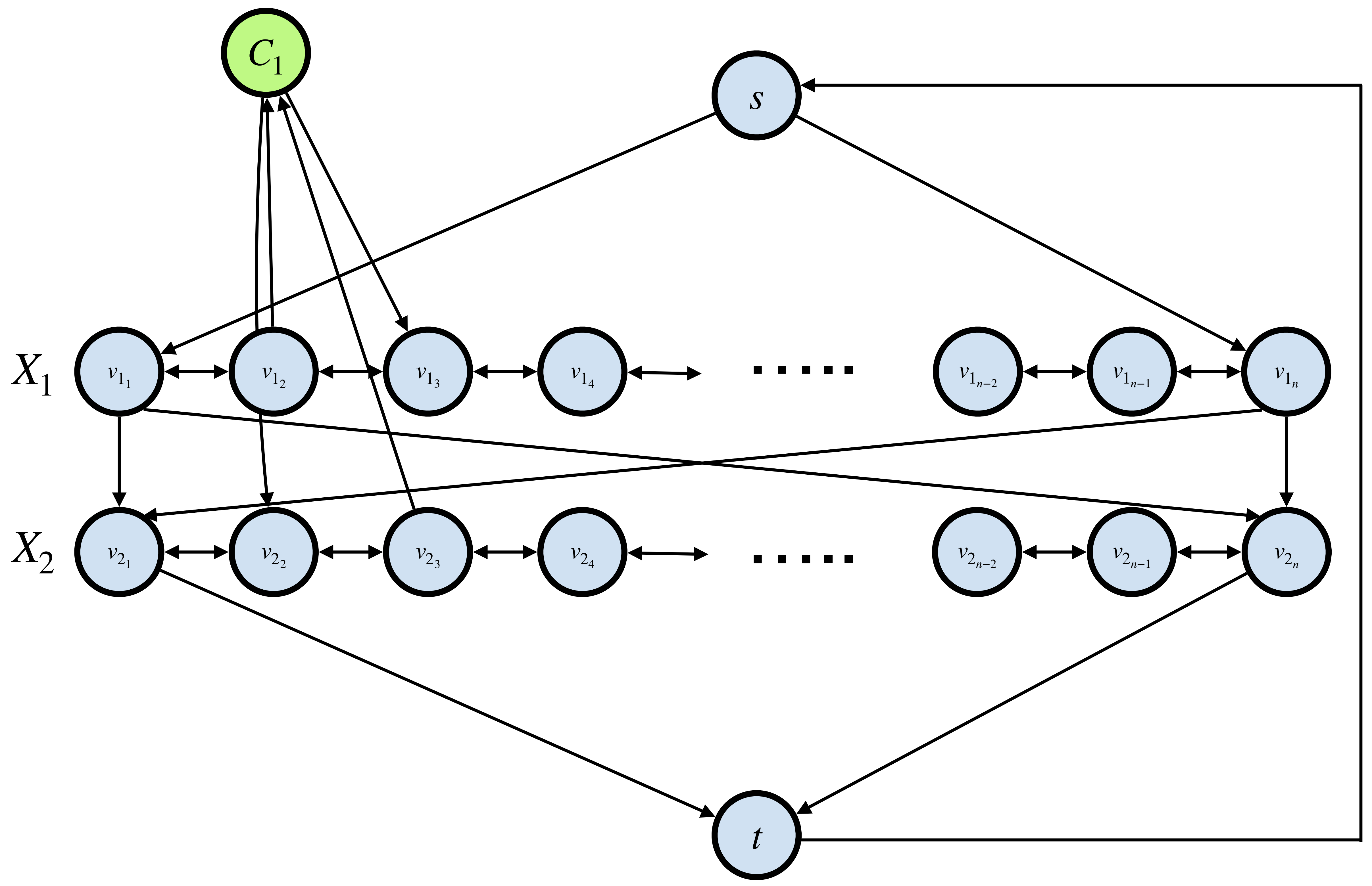
Reduction

Encoding idea III

How do we handle clauses ?

What if the expression has **multiple clauses**?

$$f(X_1, X_2) = (X_1 \vee \overline{X_2}) \wedge (\overline{X_1} \vee X_2)$$



Reduction

Encoding idea III

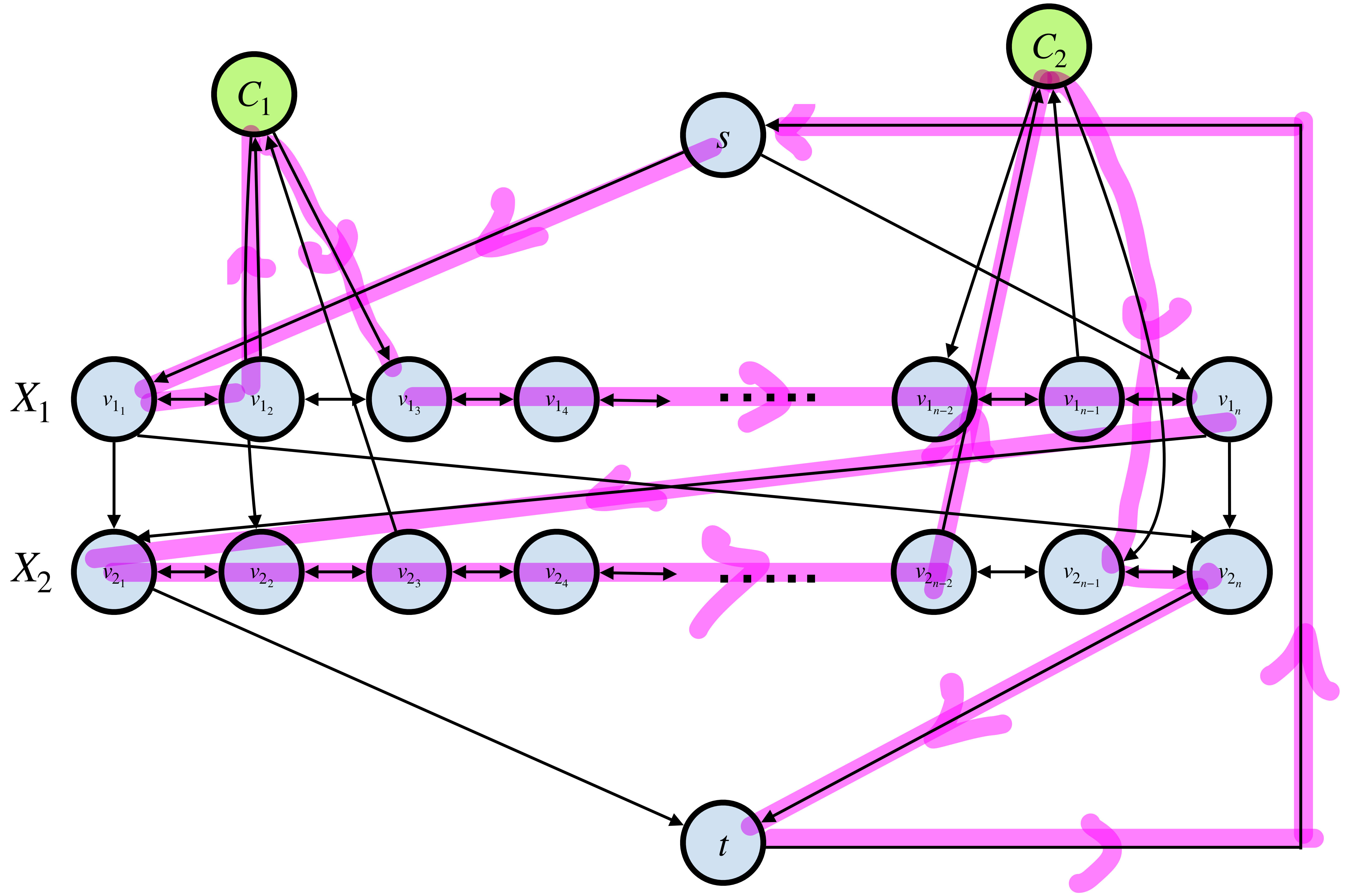
How do we handle clauses ?

What if the expression has **multiple clauses**?

$$f(X_1, X_2) = (X_1 \vee \overline{X_2}) \wedge (\overline{X_1} \vee X_2)$$

$(1 \vee 0) \wedge (0 \vee 1)$

$$\boxed{\begin{array}{l} X_1 = 1 \\ X_2 = 1 \end{array}}$$



Reduction

Encoding idea III

How do we handle clauses ?

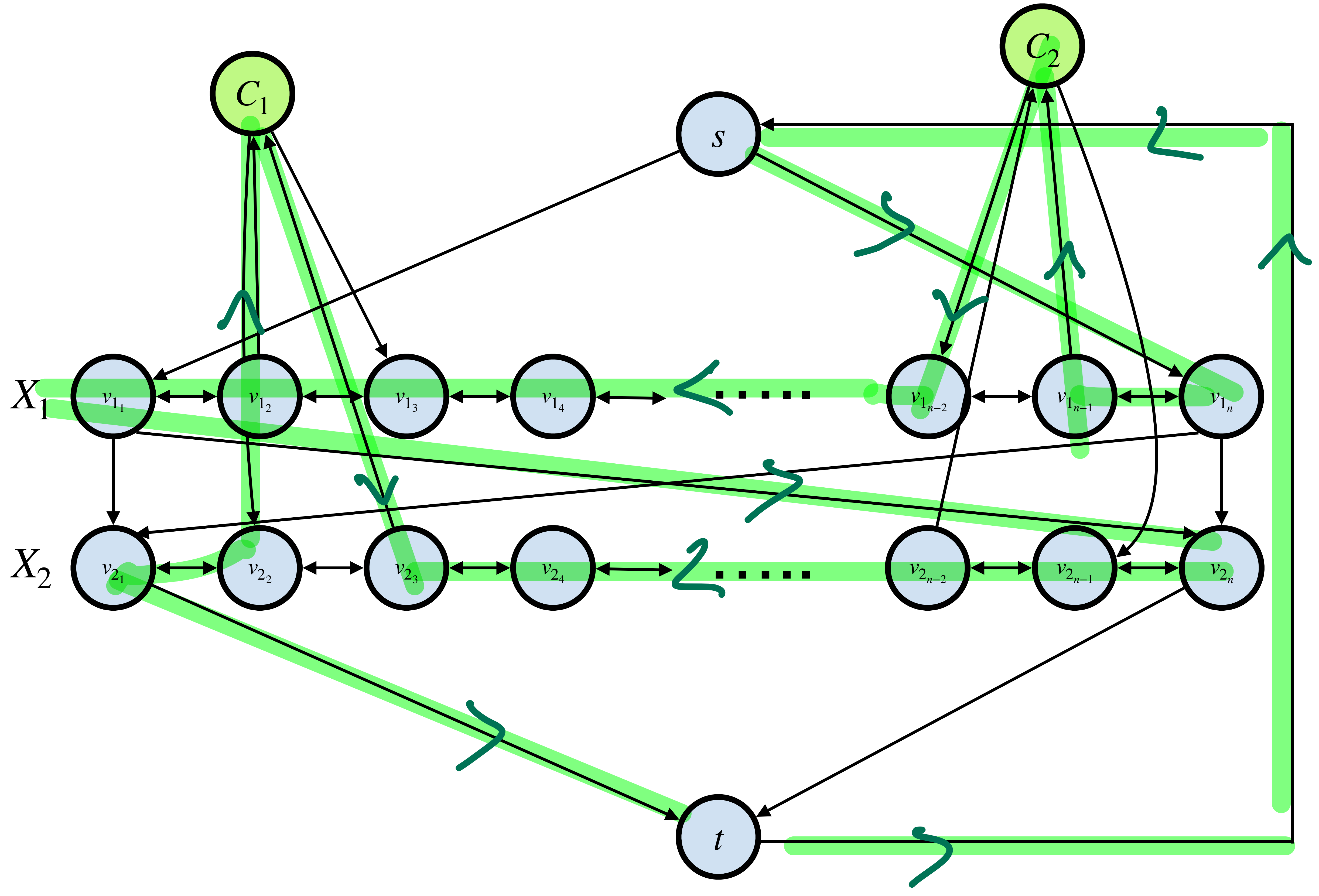
What if the expression has **multiple clauses**?

$$f(X_1, X_2) = (X_1 \vee \overline{X_2}) \wedge (\overline{X_1} \vee X_2)$$

$(0 \vee 1) \wedge (1 \vee 0)$

$x_1 = 0$

$x_2 = 0$



The Reduction

Review I

The Reduction
Review I

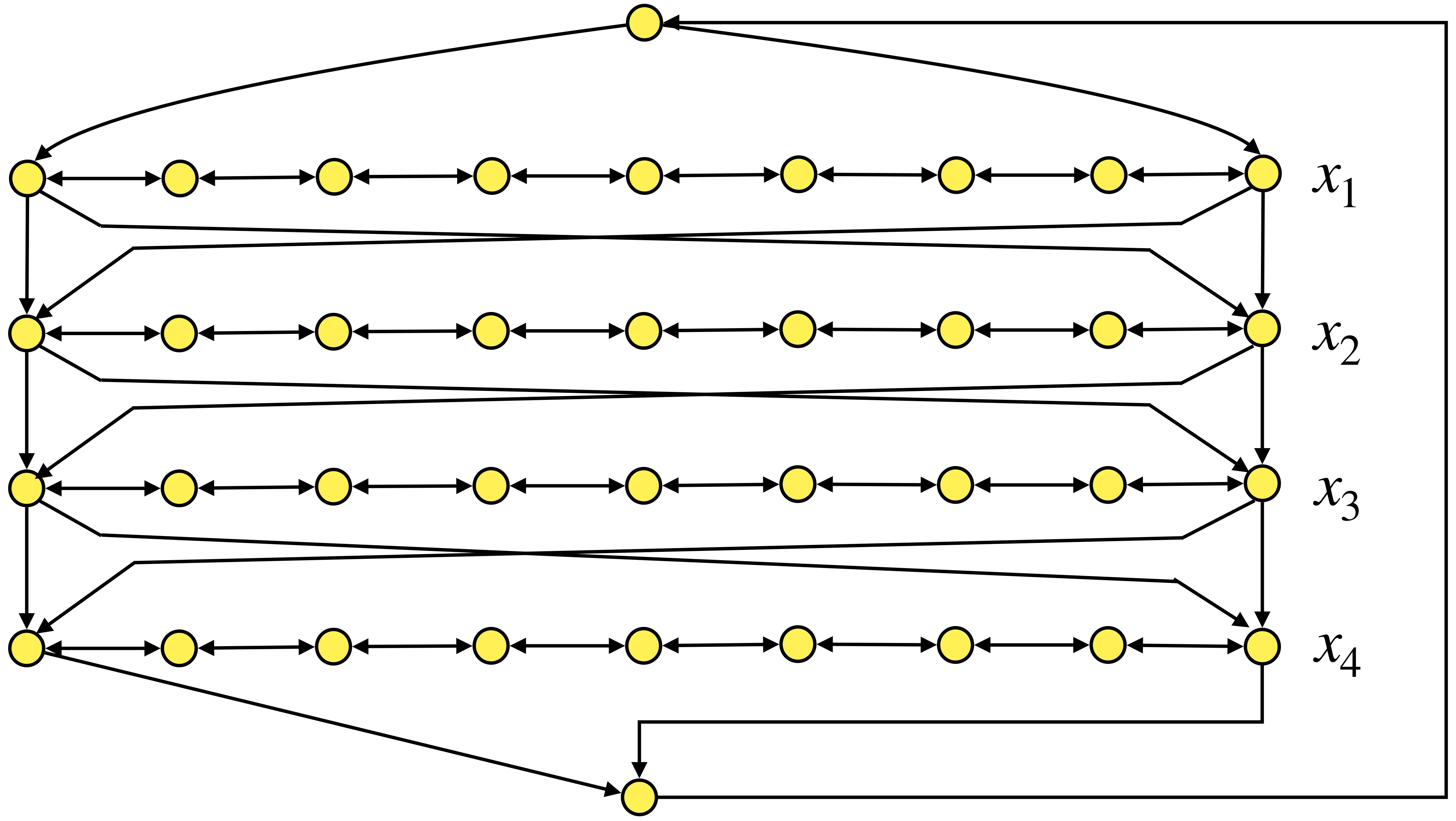
$\longrightarrow$ iff $x_i = 1$

- Traverse path i from left to right if and only if x_i is set to true

The Reduction

Review I

- Traverse path i from left to right if and only if x_i is set to true

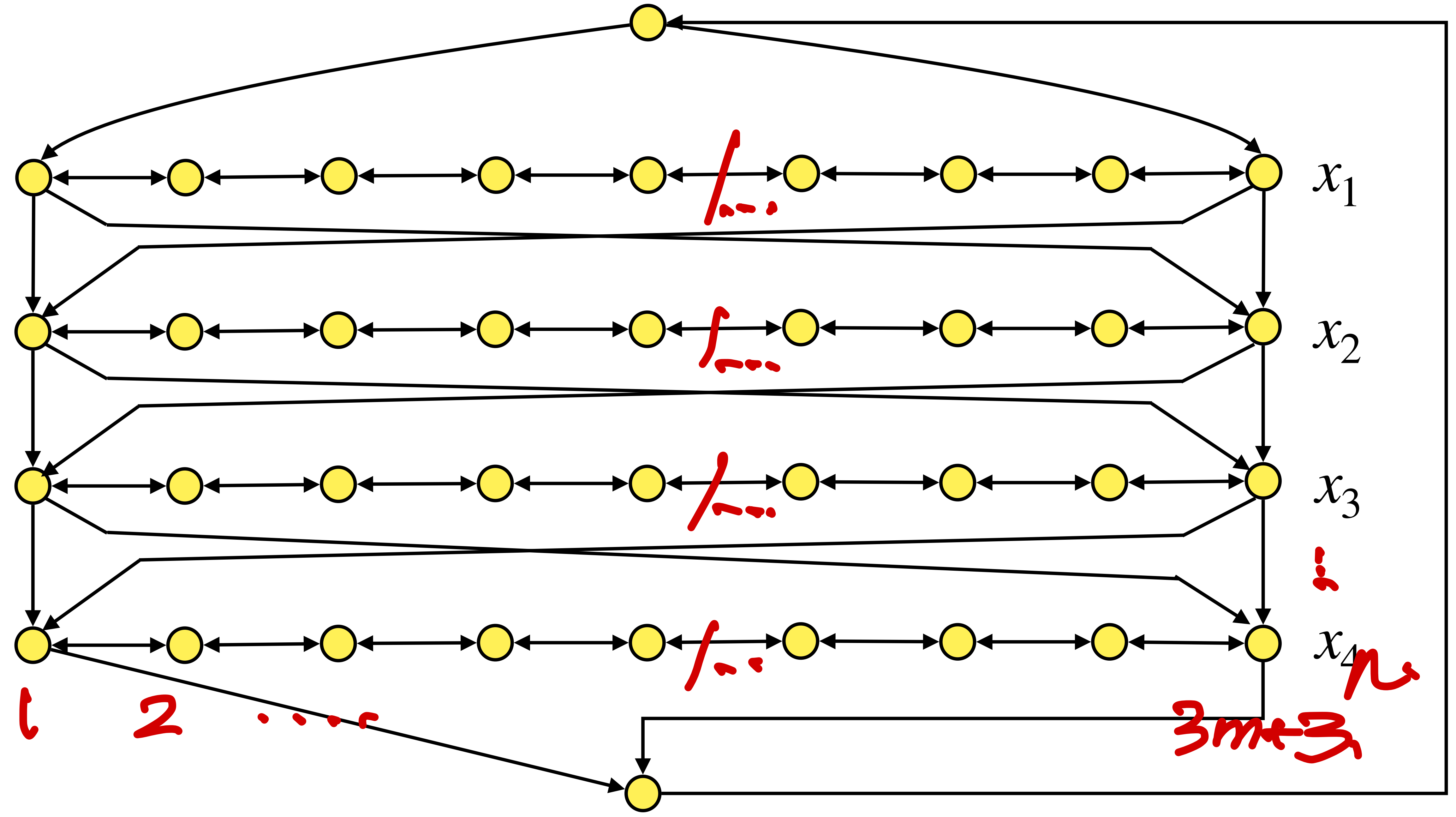


The Reduction

Review I

- Traverse path i from left to right if and only if x_i is set to true

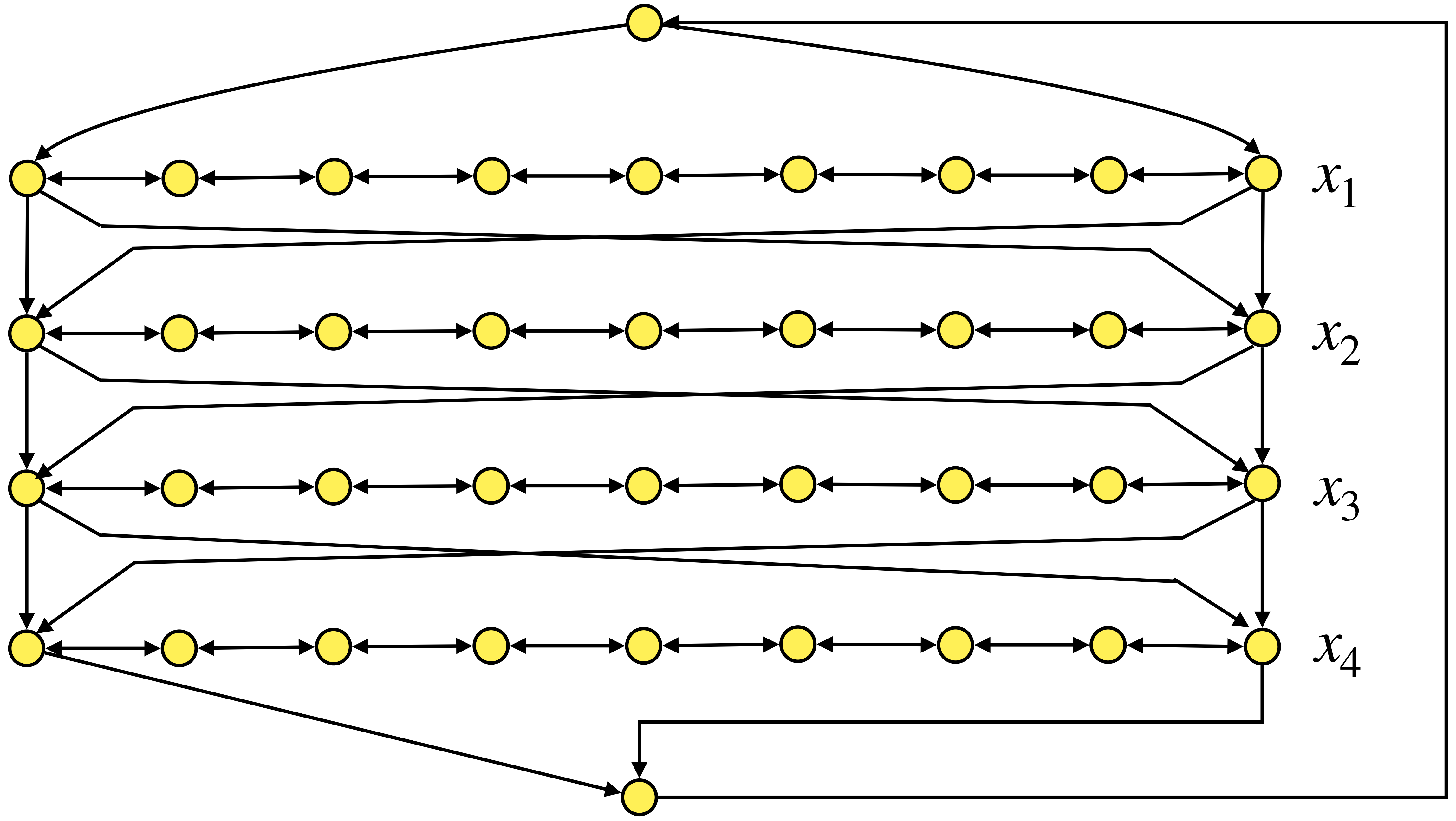
- Each path has $3(m+1)$ nodes where m is number of clauses in φ; nodes numbered from left to right $(1 \text{ to } 3m+3)$



