CS 573: Algorithms, Fall 2014

More NP-Complete Problems

Lecture 4

September 4, 2014

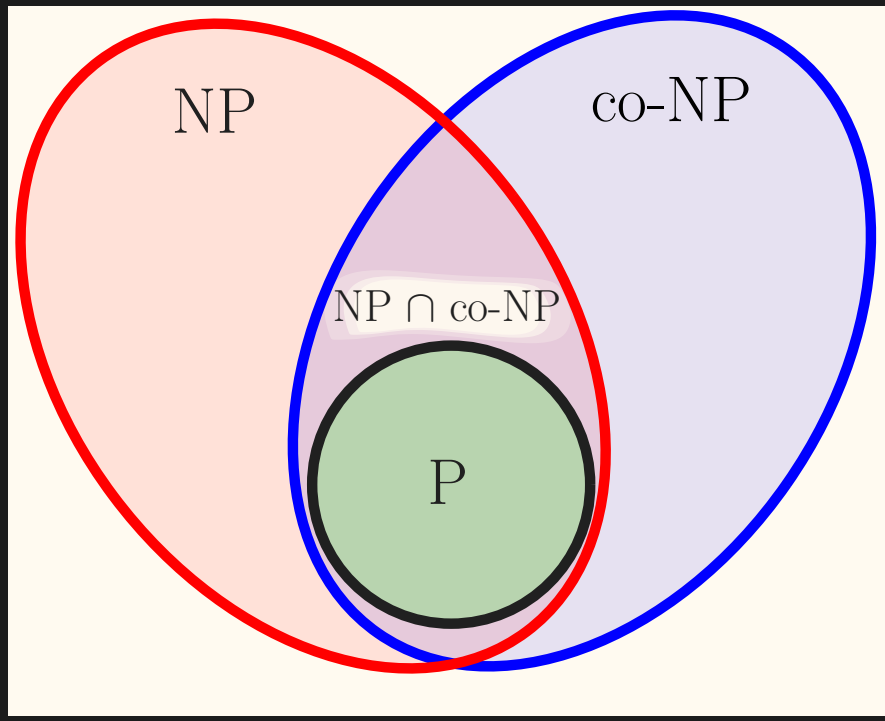
Part I

Quick & total recall

Recall...

NP

Decision problems with a polynomial certifier.
Examples: **SAT**, **Hamiltonian Cycle**, **3-Colorability**.



Definition

co-NP: class of all decision problems X s.t. $\overline{X} \in$ **NP**.
Examples: **UnSAT**, **No-Hamiltonian-Cycle**, **No-3-Colorable**.

Recall...

- **NP**: languages that have polynomial time certifiers/verifiers.
- A language L is **NP-Complete** $\iff$
  - L is in **NP**
  - for every L' in **NP**, $L' \leq_P L$
- L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.
- Cook-Level theorem...

Theorem (Cook-Levin)

Circuit-SAT *is* **NP-Complete**.

Recall...

- **NP**: languages that have polynomial time certifiers/verifiers.
- A language L is **NP-Complete** $\iff$
  - L is in **NP**
  - for every L' in **NP**, $L' \leq_P L$
- L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.
- Cook-Level theorem...

Theorem (Cook-Levin)

Circuit-SAT *is* **NP-Complete**.

Recall...

- **NP**: languages that have polynomial time certifiers/verifiers.
- A language L is **NP-Complete** $\iff$
  - L is in **NP**
  - for every L' in **NP**, $L' \leq_P L$
- L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.
- Cook-Level theorem...

Theorem (Cook-Levin)

Circuit-SAT *is* **NP-Complete**.

Recall...

1. **NP**: languages that have polynomial time certifiers/verifiers.
2. A language L is **NP-Complete** $\iff$
   - L is in **NP**
   - for every L' in **NP**, $L' \leq_P L$
3. L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.
4. Cook-Level theorem...