The Reduction

Review II

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



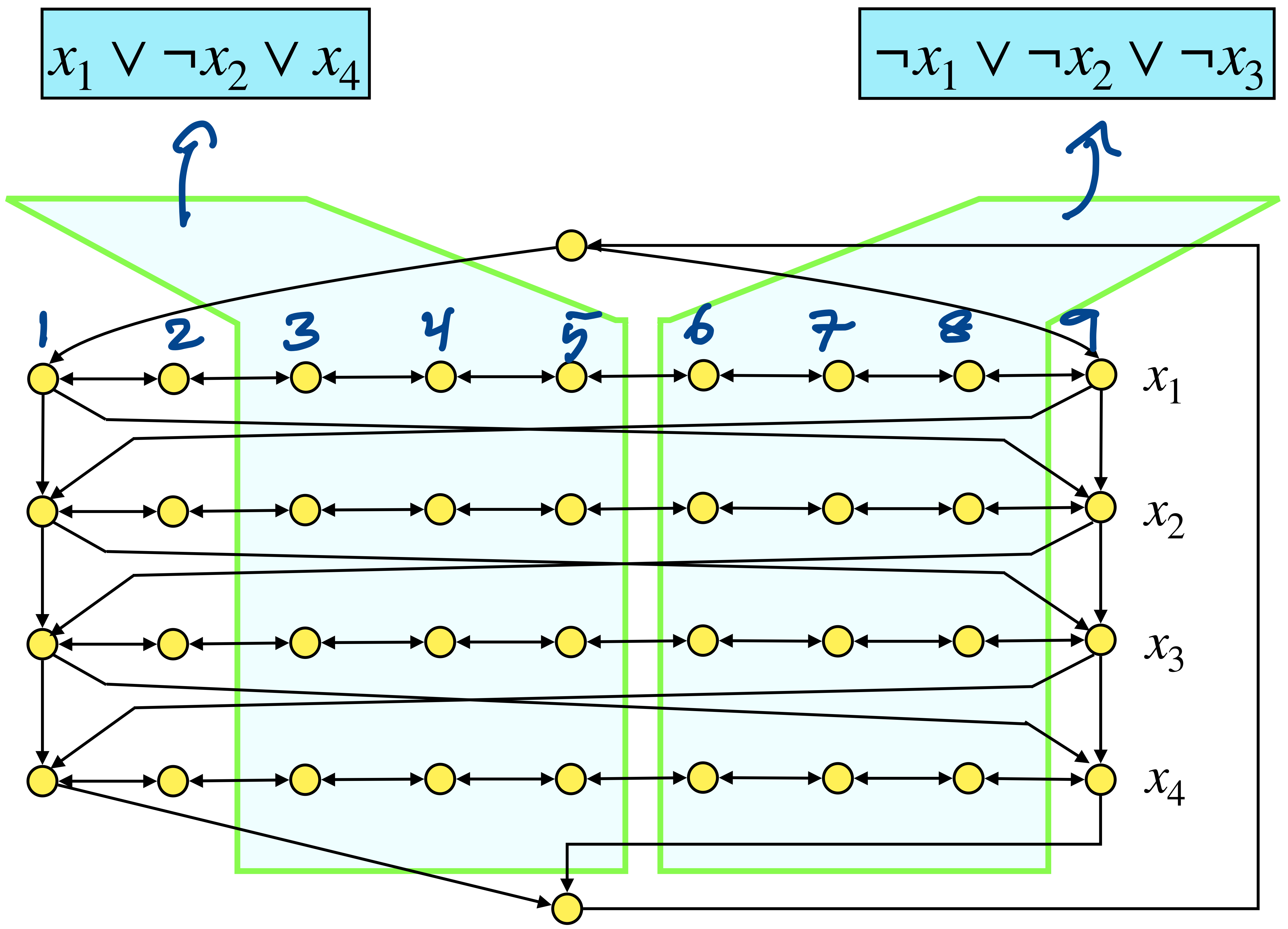
The Reduction

Review II

$m = 2$ (# of clauses)

$3(m+1) = 9$

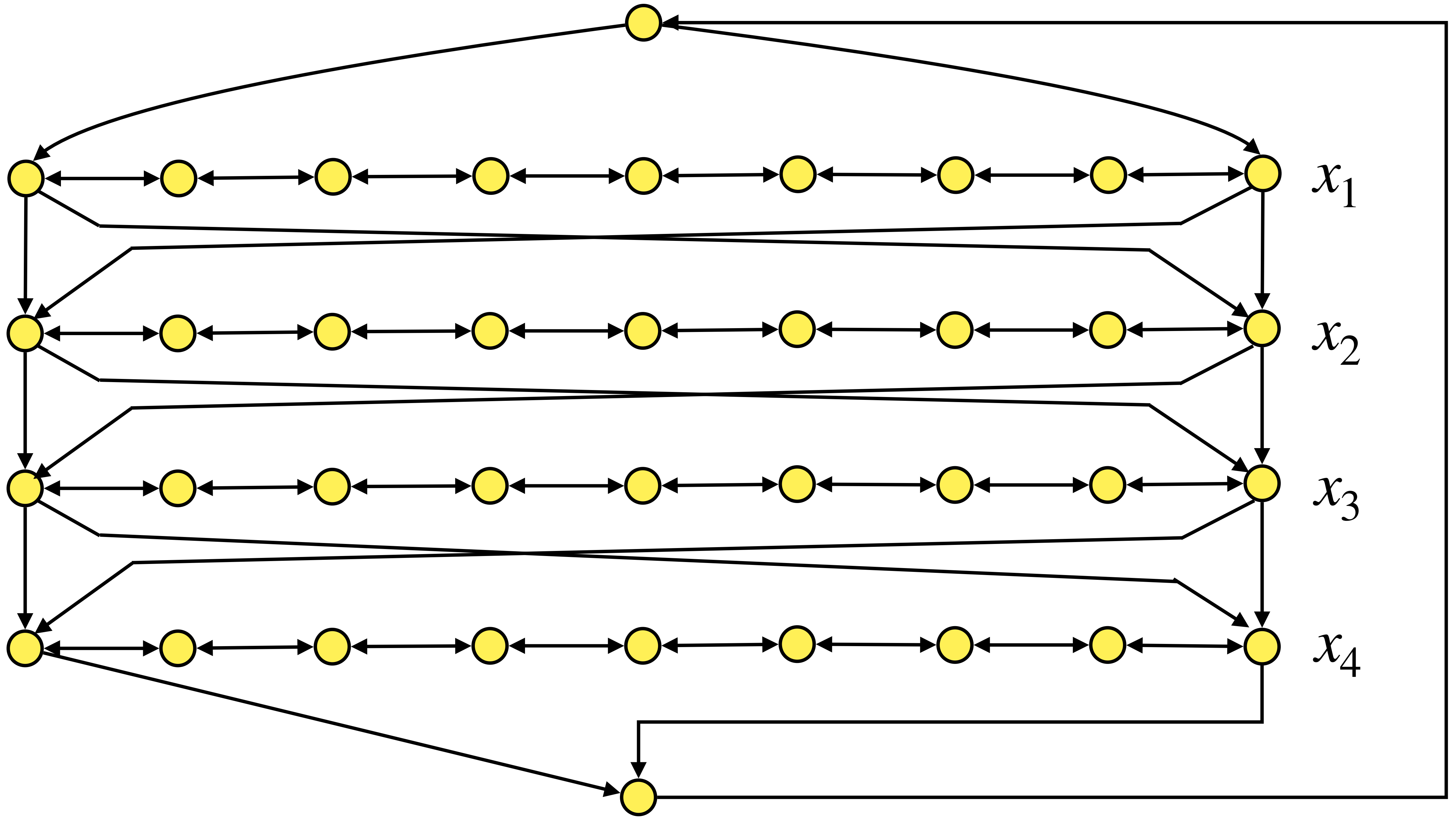
- Add vertex c_j for clause C_j .
- Vertex c_j has edge from vertex 3_j and to vertex $3_j + 1$ on path i if x_i appears in clause C_j , and
- Has edge from vertex $3_j + 1$ and to vertex 3_j if $\neg x_i$ appears in C_j .



The Reduction

Review II

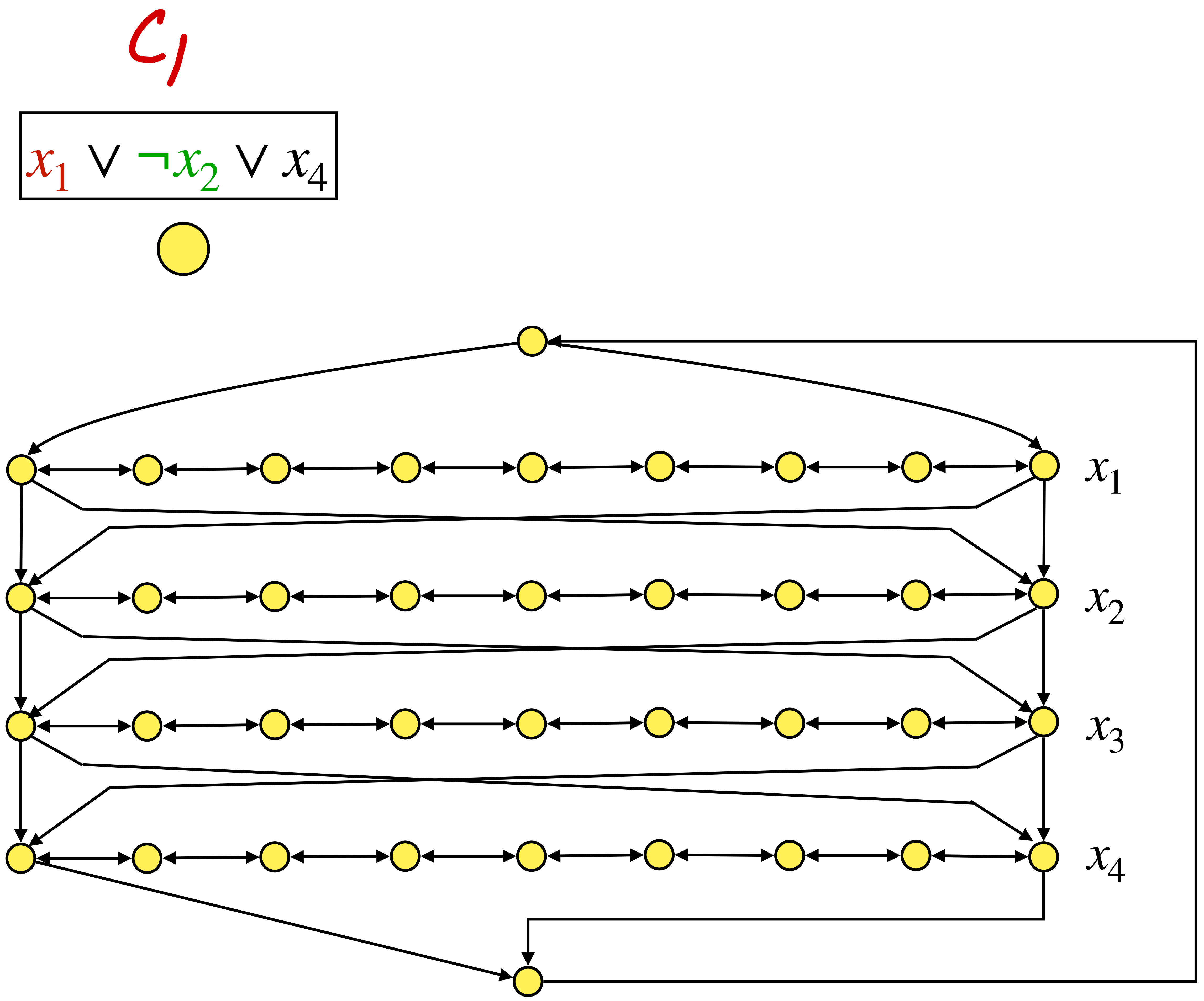
$\boxed{x_1 \vee \neg x_2 \vee x_4}$



The Reduction

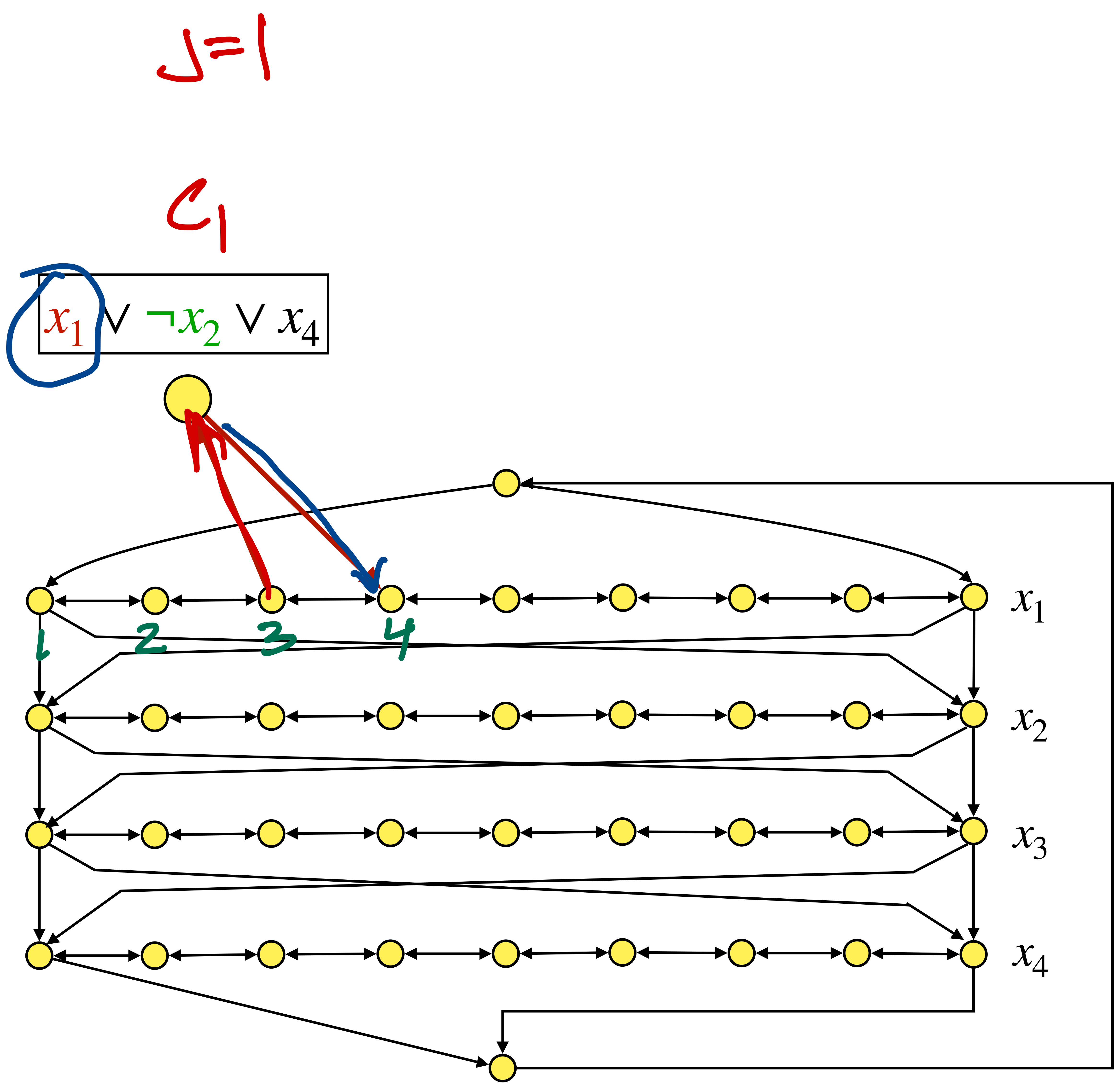
Review II

- Add vertex c_j for clause C_j .



The Reduction
Review II

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

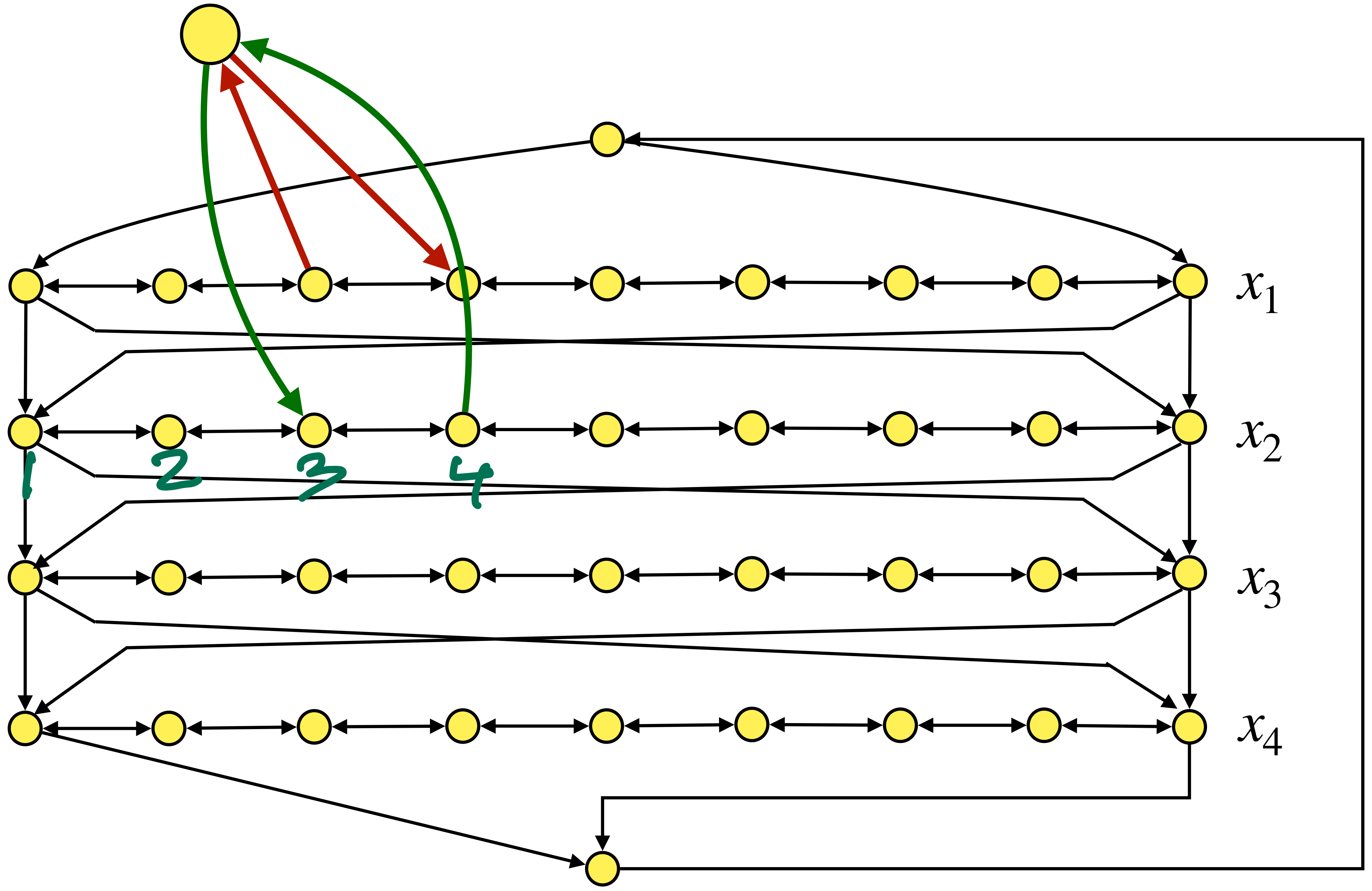


The Reduction

Review II

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

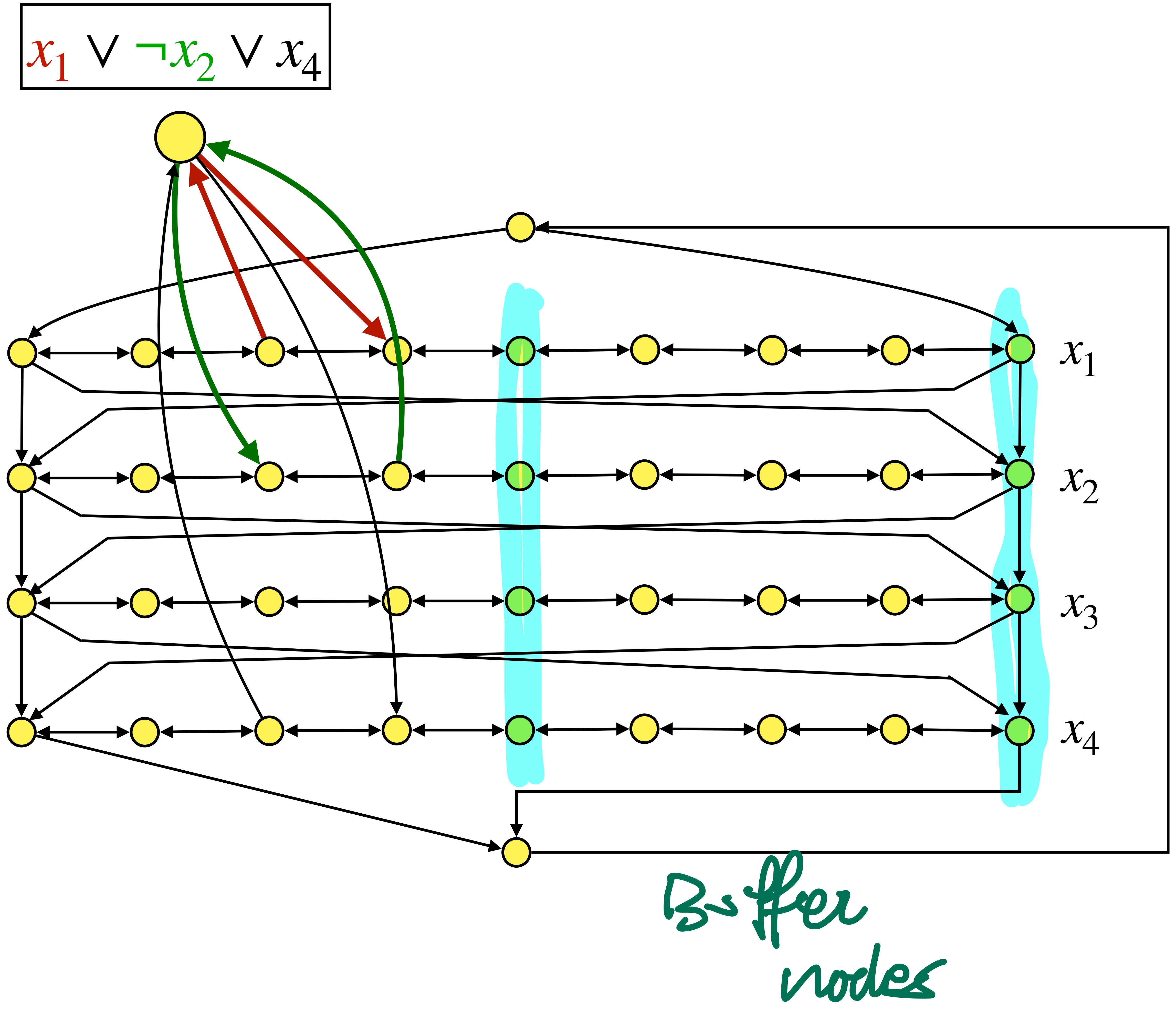
$\boxed{x_1 \vee \neg x_2 \vee x_4}$



The Reduction

Review II

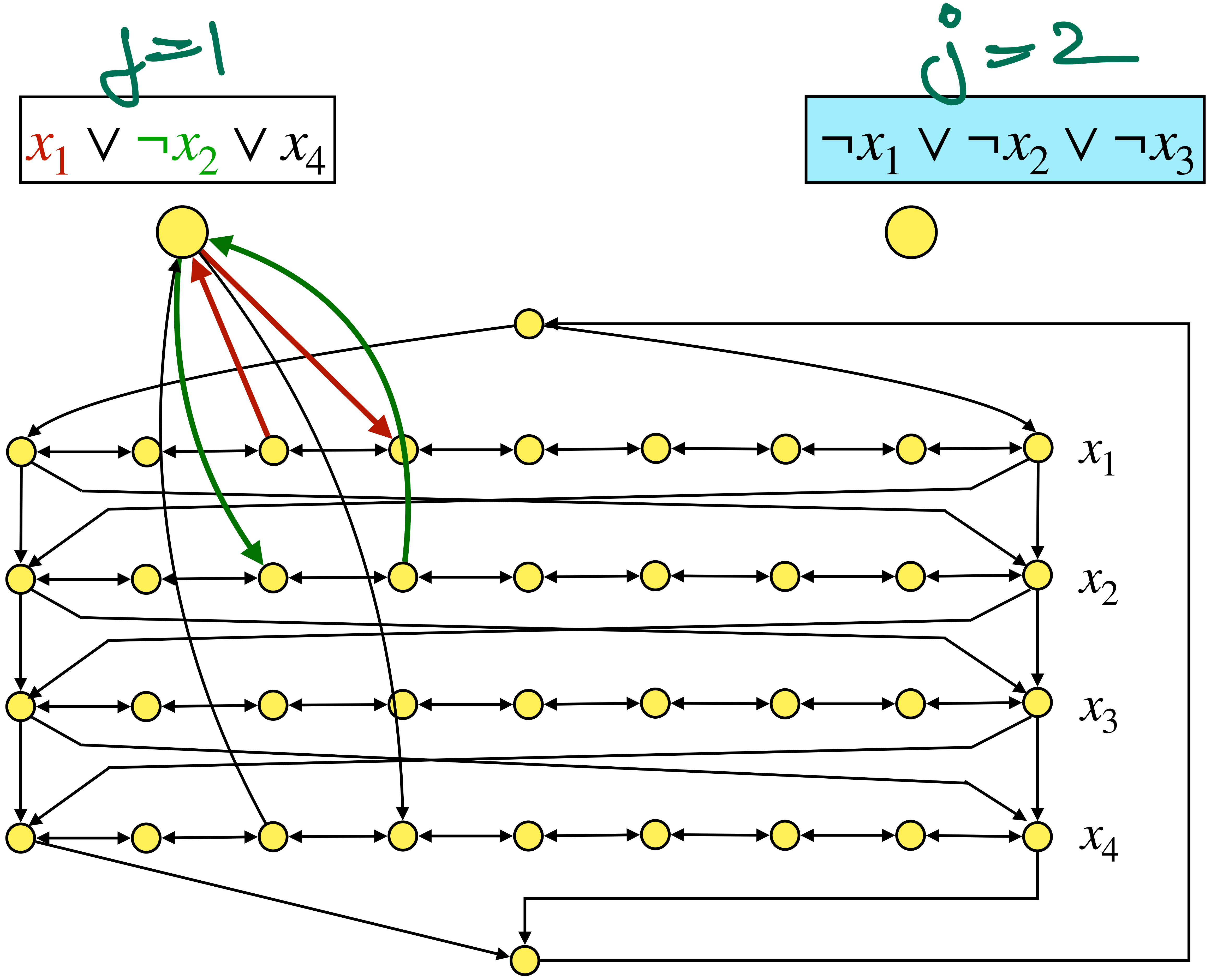
- Add vertex c_j for clause C_j .
- Vertex c_j has edge ***from*** vertex 3_j and ***to*** vertex $3_j + 1$ on path i if x_i appears in clause C_j , and
- Has edge ***from*** vertex $3_j + 1$ and ***to*** vertex 3_j if $\neg x_i$ appears in C_j .



The Reduction

Review II

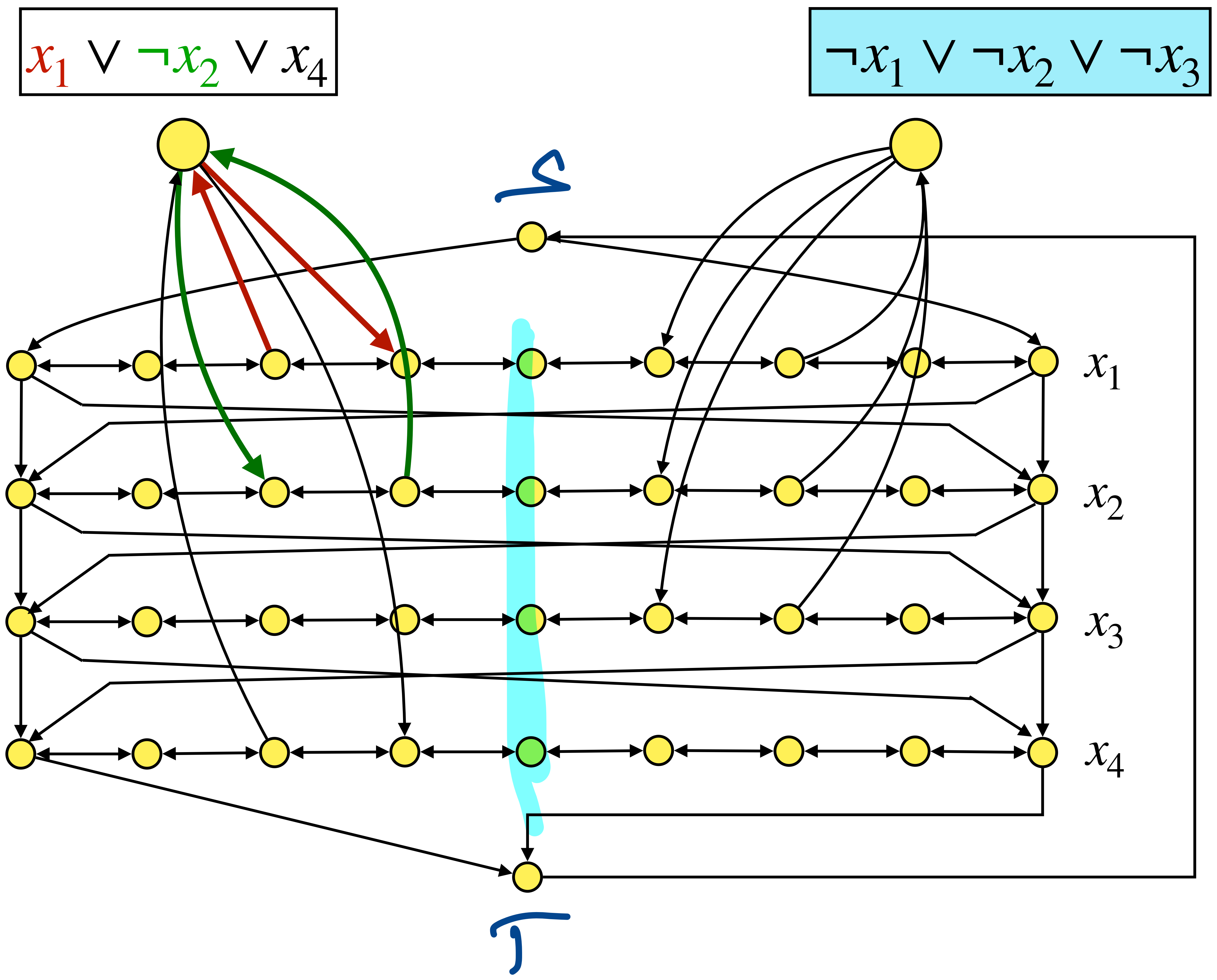
- Add vertex c_j for clause C_j .
- Vertex c_j has edge ***from*** vertex 3_j and ***to*** vertex $3_j + 1$ on path i if x_i appears in clause C_j , and
- Has edge ***from*** vertex $3_j + 1$ and ***to*** vertex 3_j if $\neg x_i$ appears in C_j .



The Reduction

Review II

- Add vertex c_j for clause C_j .
- Vertex c_j has edge ***from*** vertex 3_j and ***to*** vertex $3_j + 1$ on path i if x_i appears in clause C_j , and
- Has edge ***from*** vertex $3_j + 1$ and ***to*** vertex 3_j 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 if and only if part seperately.

Correctness proof

- **Theorem:** φ has a satisfying assignment ***iff*** G_φ has a Hamiltonian cycle.
  - Based on proving if and only if part seperately.
- ***Only if***: If φ has a satisfying assignment then G_φ has a Hamilton cycle.
  - By construction (we just did it)

Correctness proof

- **Theorem:** φ has a satisfying assignment ***iff*** G_φ has a Hamiltonian cycle.
  - Based on proving if and only if part seperately.
- ***Only if***: If φ has a satisfying assignment then G_φ has a Hamilton cycle.
  - By construction (we just did it)
- ***If***: If G_φ has a Hamilton cycle then φ has a satisfying assignment.
  - Far more involved … we will skip (see Kani's archived slides).

Hamiltonian cycle in undirected graphs

Problem

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

Goal: Does G have a Hamiltonian cycle?

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)?

NP-Completeness

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

Proof

NP-Completeness

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

Proof

- The problem is in **NP**; proof left as exercise.

NP-Completeness

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

NP-Completeness

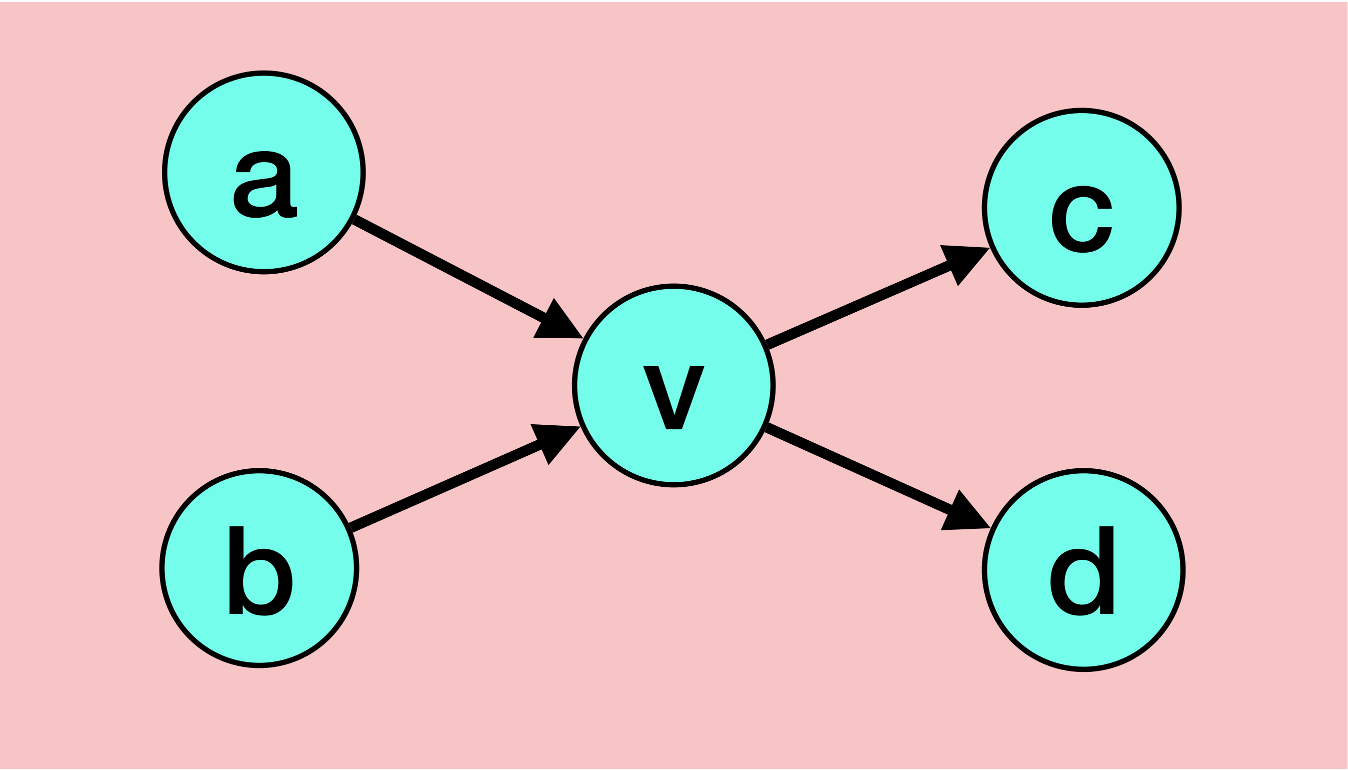
Theorem: *Hamiltonian cycle* problem for undirected graphs is NP-complete.

Proof

- The problem is in **NP**; proof left as exercise.
- Hardness proved by reducing directed Hamiltonian cycle to this problem
  - Need to go from *directed graph to undirected graph*

Reduction sketch

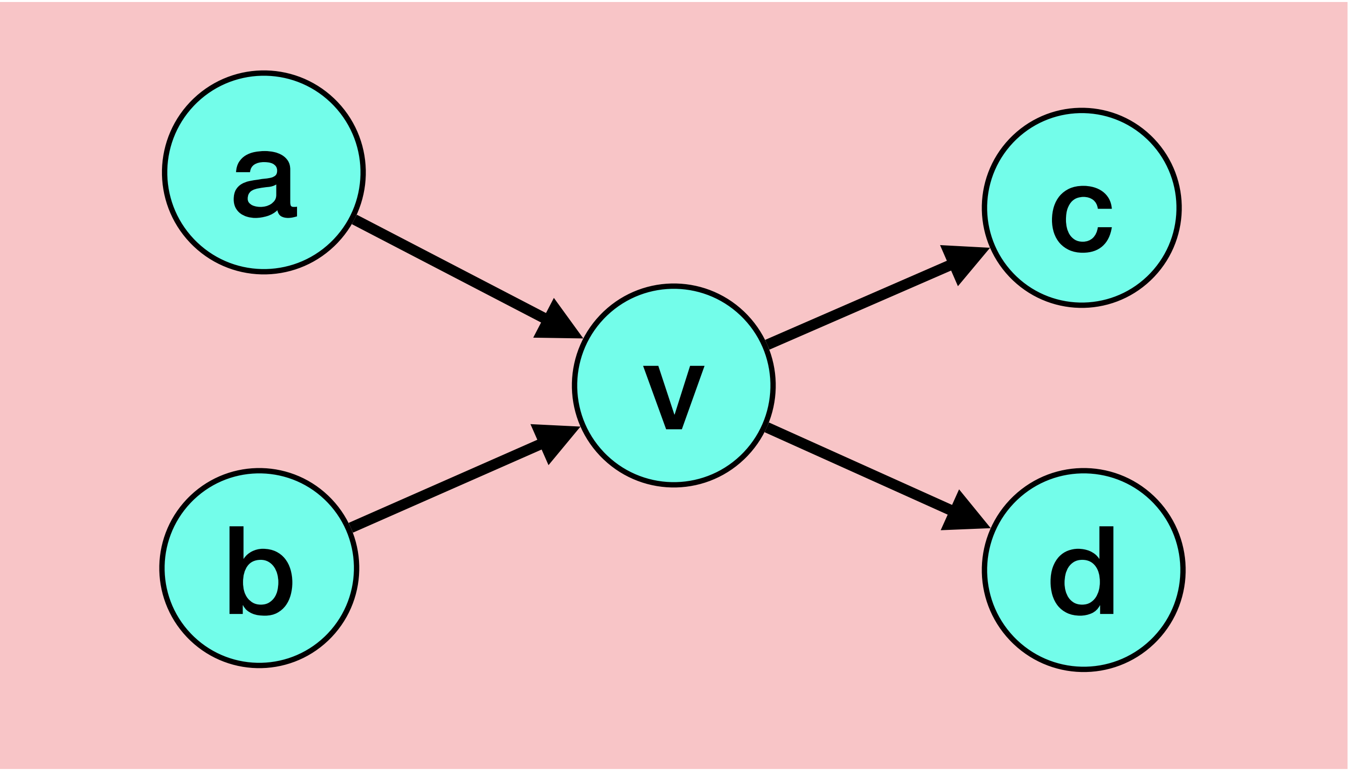
Goal: Given directed graph G, need to construct undirected graph G' such that G has Hamiltonian cycle iff G' has Hamiltonian cycle.