Theorem (Cook-Levin)

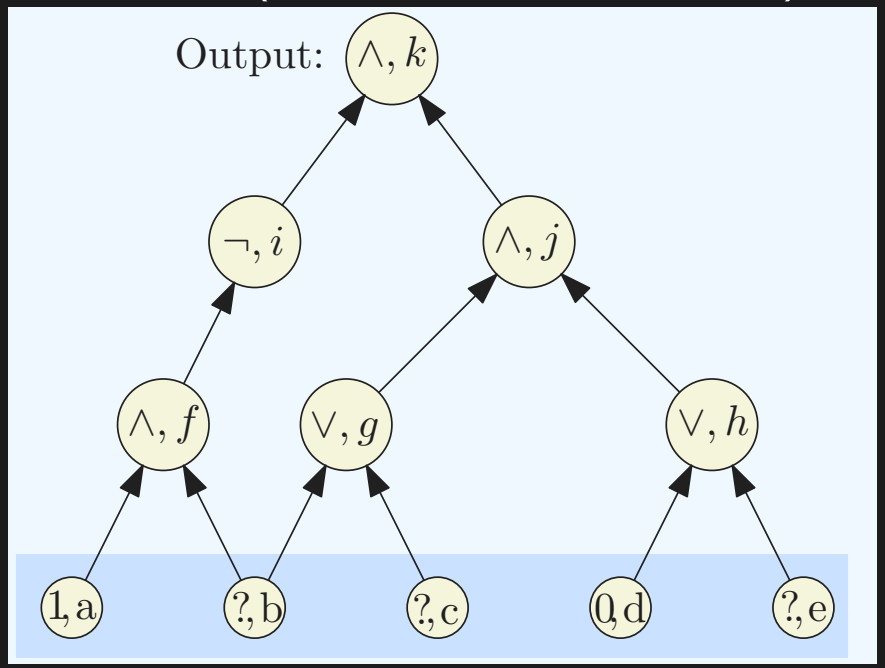
Circuit-SAT *is* **NP-Complete**.

Part II

Showing that **SAT** is **NP-Complete**

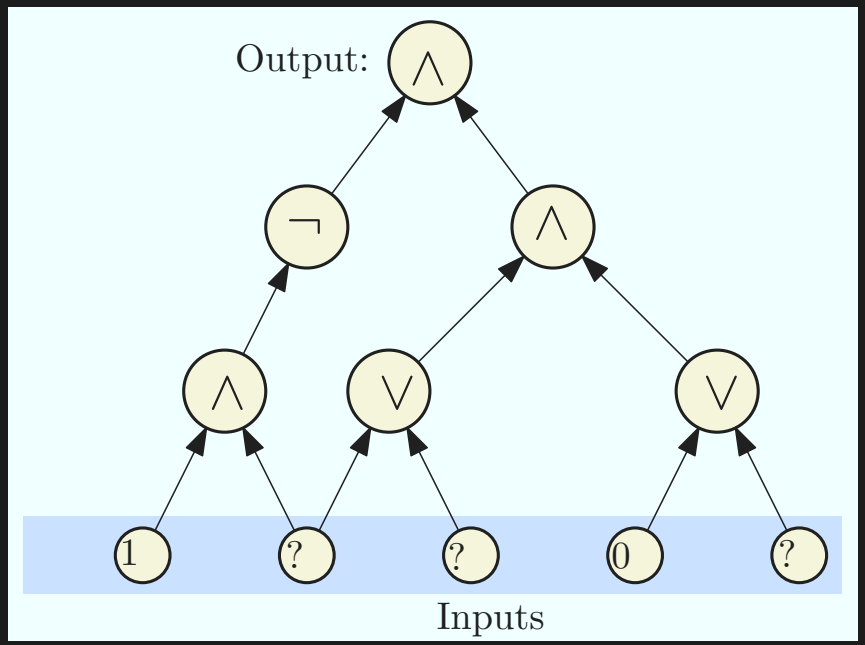
SAT is NP-Complete

- We have seen that **SAT** $\in$ **NP**
- To show **NP-Hardness**, we will reduce Circuit Satisfiability (**CSAT**) to **SAT**
  Instance of **CSAT** (we label each node):

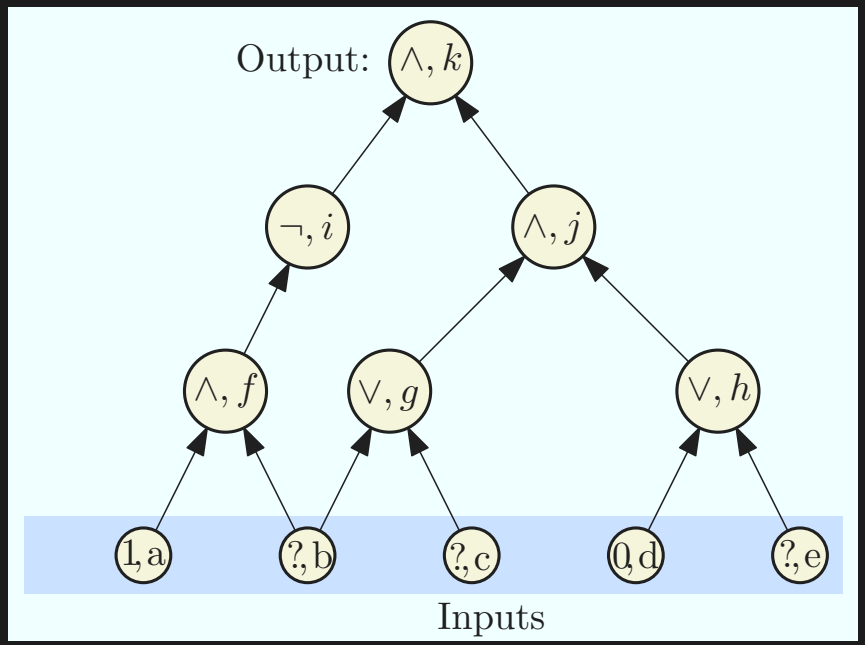


Converting a circuit into a CNF formula

Label the nodes



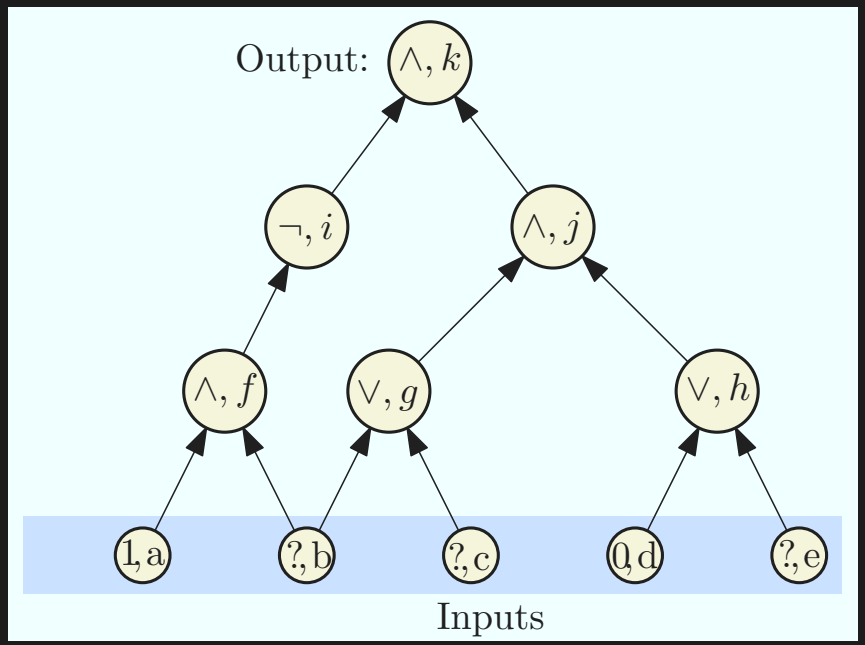
(A) Input circuit



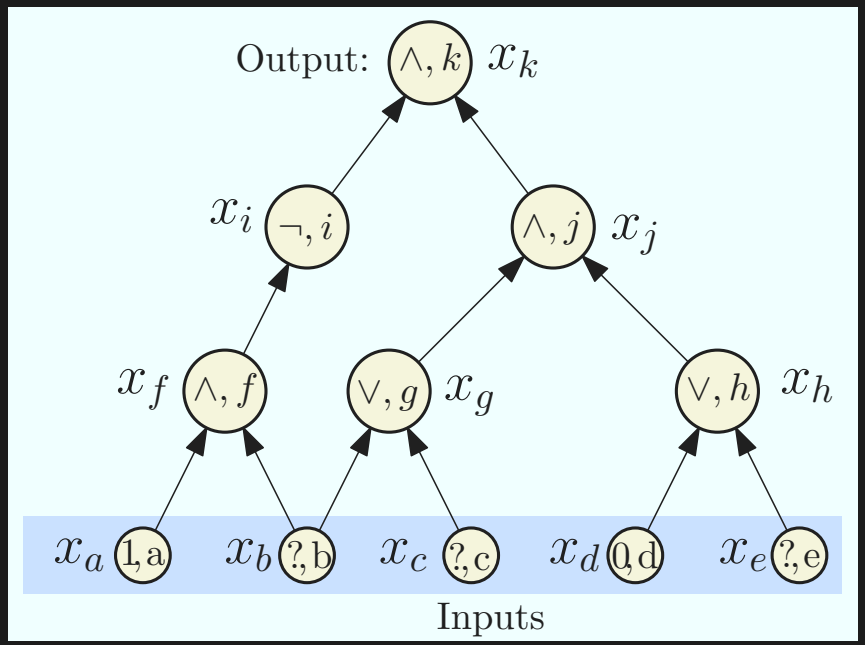
(B) Label the nodes.

Converting a circuit into a CNF formula

Introduce a variable for each node



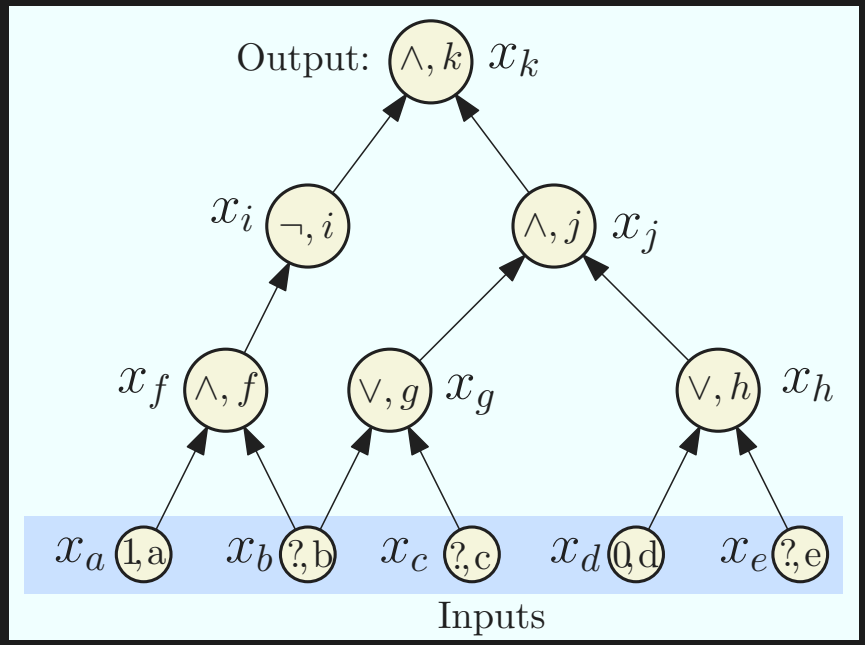
(B) Label the nodes.



(C) Introduce var for each node.

Converting a circuit into a CNF formula

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



(C) Introduce var for each node.

x_k (Demand a sat' assignment!)

$$x_k = x_i \wedge x_k$$
$$x_j = x_g \wedge x_h$$
$$x_i = \neg x_f$$
$$x_h = x_d \vee x_e$$
$$x_g = x_b \vee x_c$$
$$x_f = x_a \wedge x_b$$
$$x_d = 0$$
$$x_a = 1$$