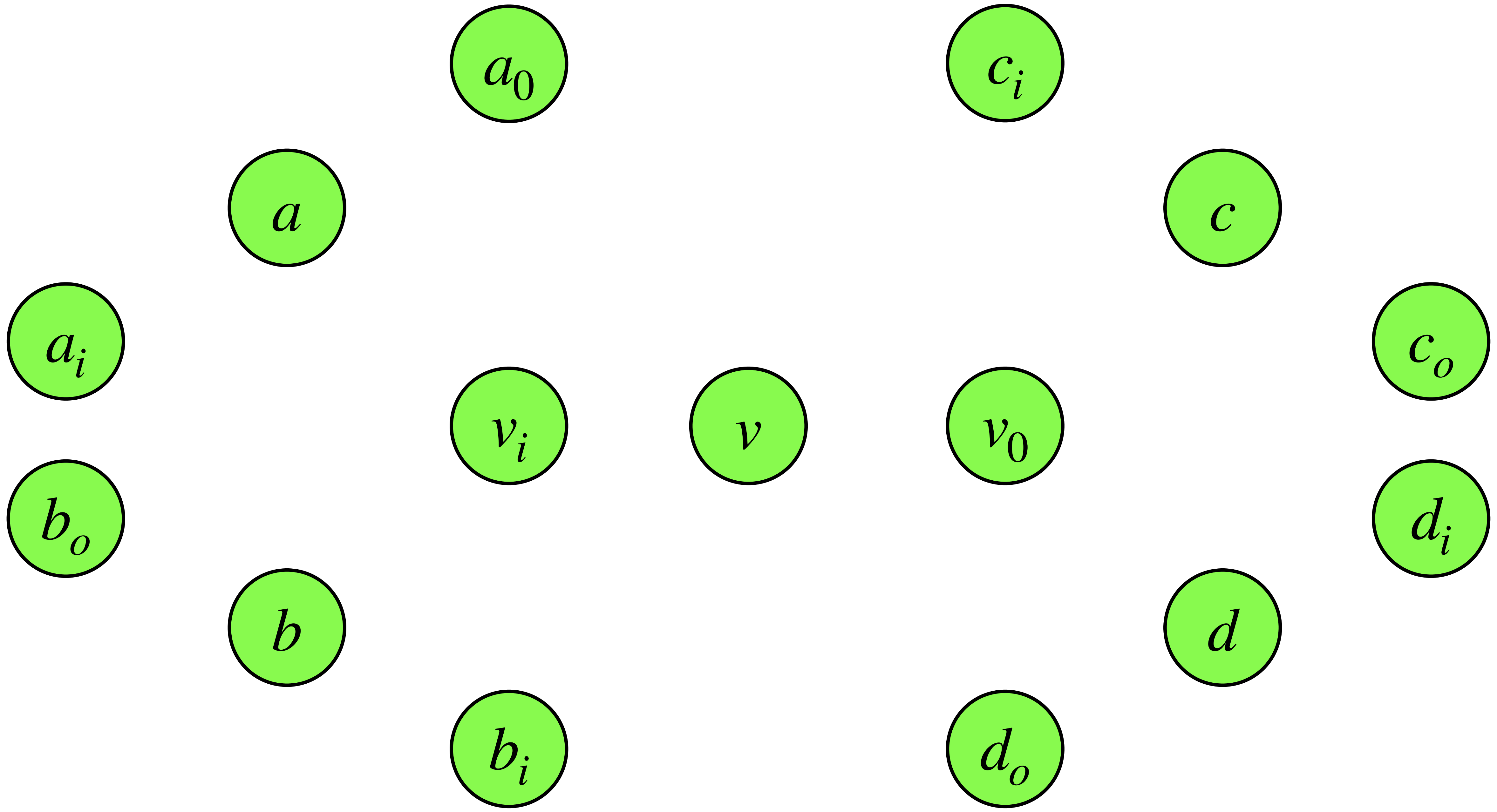


Reduction sketch

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

- Replace each vertex v by 3 vertices: v_i, v, and v_o

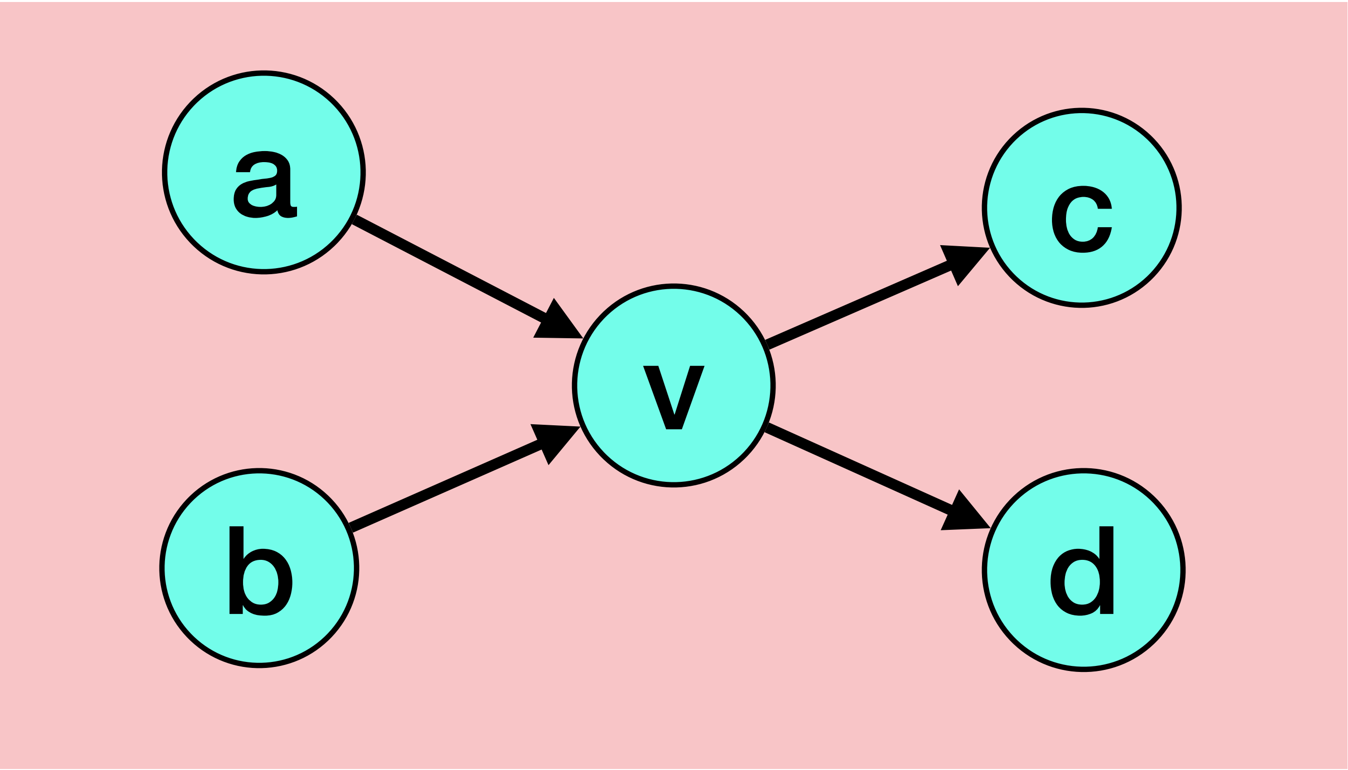


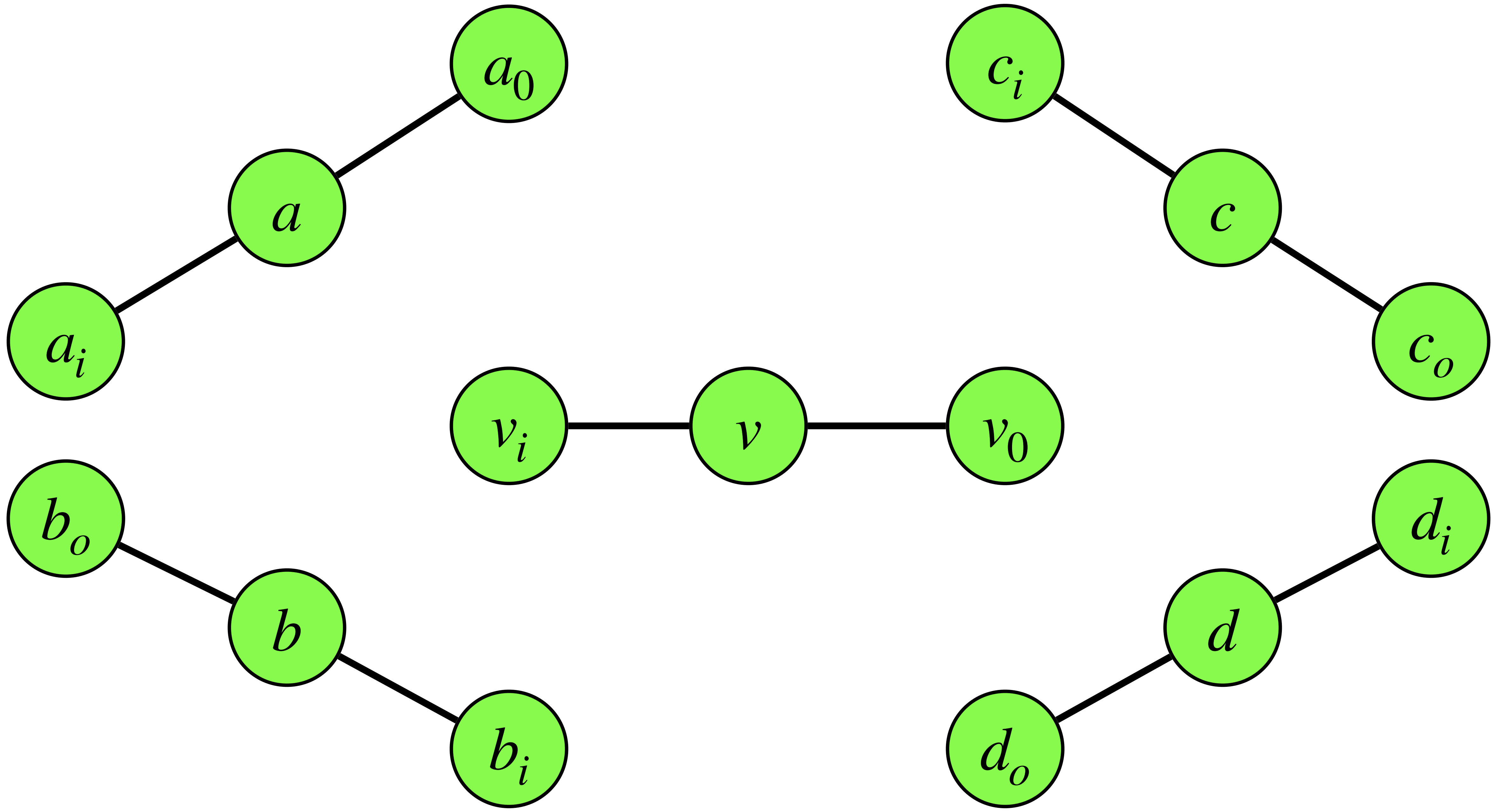


Reduction sketch

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

- Replace each vertex v by 3 vertices: v_i, v, and v_o

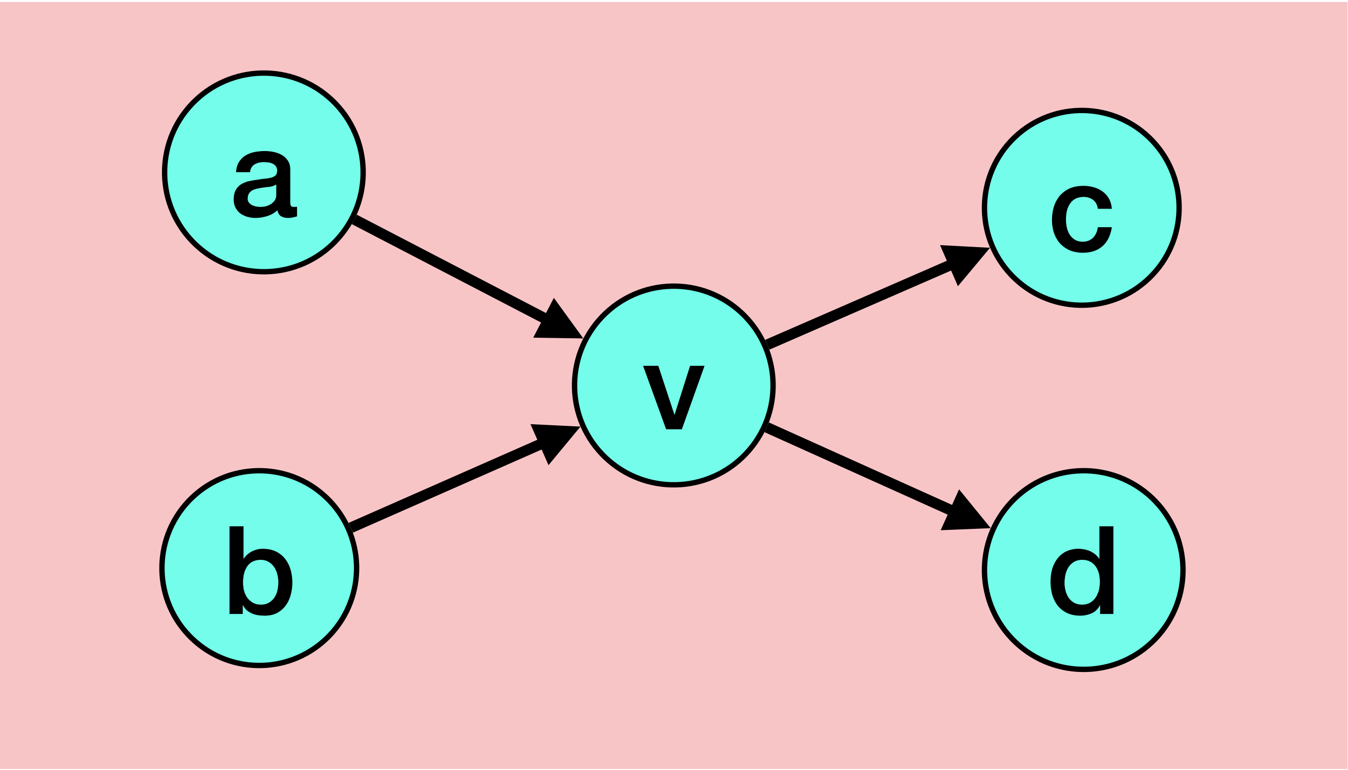


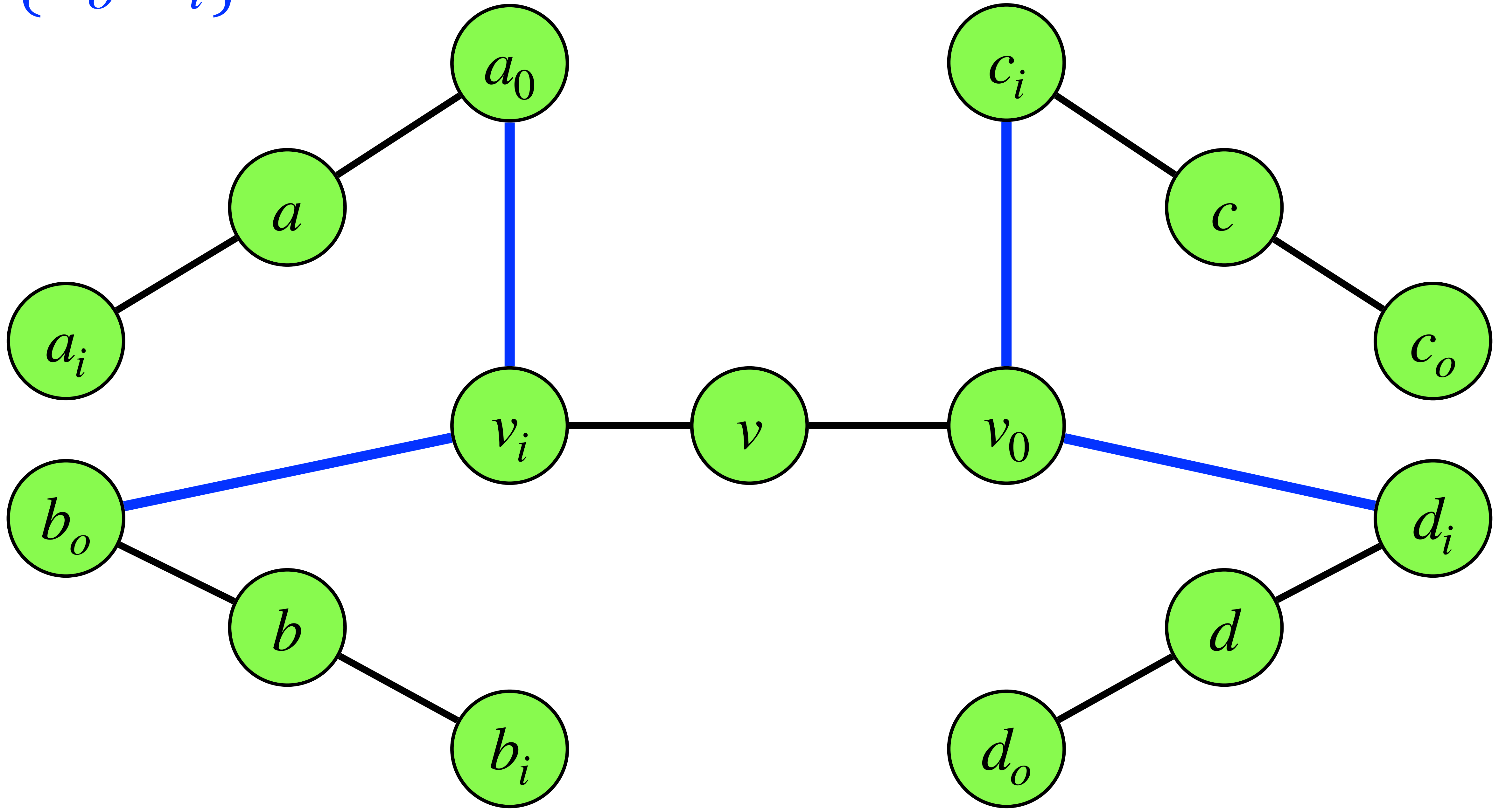


Reduction sketch

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

- Replace each vertex v by 3 vertices: v_i, v, and v_o
- A directed edge (a, b) is replaced by edge $\{a_o, b_i\}$

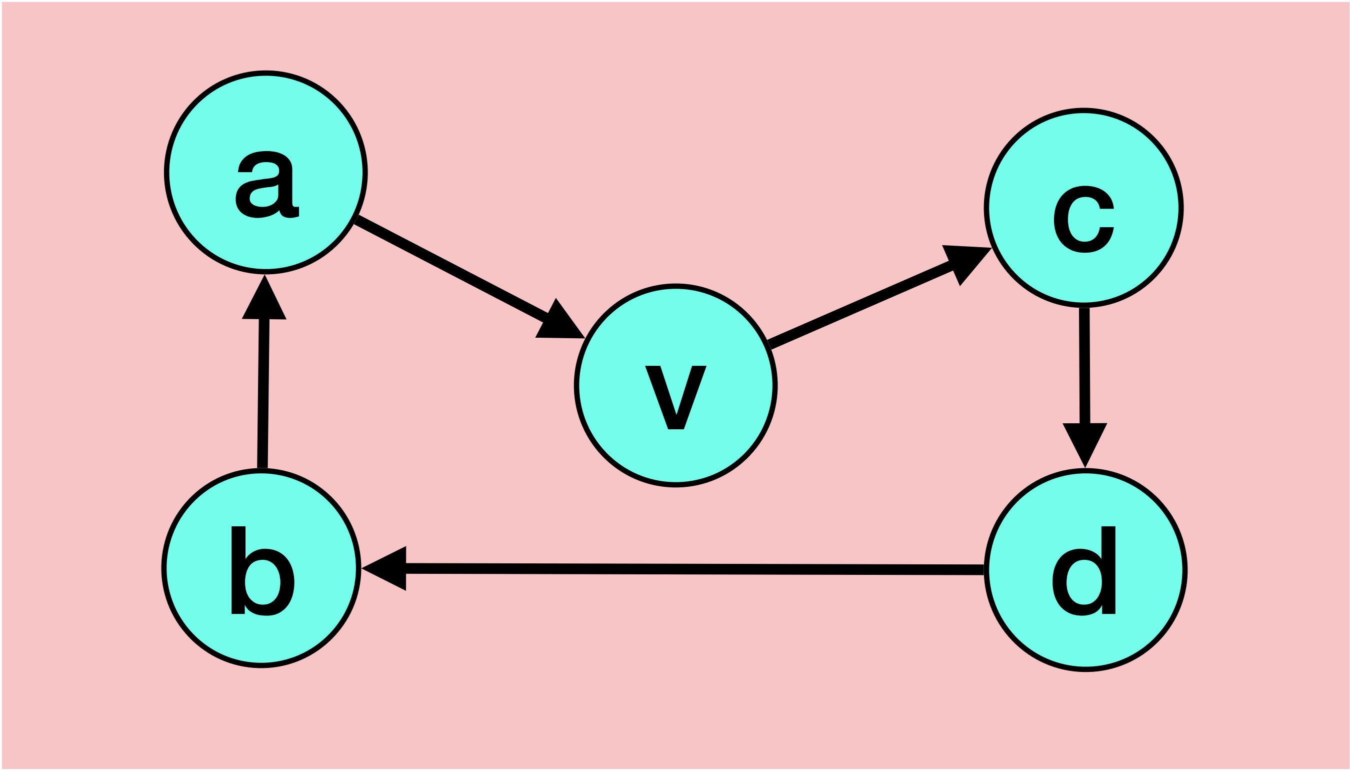


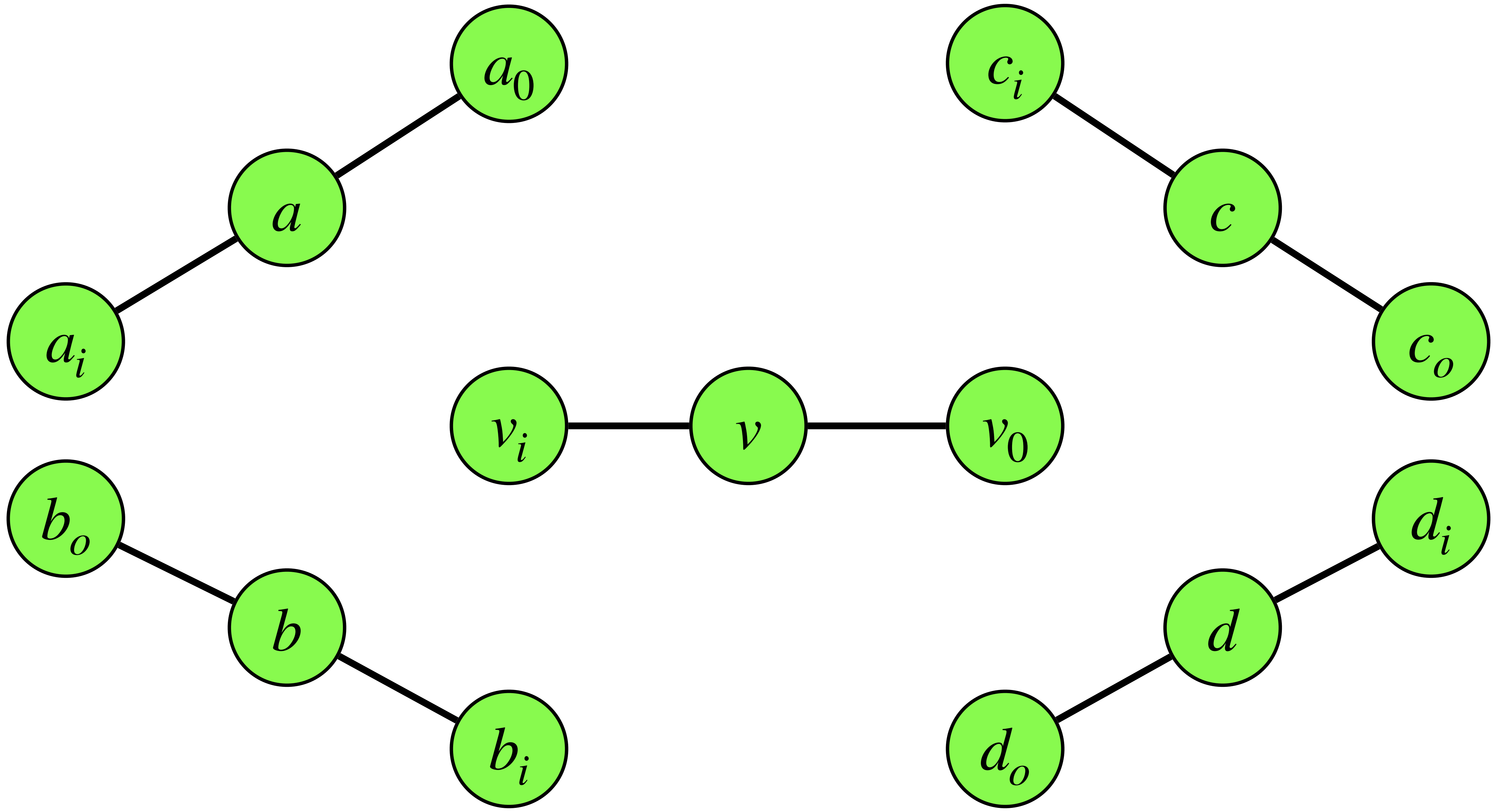


Reduction sketch

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

- Replace each vertex v by 3 vertices: v_i, v, and v_o

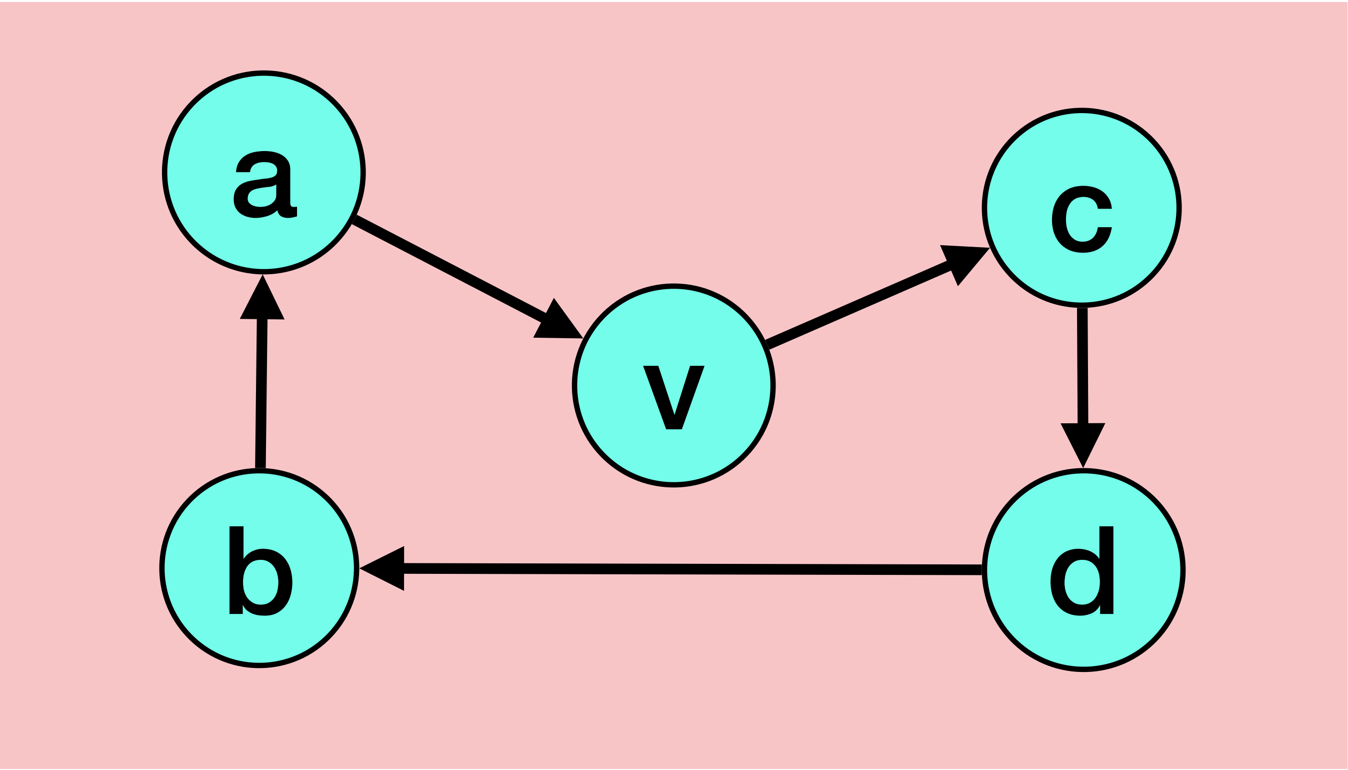


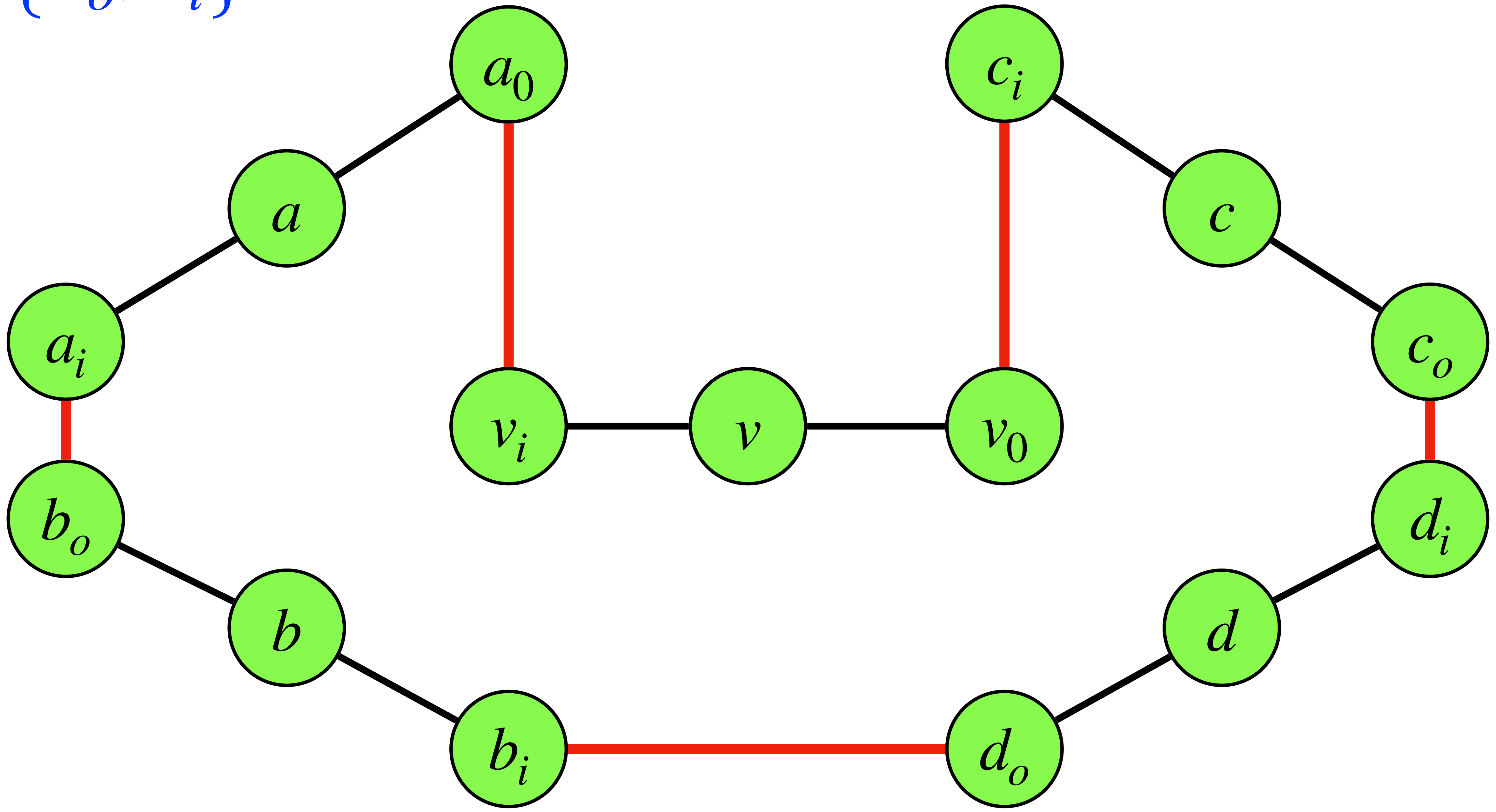


Reduction sketch

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

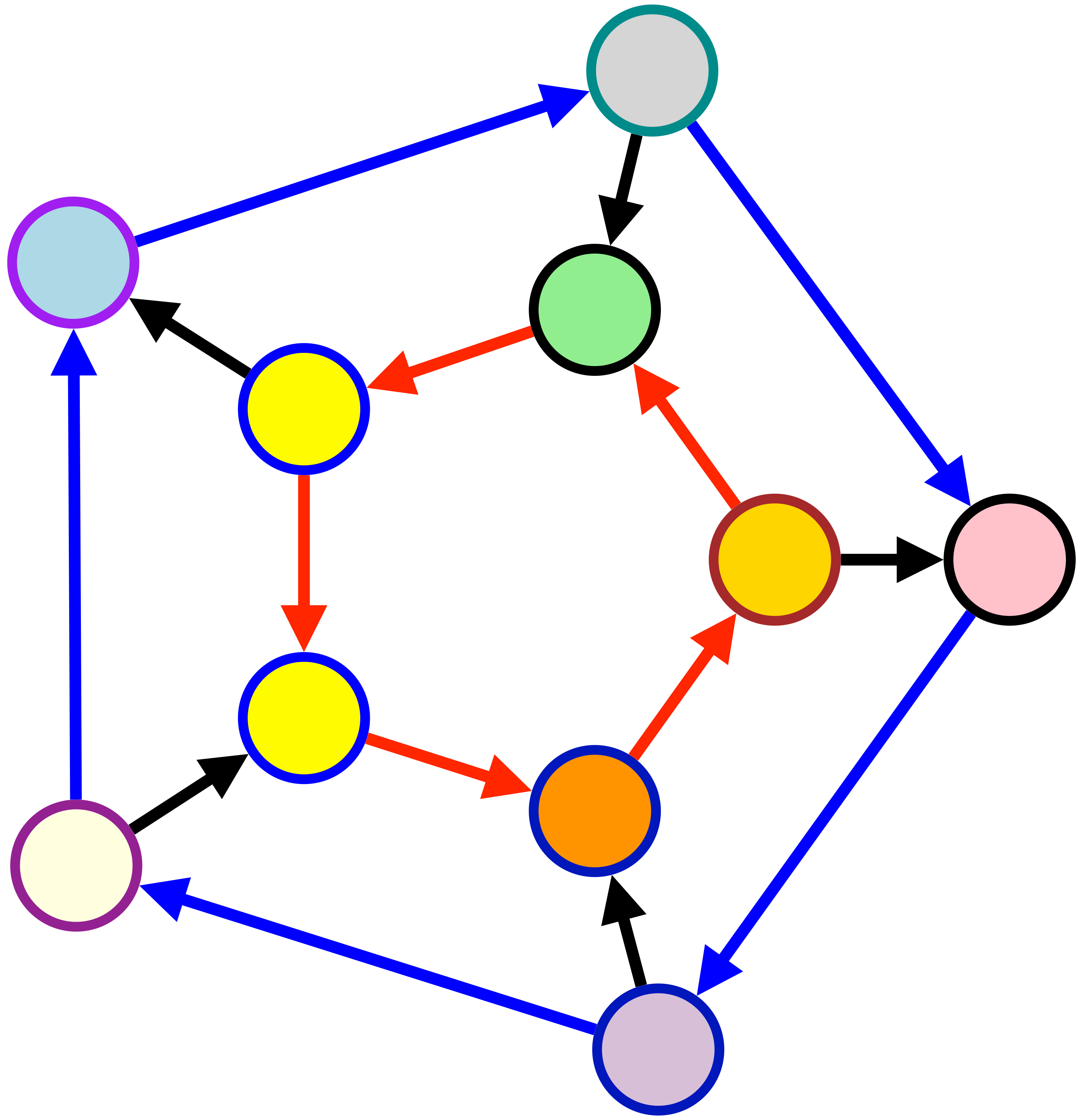
- Replace each vertex v by 3 vertices: v_i, v, and v_o
- A directed edge (a, b) is replaced by edge $\{a_o, b_i\}$





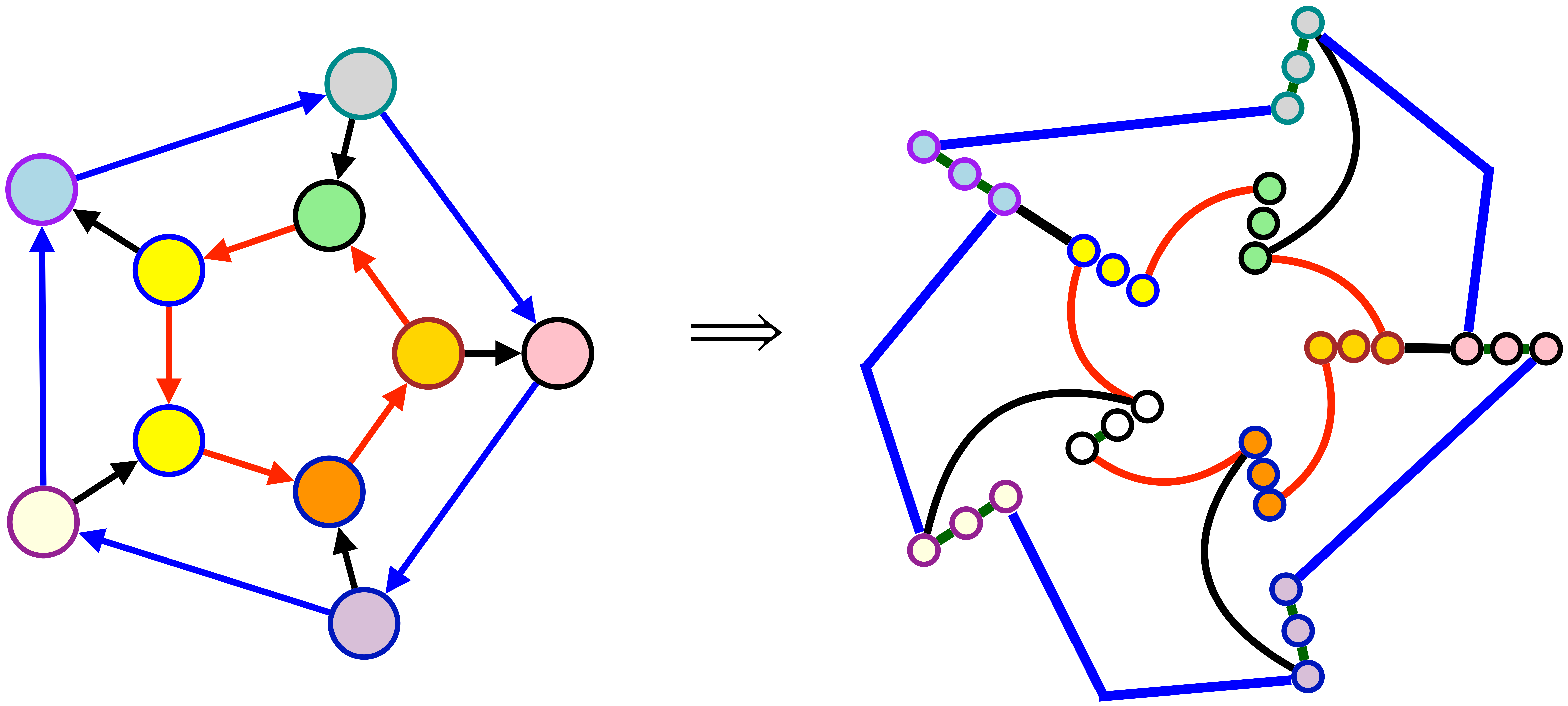
Hamiltonian cycle reduction

Directed to undirected



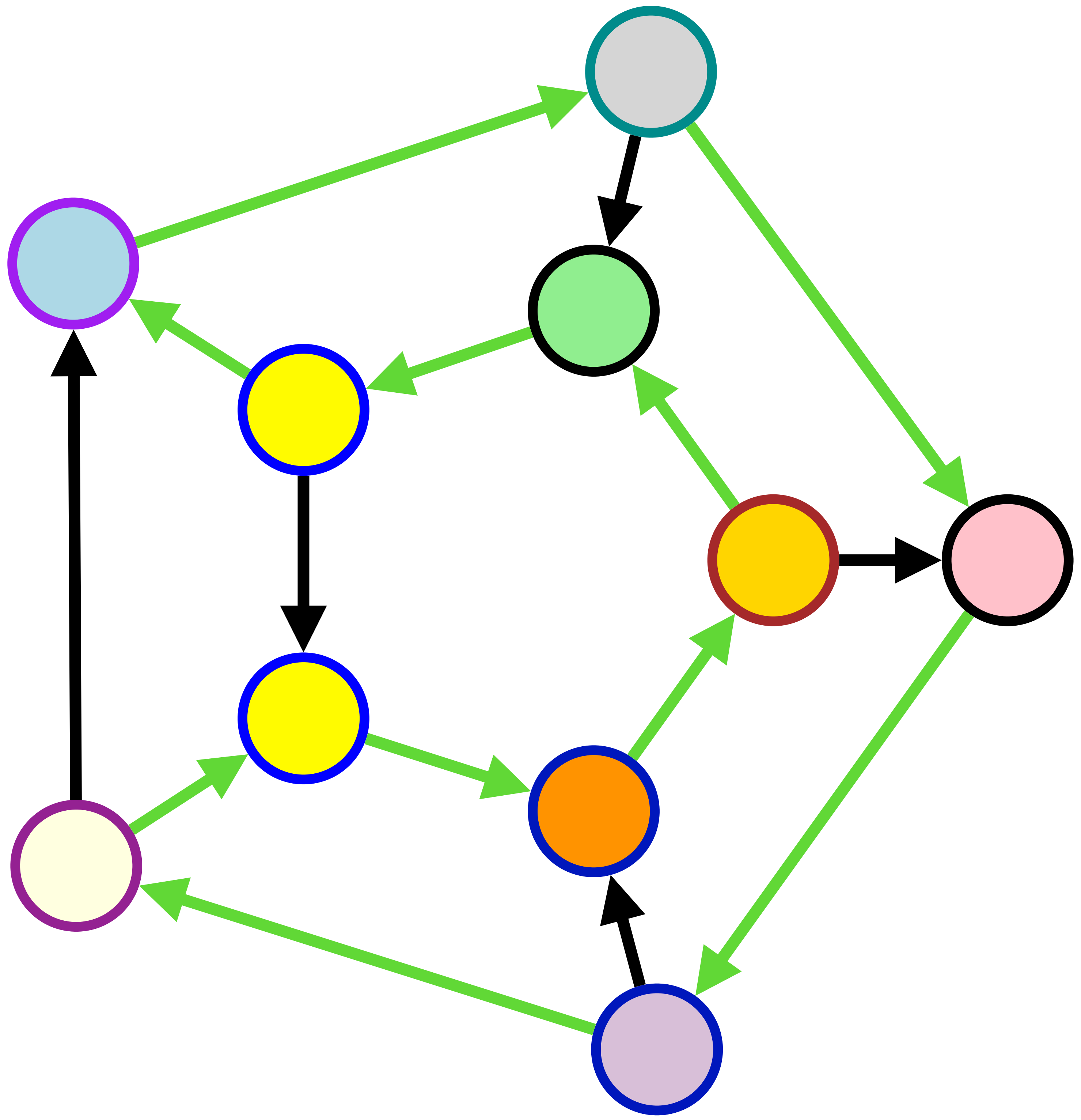
Hamiltonian cycle reduction

Directed to undirected

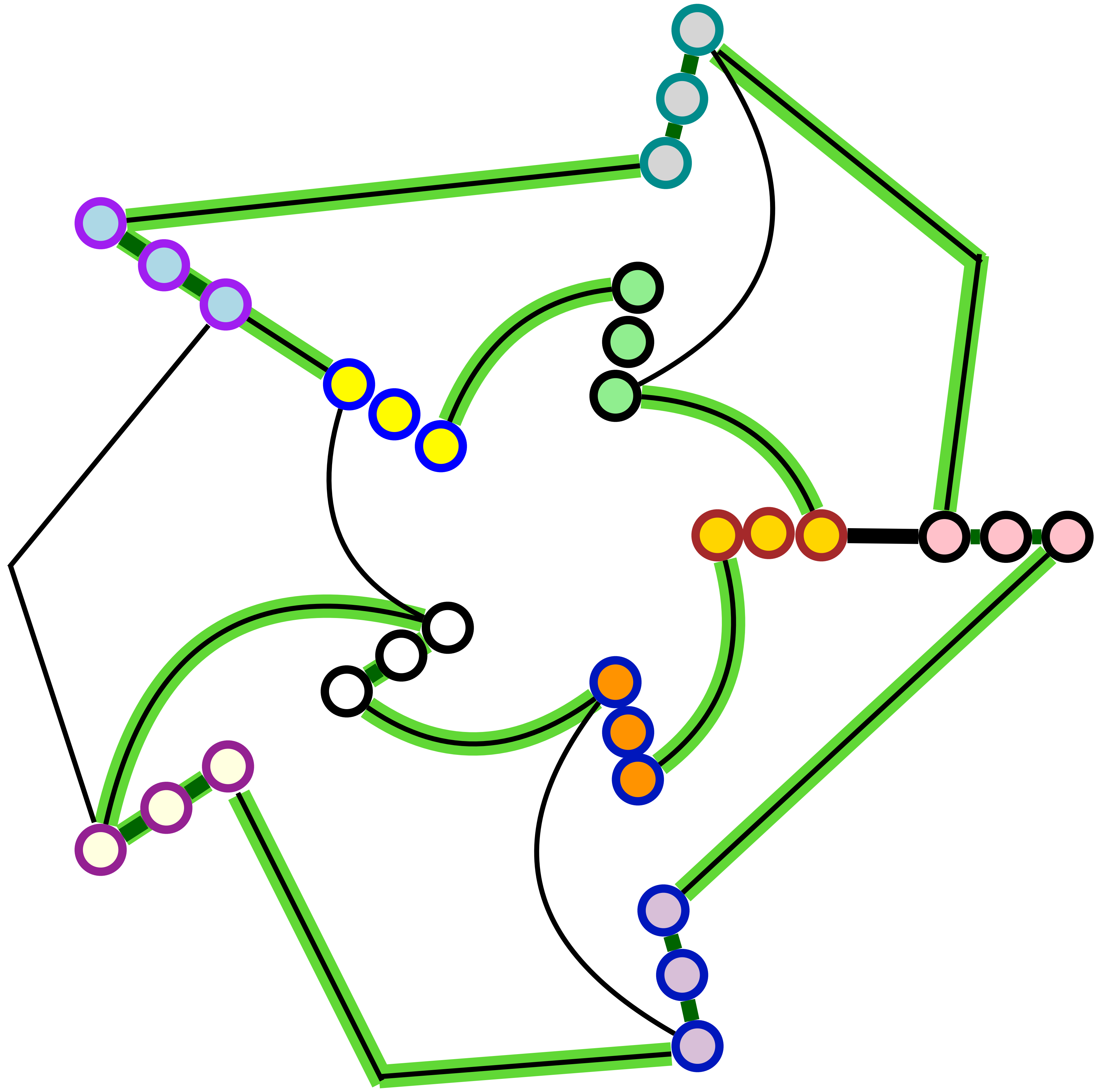


Hamiltonian cycle reduction

Directed to undirected



$\Longrightarrow$



Hamiltonian paths

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

Goal: Does G have a Hamiltonian path?

Hamiltonian paths

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 paths

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.

Hamiltonian paths

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.

Modify the reduction from *3-SAT* to *Hamiltonian Cycle* or do a reduction from *Hamiltonian Cycle* (homework?)

NP-completeness of graph coloring

Generic graph coloring

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

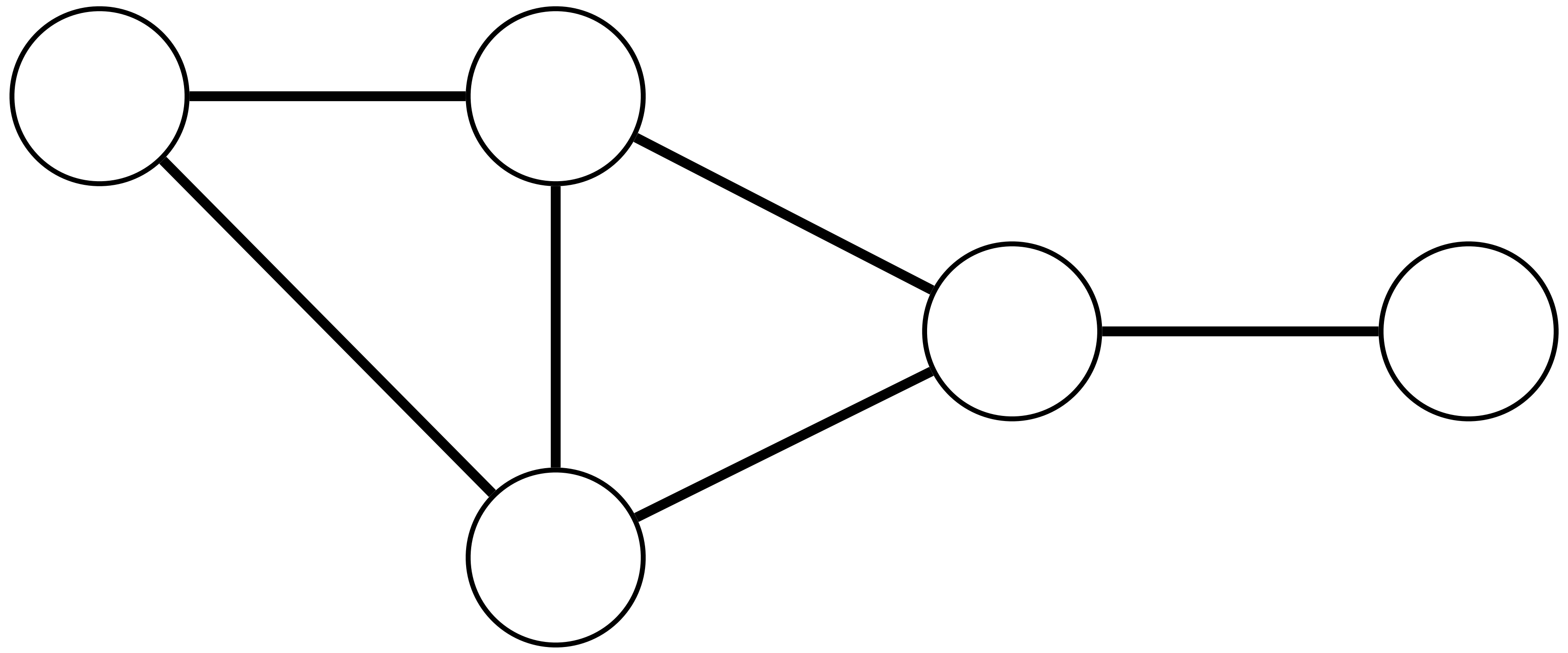
Question: Can the vertices of the graph be colored using k colors so that vertices connected by an edge **do not** get the same color?

NP-completeness of graph coloring

Graph 3-Coloring

Instance: $G = (V, E)$: Undirected graph, integer $k = 3$.

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?

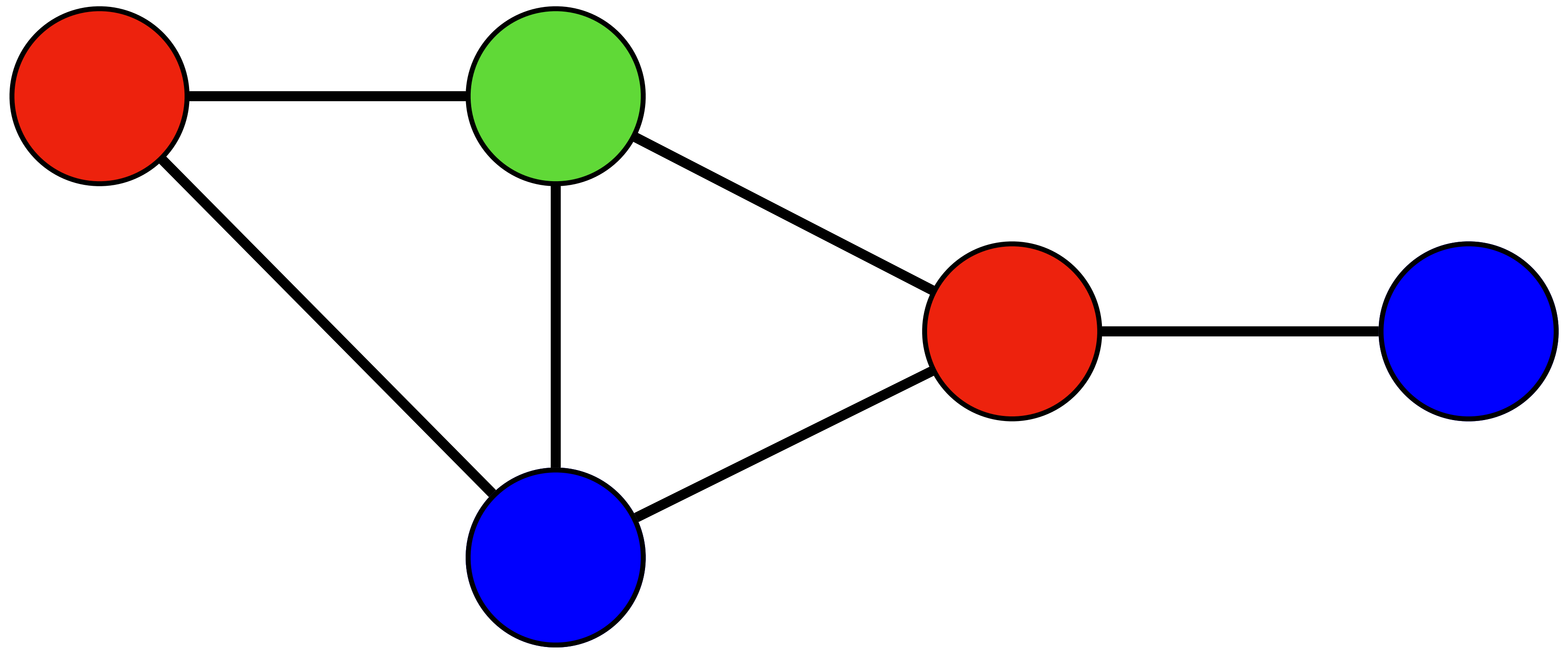


NP-completeness of graph coloring

Graph 3-Coloring

Instance: $G = (V, E)$: Undirected graph, integer $k = 3$.

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

Graph 2-Coloring

$G=(\mathcal{U},\mathcal{V},E)$ s.t $\mathcal{U}\cap\mathcal{V}=\emptyset$

and $e=(u,v)\in E$ s.t

$u\in\mathcal{U}$, $v\in\mathcal{V}$

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 ***if and only if*** 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.

[Chatin] https://dl.acm.org/doi/10.1145/989393.989403

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 $k \geq 3$

Problems related to graph coloring

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 FCC given 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

Problems related to graph coloring

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 FCC given frequency range $[a, b]$ into disjoint <u>bands</u> 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 get same band, otherwise signals will interfere

Problems related to graph coloring

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 FCC given frequency range $[a, b]$ into disjoint <u>bands</u> 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 get same band, otherwise signals will interfere

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

Problems related to graph coloring

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 FCC given 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 get same band, otherwise signals will interfere

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-SAT $\leq_P$ 3-Coloring.

Reduction Idea I - Simple 3-color gadget

We want to create a ***gadget***[1] that:

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

1: https://en.wikipedia.org/wiki/Gadget_(computer_science)

Reduction Idea I - Simple 3-color gadget

We want to create a ***gadget*[1]** 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:

1: https://en.wikipedia.org/wiki/Gadget_(computer_science)

Reduction Idea I - Simple 3-color gadget

We want to create a ***gadget***[1] 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:

$$f(X_1, X_2) = (X_1 \vee X_2)$$

1: https://en.wikipedia.org/wiki/Gadget_(computer_science)

Reduction Idea I - Simple 3-color gadget

We want to create a ***gadget***[1] 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:

$$f(X_1, X_2) = (X_1 \vee X_2)$$

Assume green=true and red=false, essentially need to create an OR-gate with graph coloring.

1: https://en.wikipedia.org/wiki/Gadget_(computer_science)

Reduction Idea I - Simple 3-color gadget

$$f(X_1, X_2) = (X_1 \vee X_2)$$

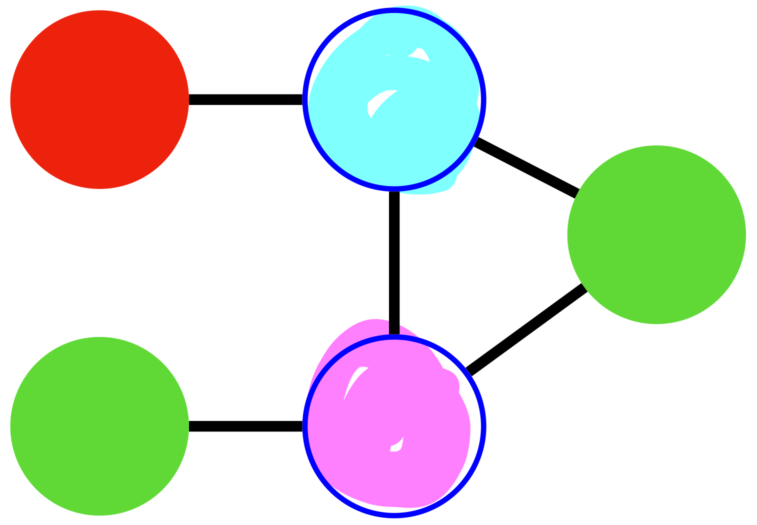
- Is 3-colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

Fix output to be green

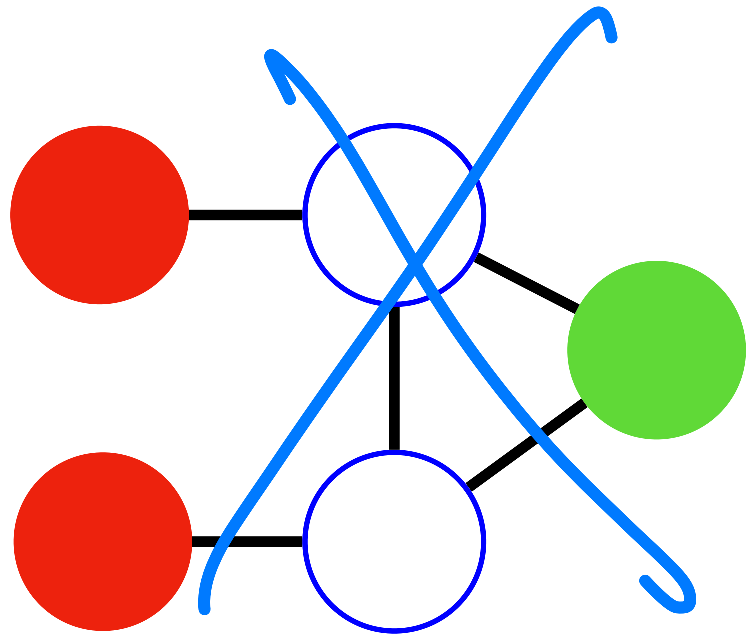
Reduction Idea I - Simple 3-color gadget

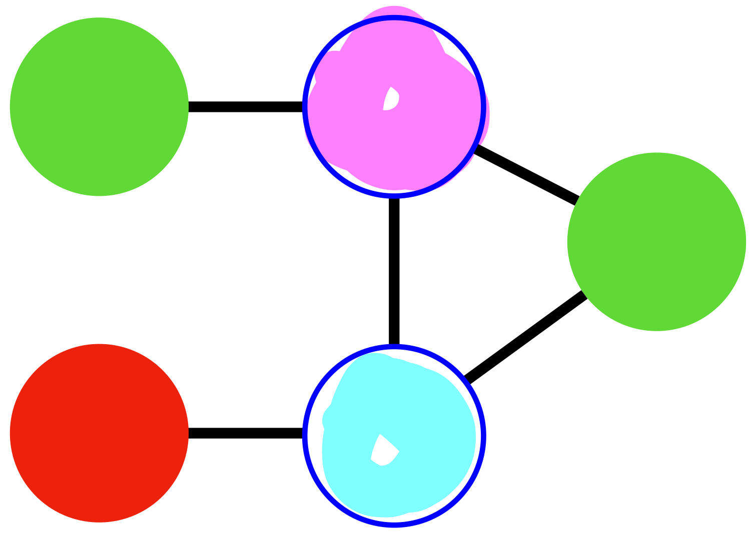
3rd color is blue.

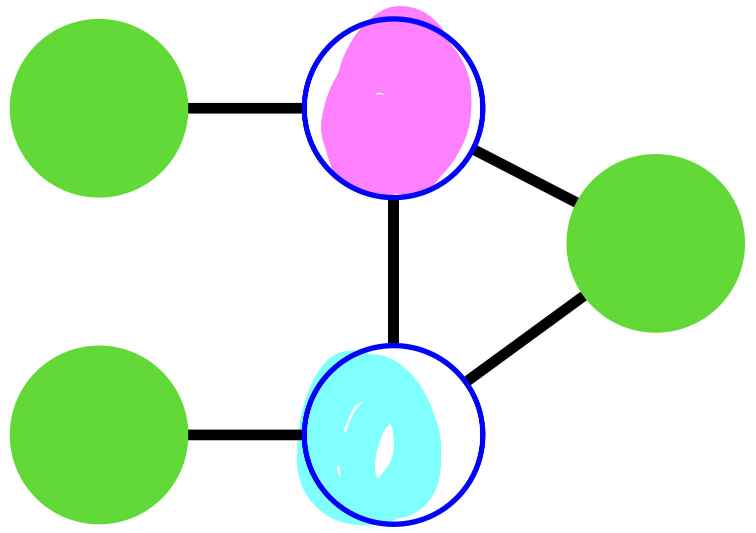
$$f(X_1, X_2) = (X_1 \vee X_2)$$



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







Fix output to be green

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

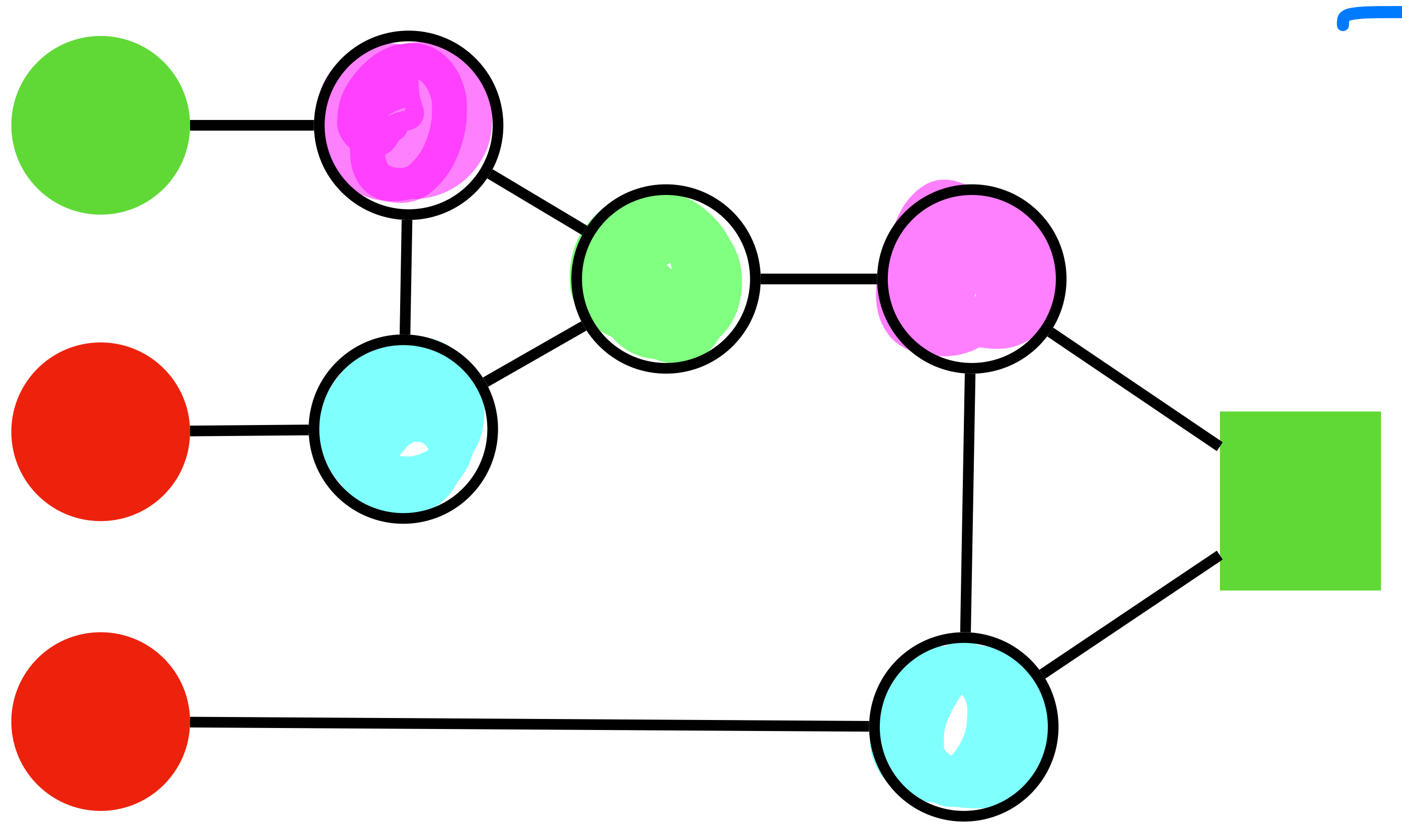
How do we do the same thing for 3 variables?:

$$f(X_1, X_2, X_3) = (X_1 \vee X_2 \vee X_3)$$

Assume green=true and red=false.

3-color this gadget I

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).



acts as an OR.

A. Yes

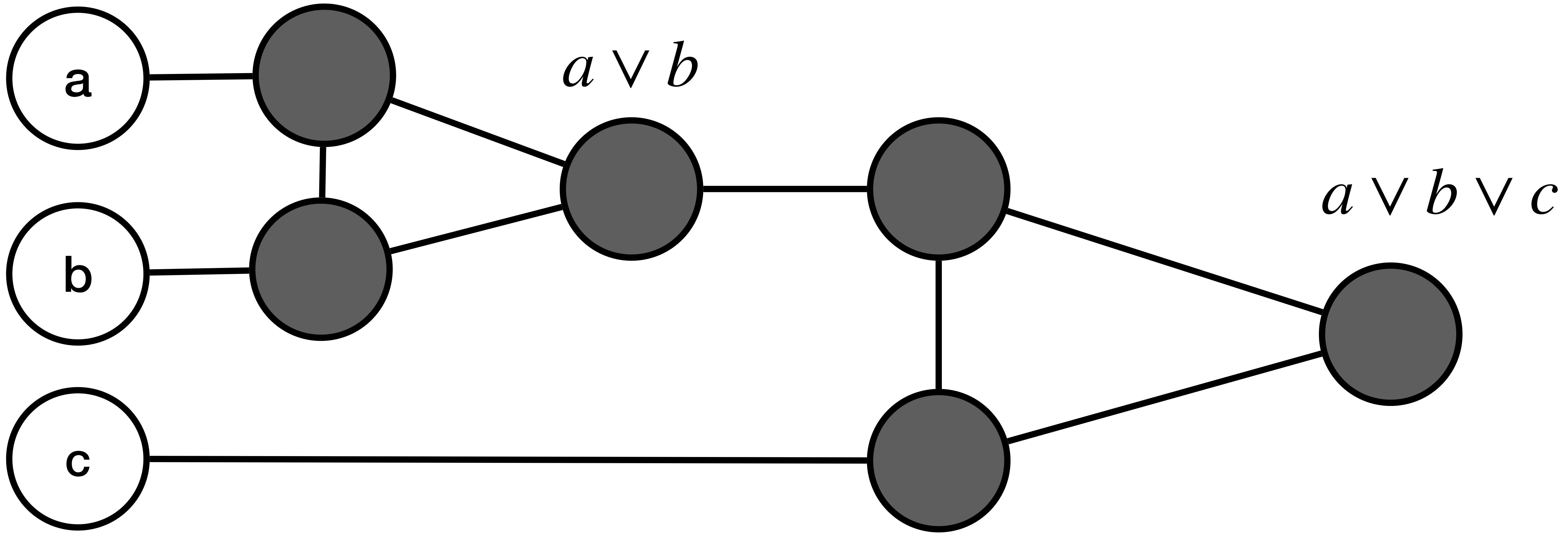
B. No

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

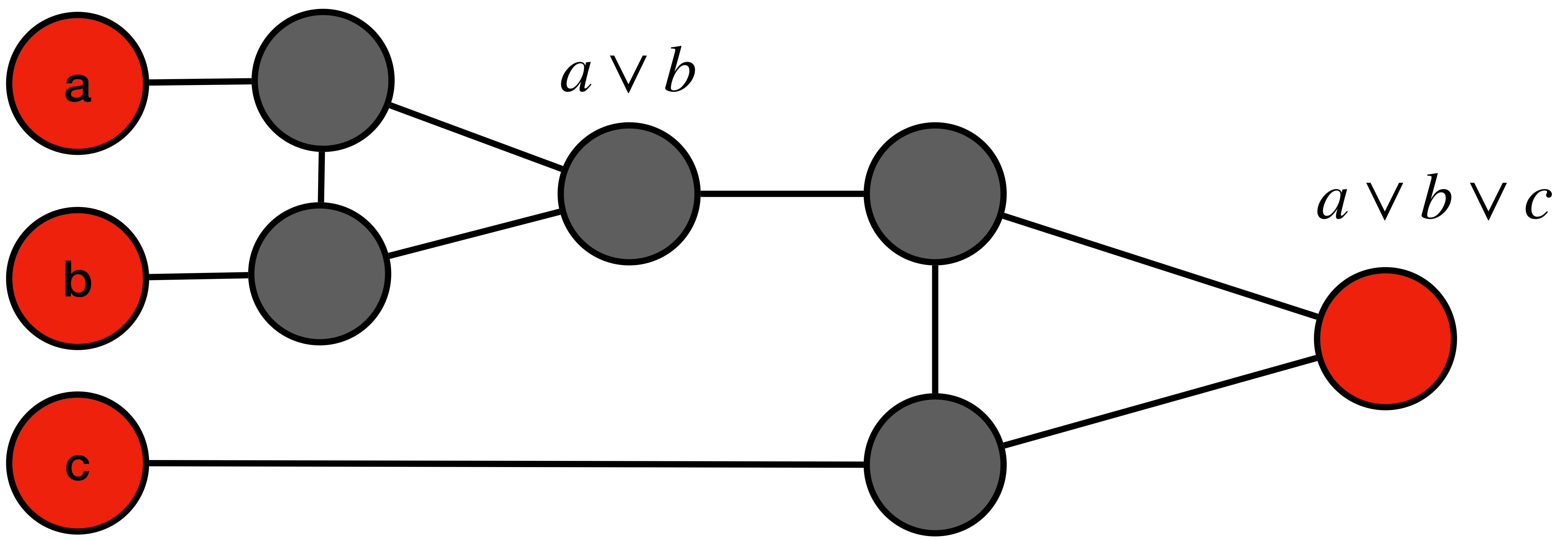


Clause Satisfiability gadget

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

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



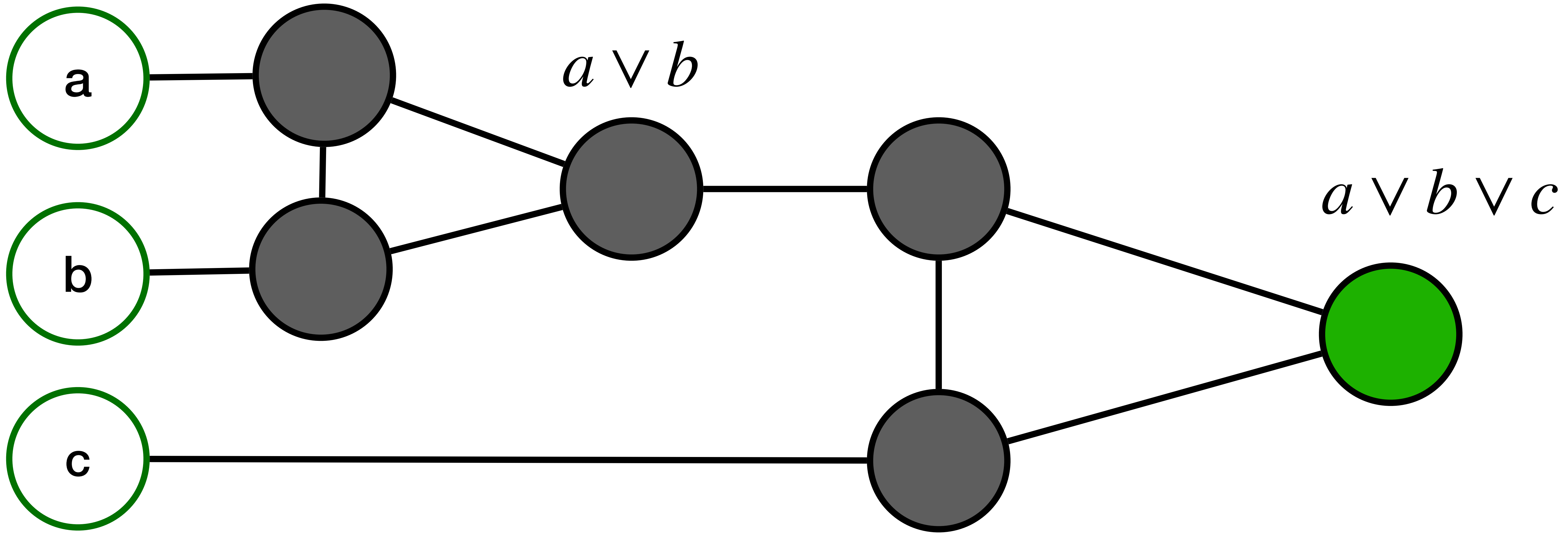
Clause Satisfiability gadget

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

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

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.

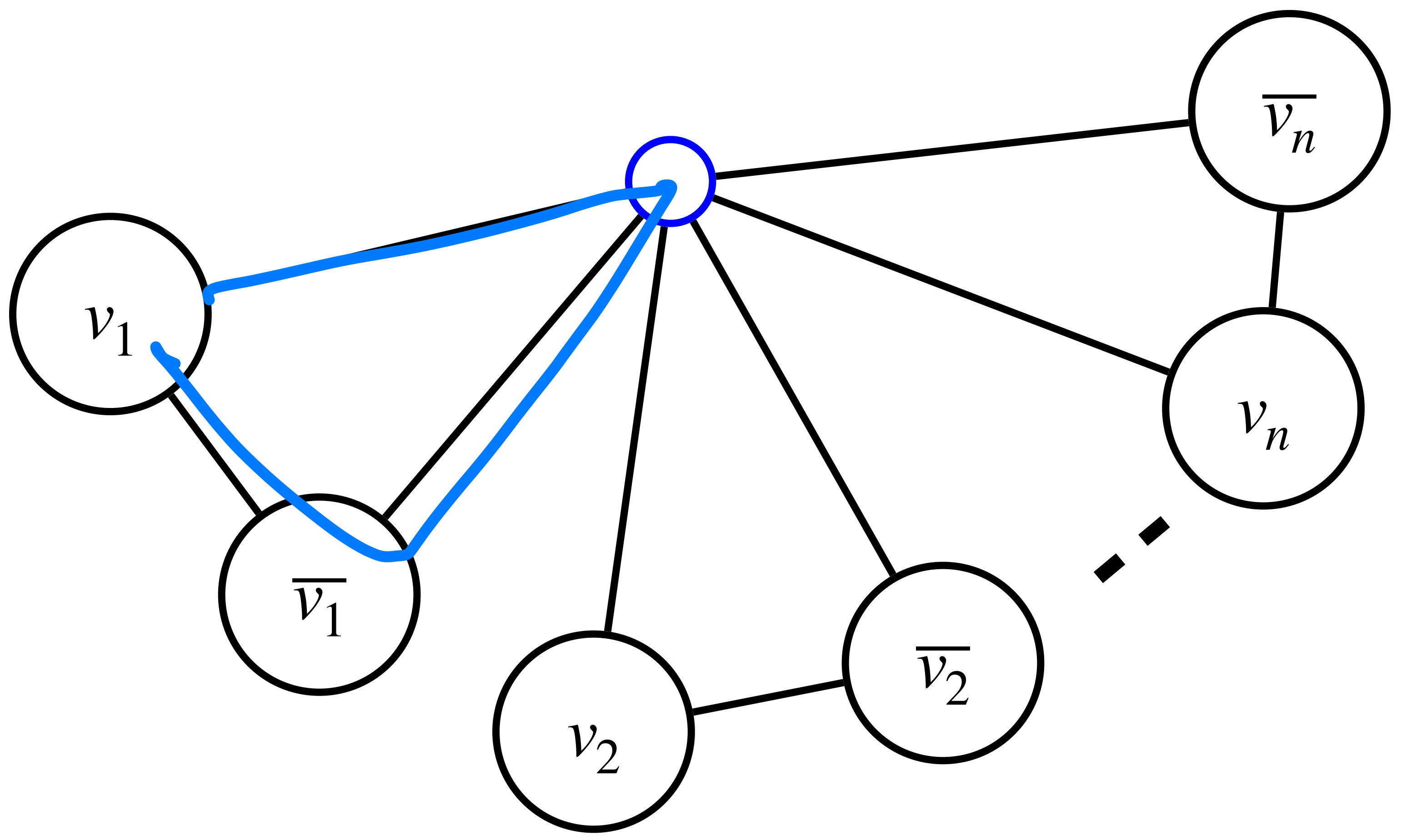


Reduction Idea II

Literal assignment I

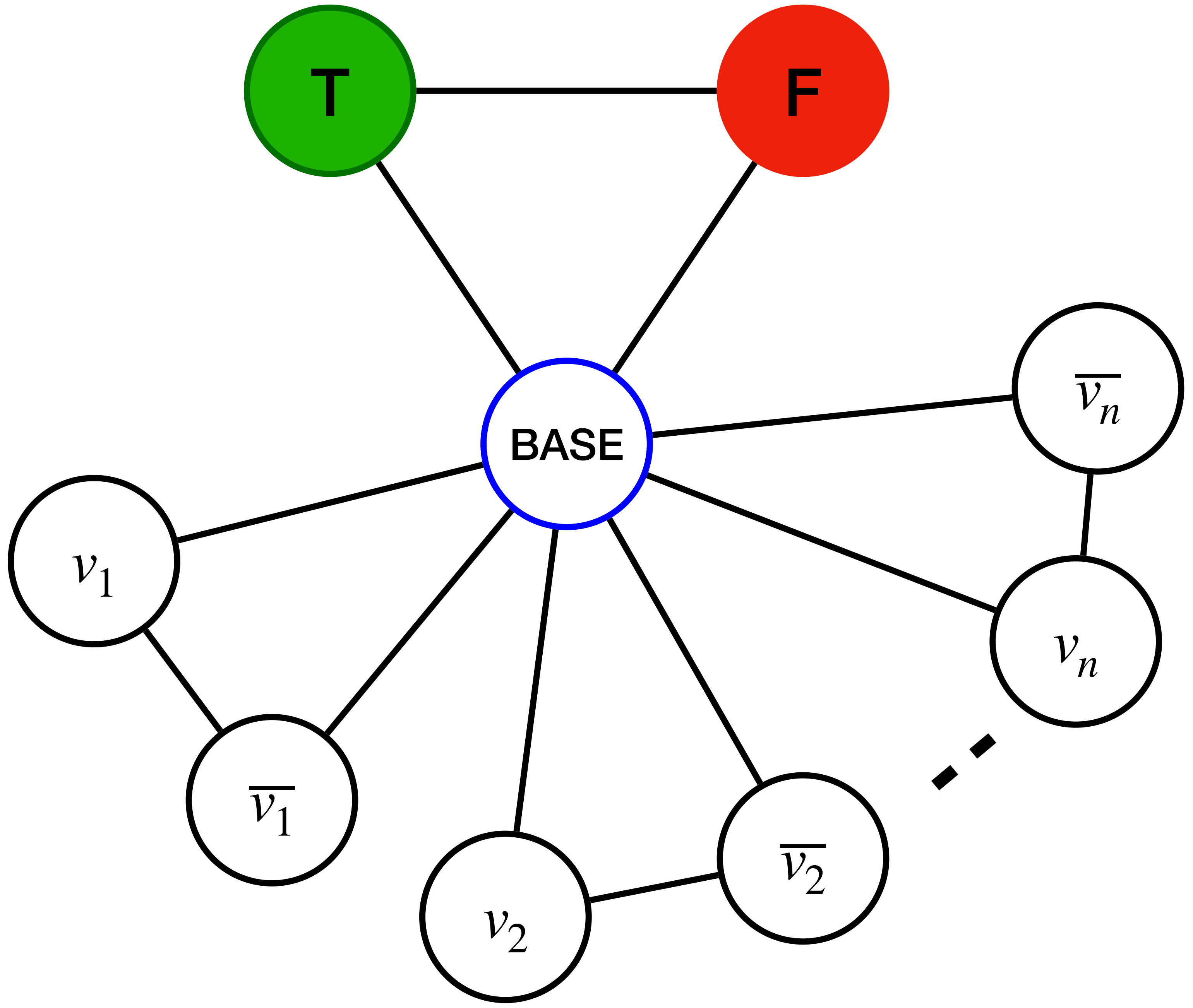
Next we need a gadget that assigns literals. Our previously constructed gadget assumes:

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



Reduction idea

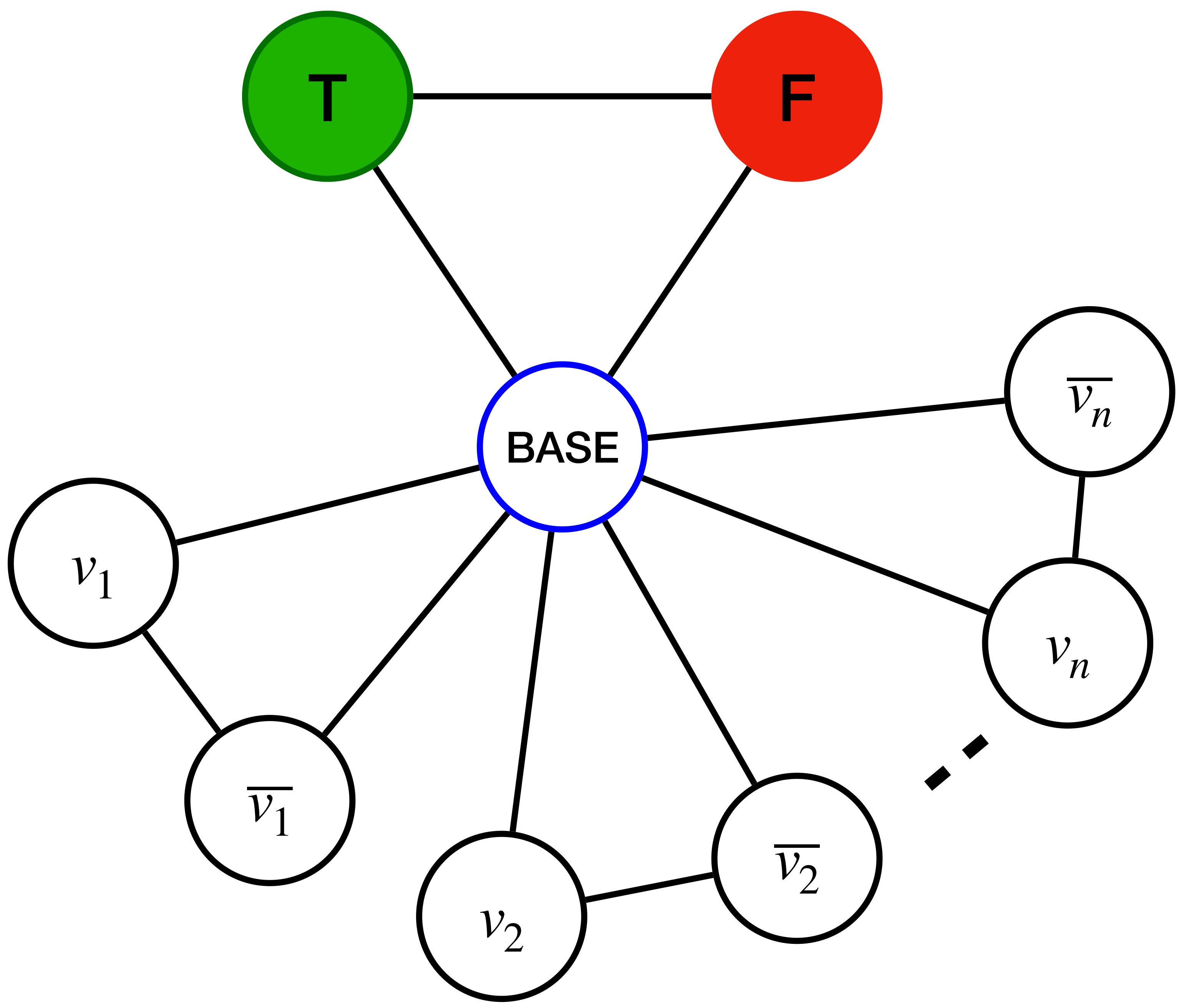
Start with **3SAT** formula (i.e., 3CNF formula) φ with n variables $X_1, \ldots, X_n$ and $m$clauses $C_1, \ldots, C_m$. Create graph G_φ such that G_φ is 3-colorable iff φ is satisfiable



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

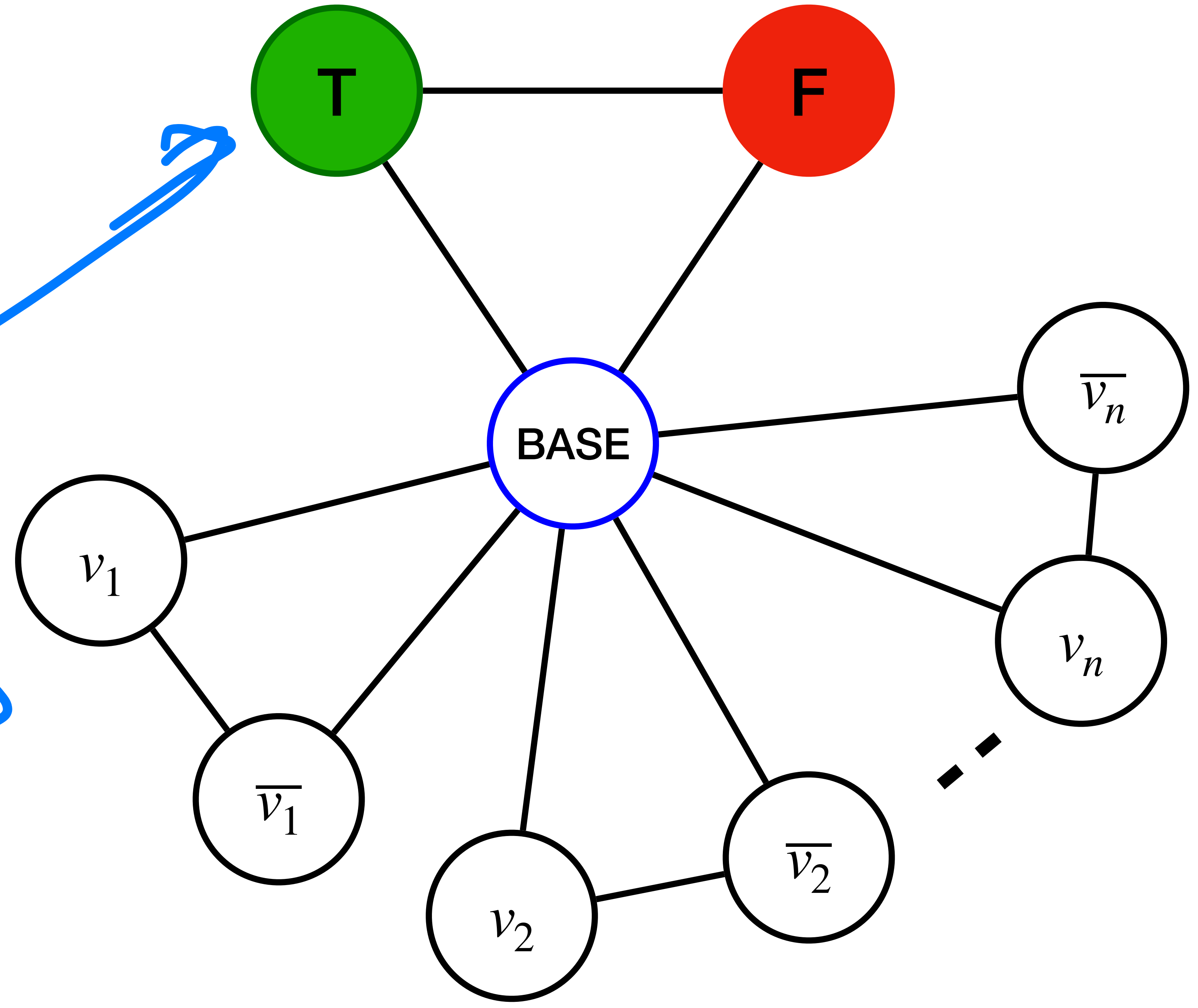
- need to establish truth assignment for $X_1, \ldots, X_n$ via colors for some nodes in G_φ



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

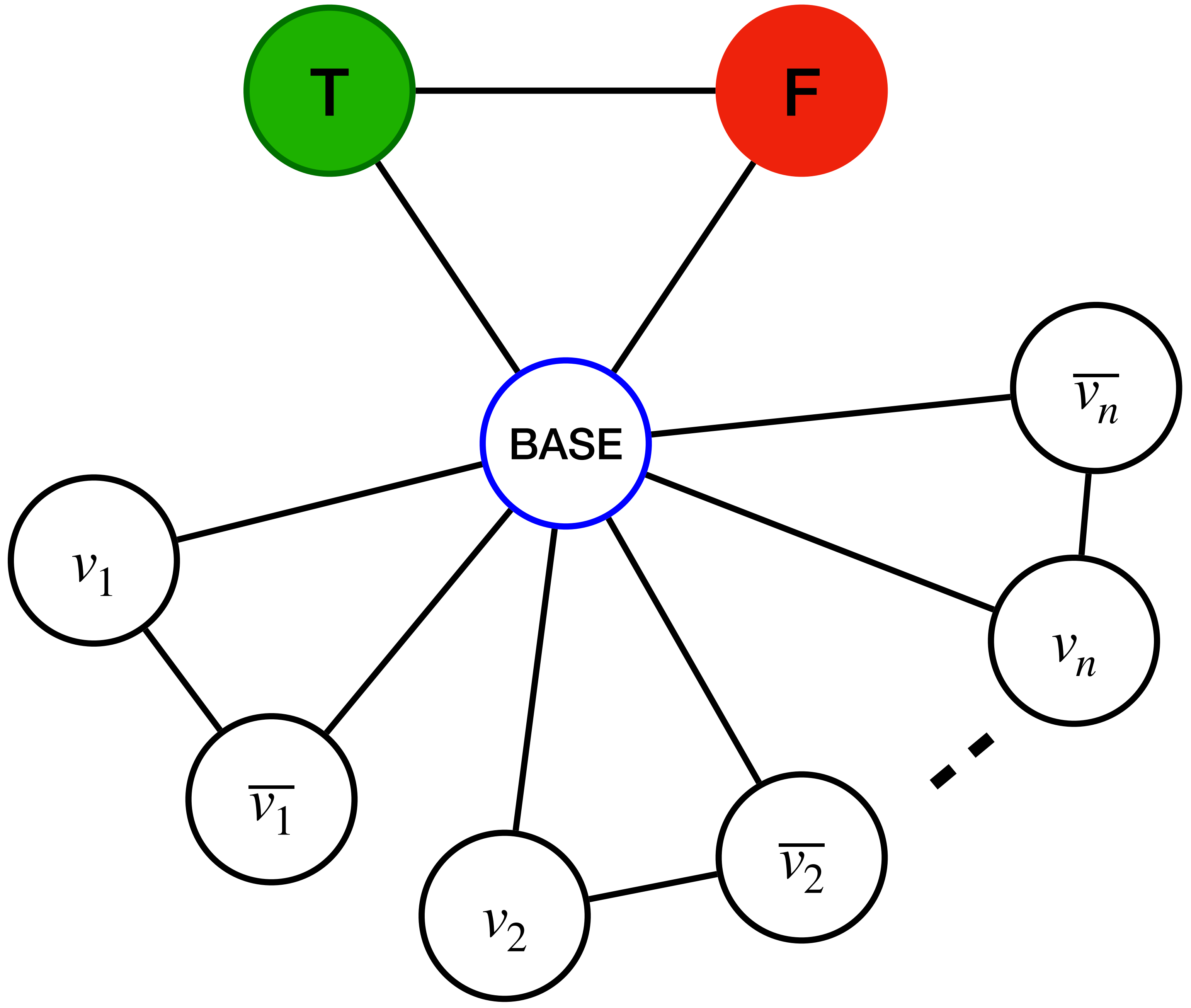
- need to establish truth assignment for $X_1, \ldots, X_n$ via colors for some nodes in G_φ
- create triangle with nodes: True, False, Base
- for each variable X_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base



Reduction idea

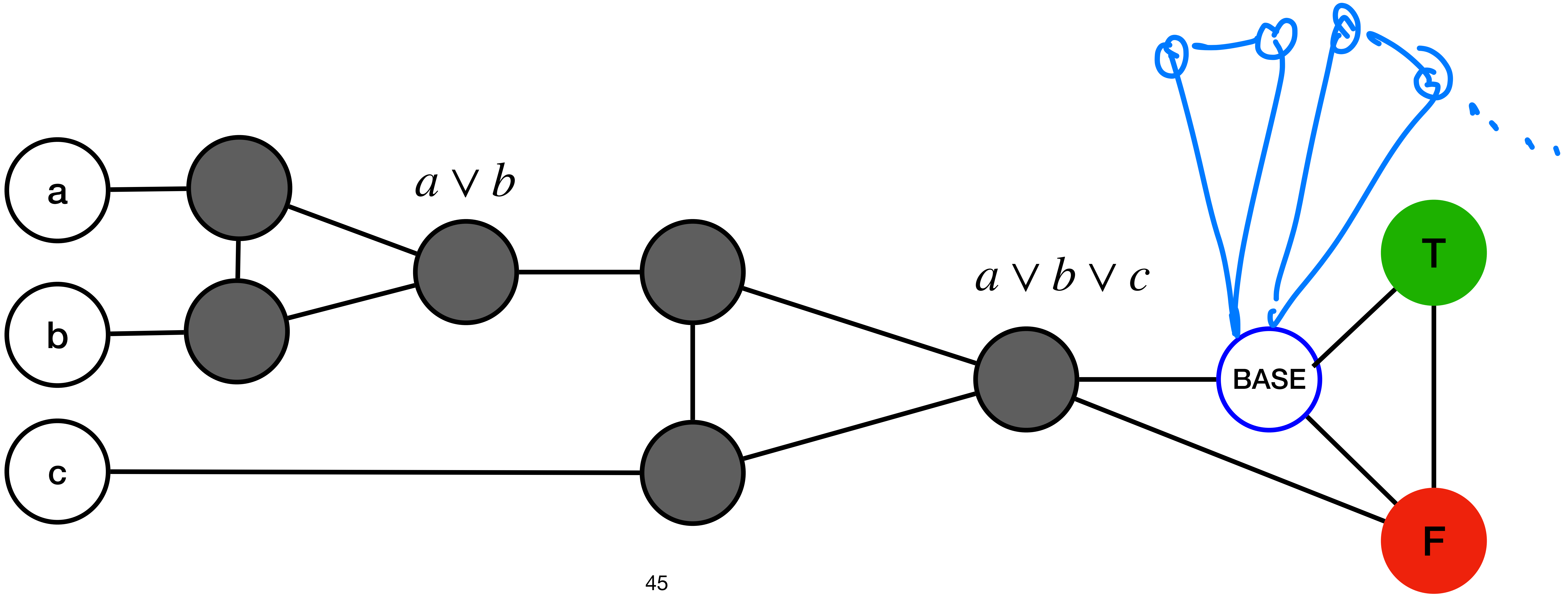
Start with **3SAT** formula (i.e., 3CNF formula) φ with n variables $X_1, \ldots, X_n$ and $m$clauses $C_1, \ldots, C_m$. Create graph G_φ such that G_φ is 3-colorable iff φ is satisfiable

- need to establish truth assignment for $X_1, \ldots, X_n$ via colors for some nodes in G_φ
- create triangle with nodes: 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



Reduction

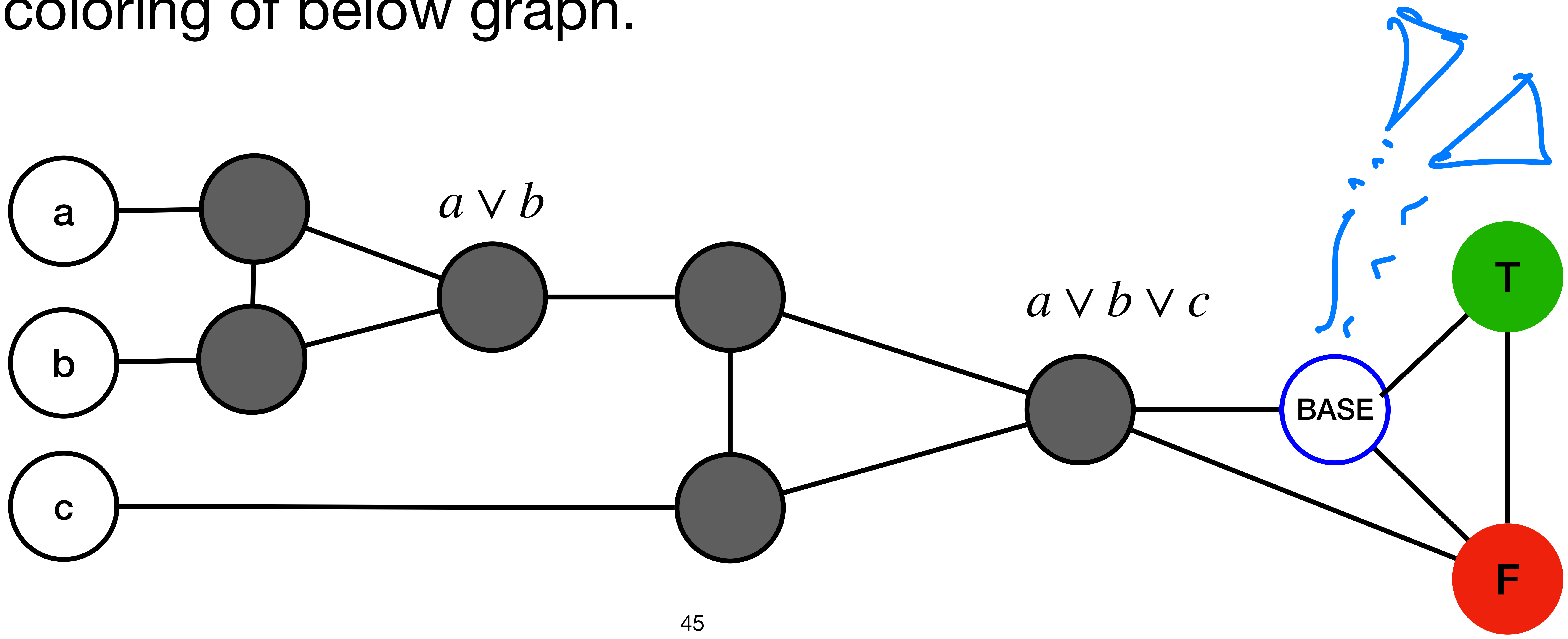
- For each clause $C_j = (a \vee b \vee c)$, add OR-gadget graph with input nodes a, b, c and connect output node of gadget to both False and Base.



Reduction

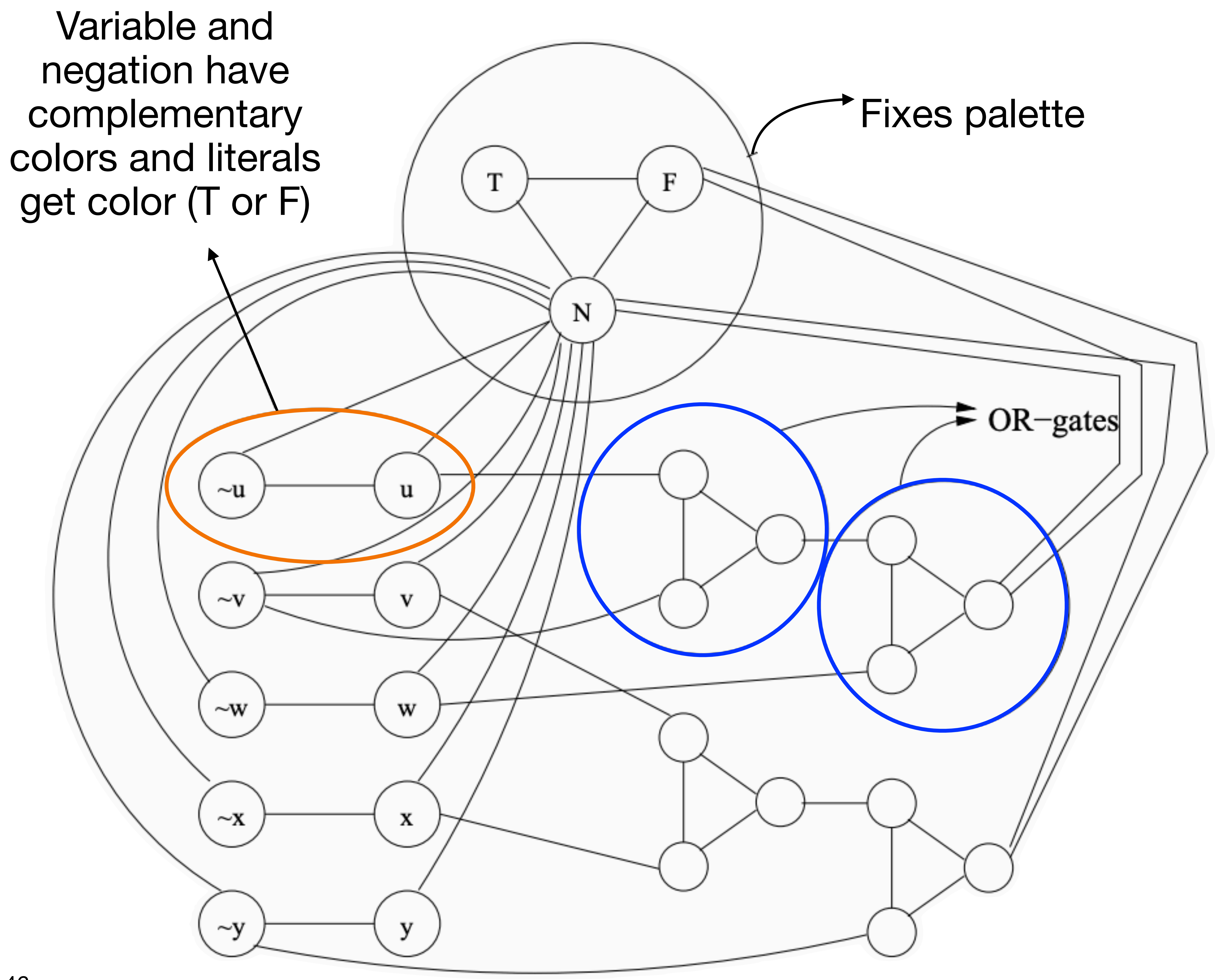
3SAT $\le_p$ 3-COLOR

- 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.
- **Claim:** No legal 3-coloring of below 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 below graph.



Reduction Outline

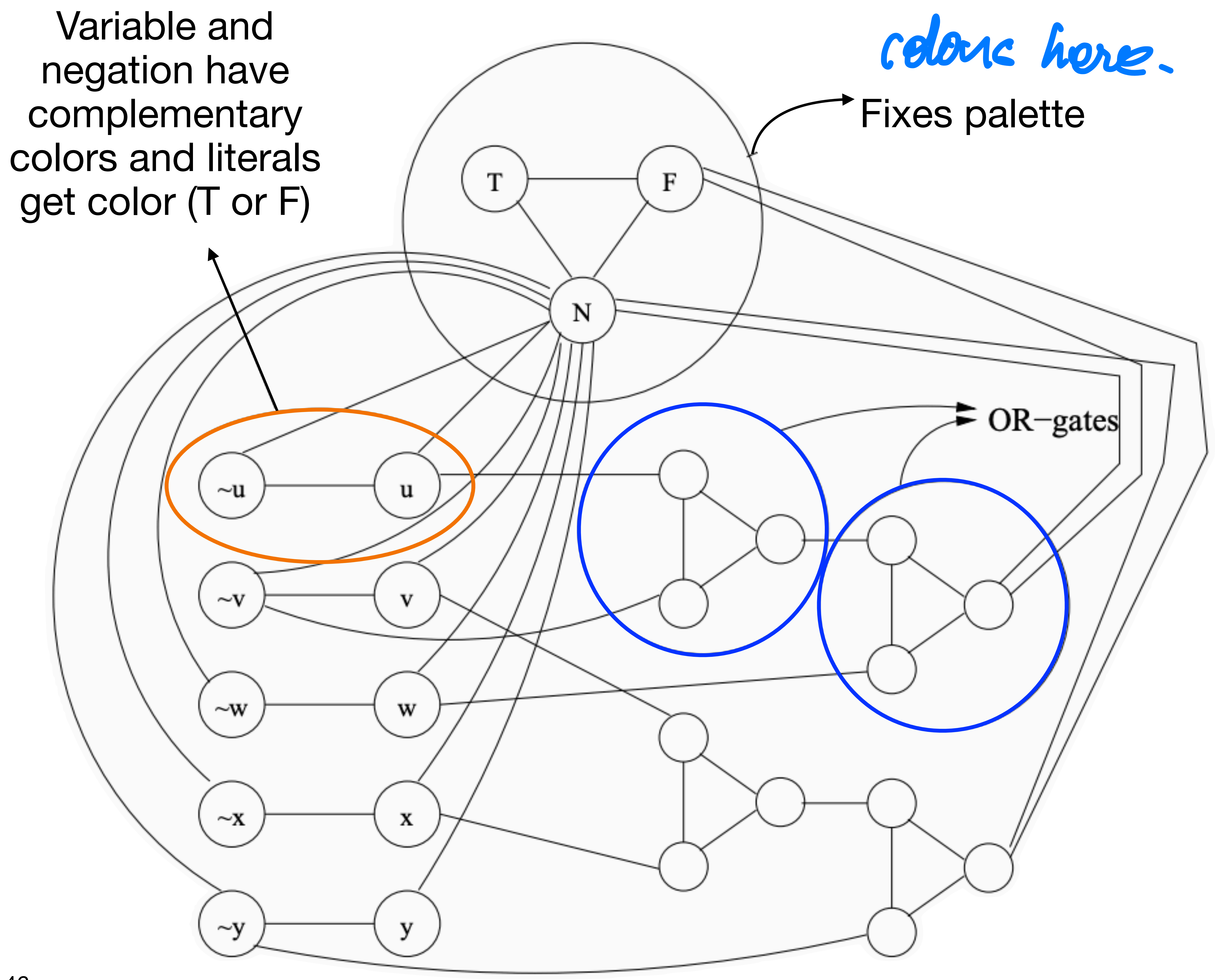
Example:



Reduction Outline

Example:

$$\varphi = \color{blue}{(u \vee \neg v \vee w)} \color{black}{\wedge (v \vee x \vee \neg y)}$$



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
- 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

Circuit-SAT Problem

Circuits

A circuit is a ***directed acyclic graph*** with

Circuit-SAT Problem

Circuits

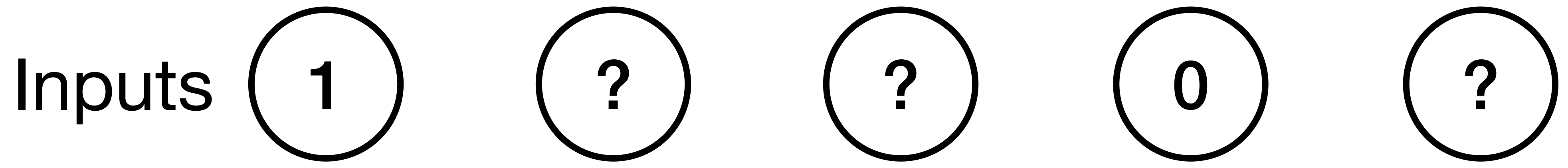
A circuit is a ***directed acyclic graph*** with

- Input vertices (without incoming edges) labeled with $0,1$ or a distinct variable.

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.

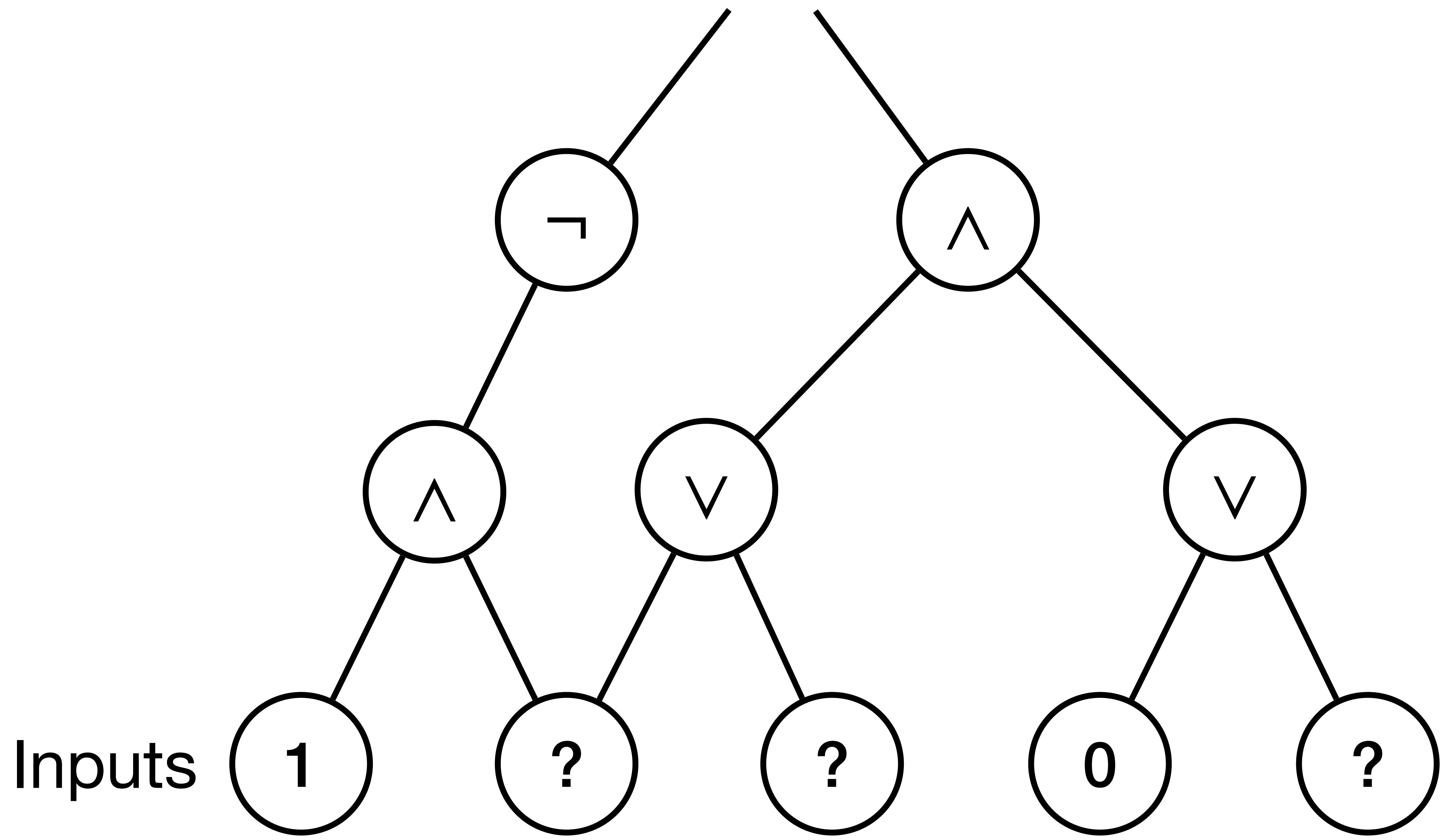


Circuit-SAT Problem

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 $\neg$.

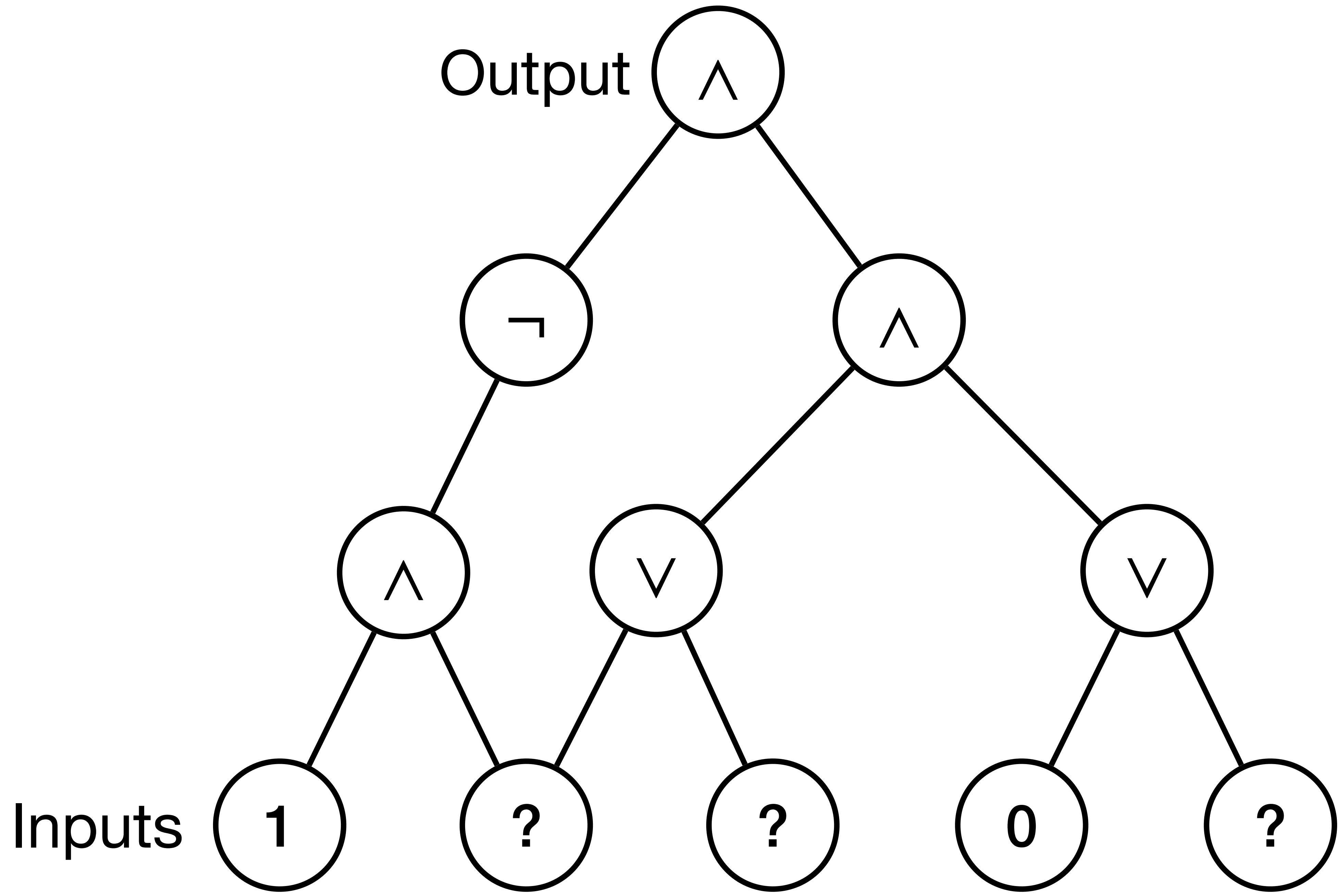


Circuit-SAT Problem

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.

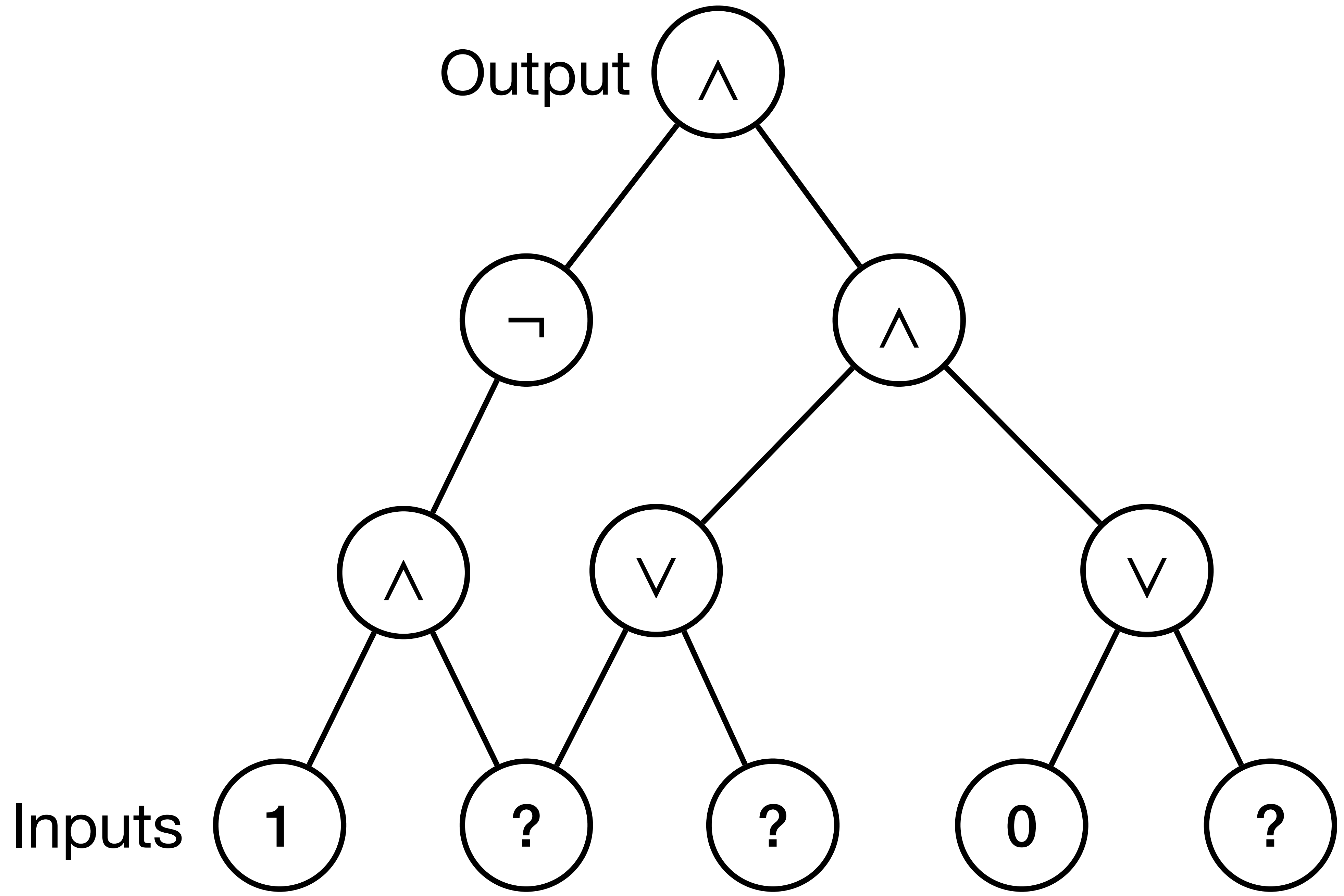


Circuit-SAT Problem

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.



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

Circuit-SAT Problem

Circuits

Problem definition (CSAT): Given a ***circuit*** as 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.
- Can show: 3SAT $\leq_P$ CSAT

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

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.

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 $x_1, x_2, \ldots, x_n$

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

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 $(l_1 \vee l_2 \vee l_3)$ use two OR gates to mimic formula

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 $(l_1 \vee l_2 \vee l_3)$ 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

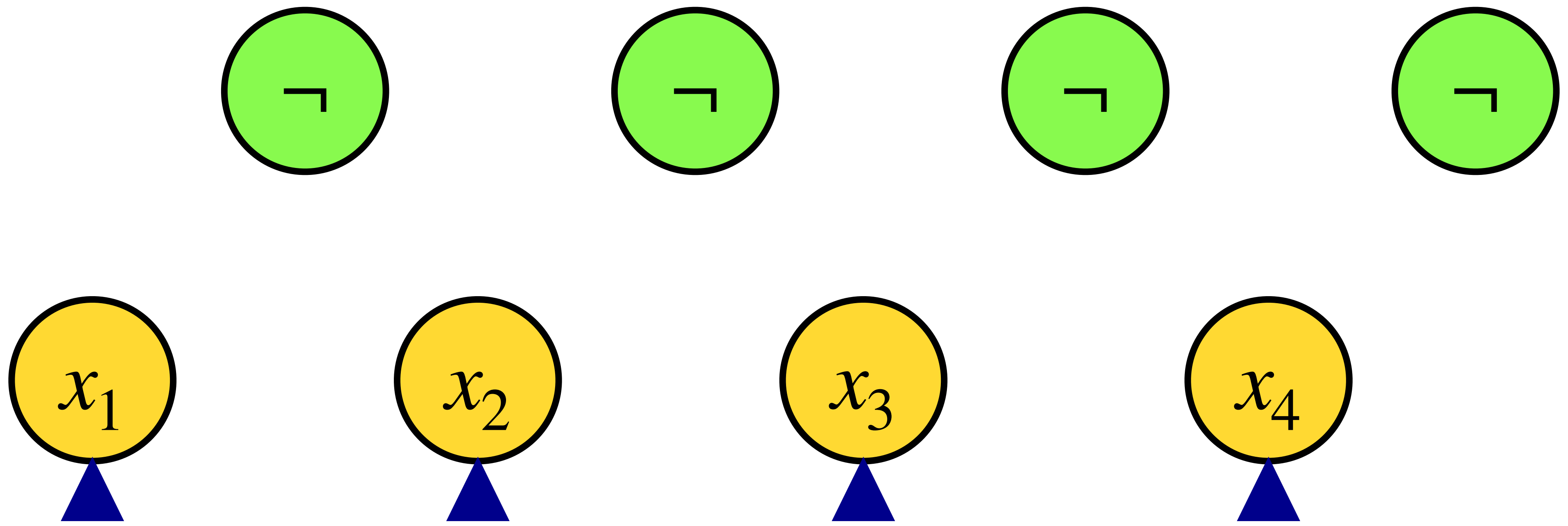
$$\varphi = (x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$

Example: 3SAT $\leq_P$ CSAT

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

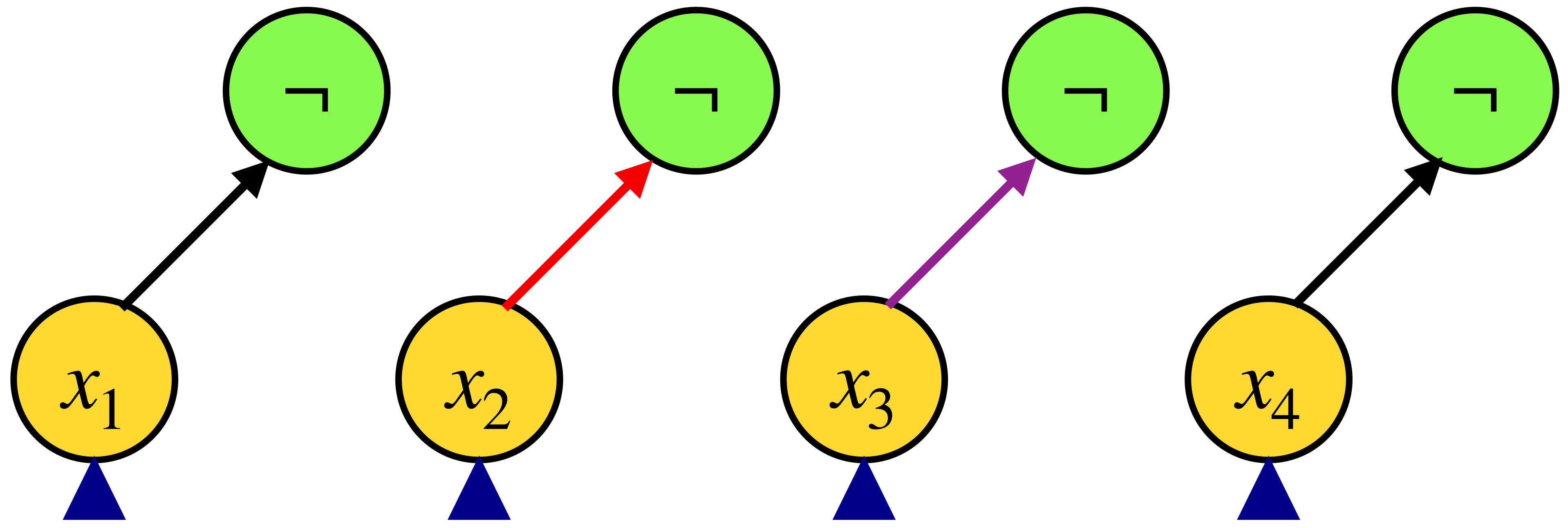
Example: 3SAT $\leq_P$ CSAT

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



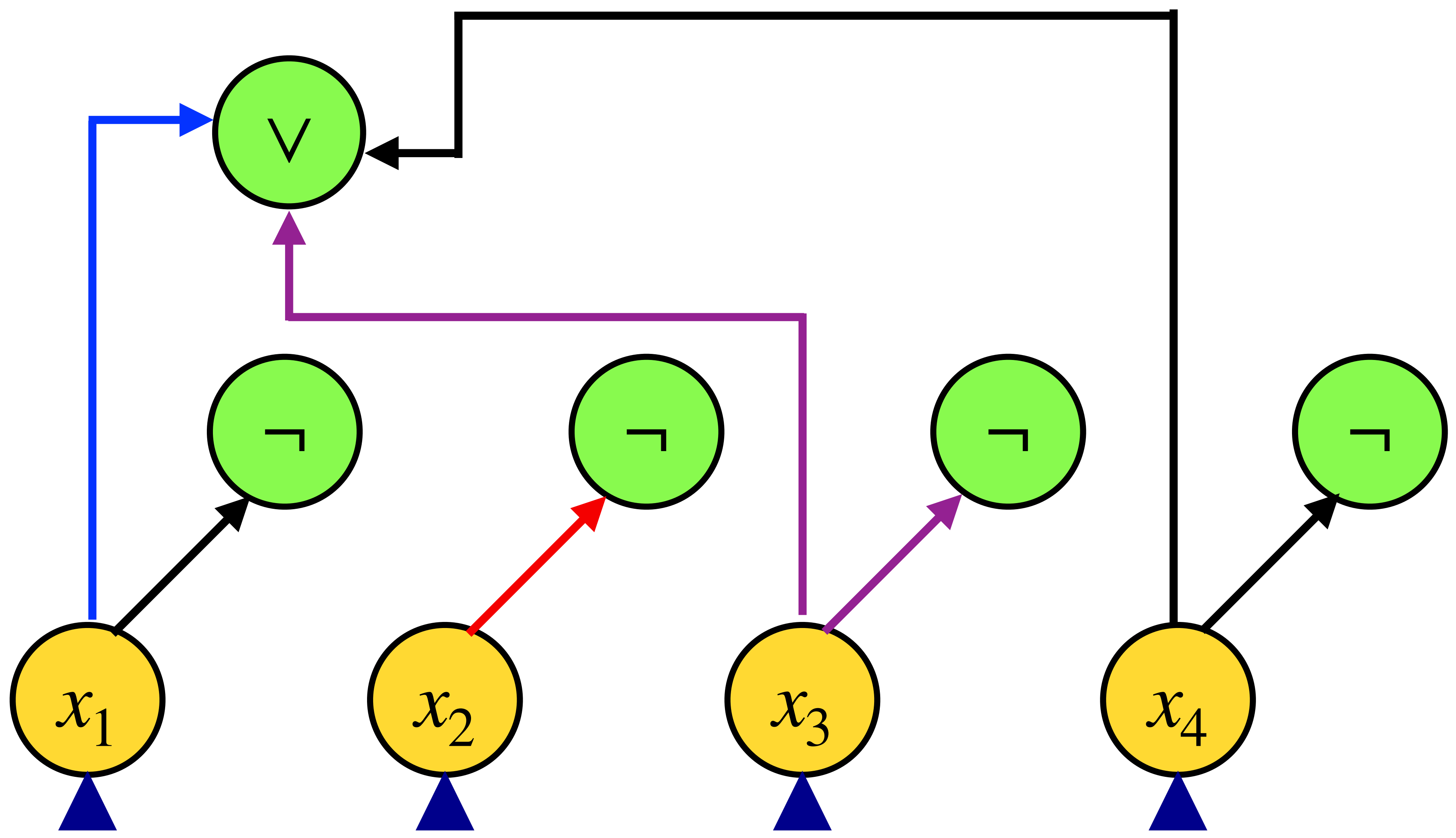
Example: 3SAT $\leq_P$ CSAT

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



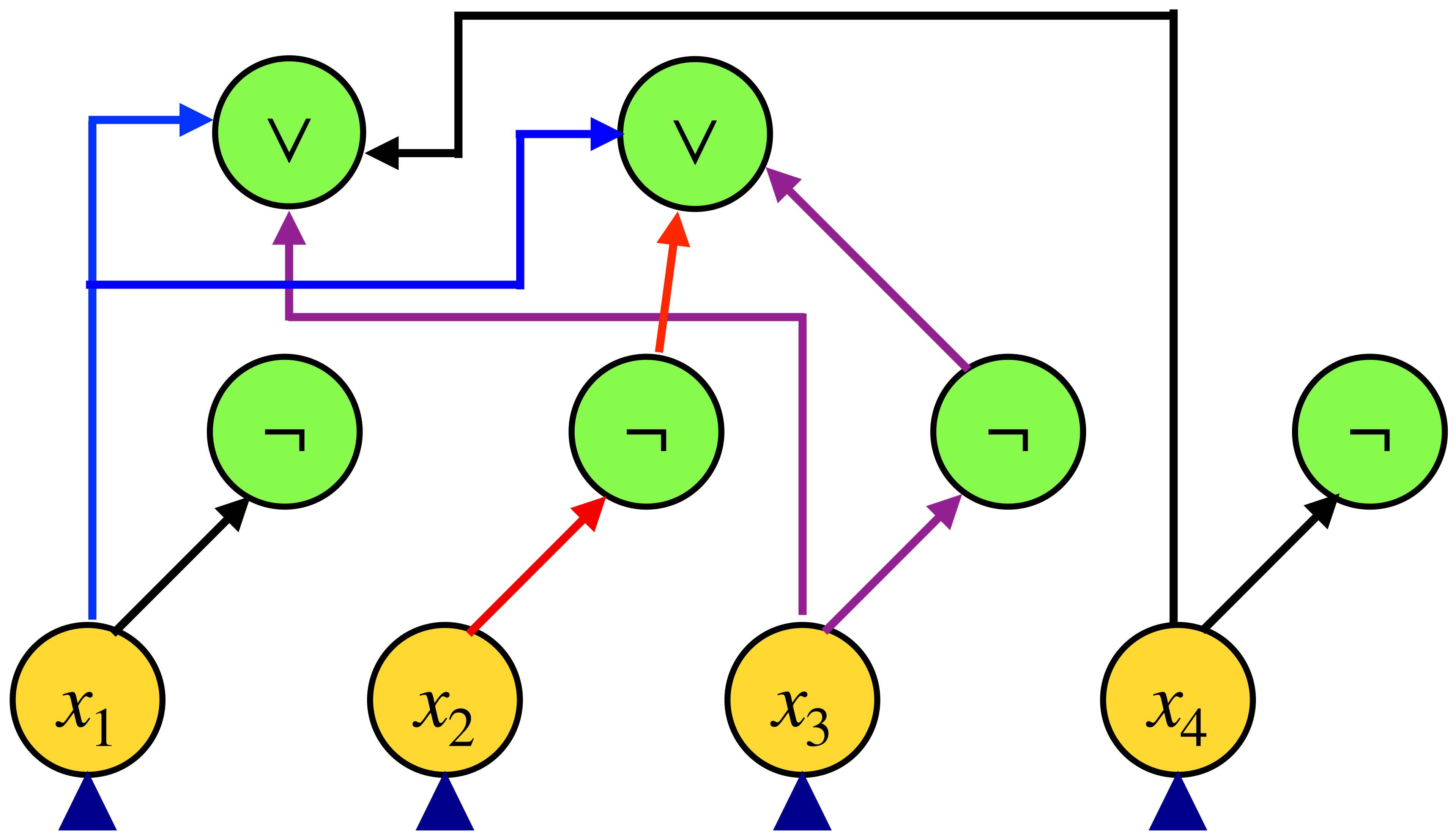
Example: 3SAT $\leq_P$ CSAT

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



Example: 3SAT $\leq_P$ CSAT

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



Example: 3SAT $\leq_P$ CSAT

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