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

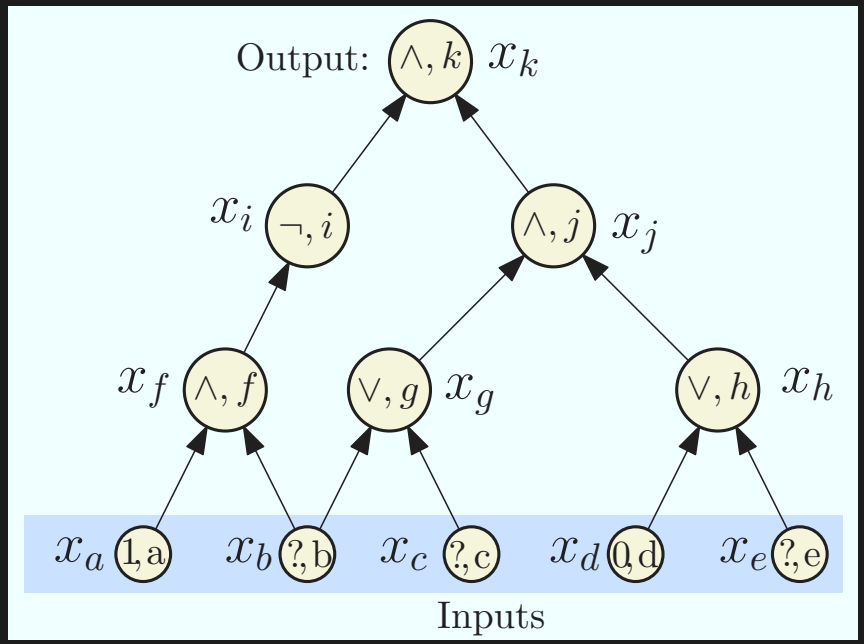
Converting a circuit into a CNF formula

Convert each sub-formula to an equivalent 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

Take the conjunction of all the CNF sub-formulas



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

We got a CNF formula that is satisfiable $\iff$ 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

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

Reduction: CSAT $\leq_{\mathrm{P}}$ SAT

Continued...

- 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

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

Reduction: CSAT $\leq_{\mathrm{P}}$ SAT

Continued...

- 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

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

Reduction: CSAT $\leq_{\mathrm{P}}$ SAT

Continued...

- 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

1. Find values of all gates in C under a
2. Give value of gate v to variable x_v; call this assignment a'
3. a' satisfies φ_C (exercise)

$\Leftarrow$ Consider a satisfying assignment a for φ_C

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

Theorem

SAT *is* **NP-Complete**.

Proving that a problem X is NP-Complete

1. To prove X is **NP-Complete**, show
   1. Show X is in **NP**.
      1. certificate/proof of polynomial size in input
      2. polynomial time certifier $C(s, t)$
   2. Reduction from a known **NP-Complete** problem such as **CSAT** or **SAT** to X
2. **SAT** $\leq_P$ X implies that every **NP** problem $Y \leq_P X$. Why?
   Transitivity of reductions:
3. $Y \leq_P$ **SAT** and **SAT** $\leq_P X$ and hence $Y \leq_P X$.

Proving that a problem X is NP-Complete

1. To prove X is **NP-Complete**, show
   1. Show X is in **NP**.
      1. certificate/proof of polynomial size in input
      2. polynomial time certifier $C(s, t)$
   2. Reduction from a known **NP-Complete** problem such as **CSAT** or **SAT** to X
2. **SAT** $\leq_P$ X implies that every **NP** problem $Y \leq_P X$. Why?

   Transitivity of reductions:
3. $Y \leq_P$ **SAT** and **SAT** $\leq_P X$ and hence $Y \leq_P X$.

Proving that a problem X is NP-Complete

1. To prove X is **NP-Complete**, show
   1. Show X is in **NP**.
      1. certificate/proof of polynomial size in input
      2. polynomial time certifier $C(s, t)$
   2. Reduction from a known **NP-Complete** problem such as **CSAT** or **SAT** to X
2. **SAT** $\leq_P$ X implies that every **NP** problem $Y \leq_P X$.
   Why?
   Transitivity of reductions:
3. $Y \leq_P$ **SAT** and **SAT** $\leq_P X$ and hence $Y \leq_P X$.

Proving that a problem X is NP-Complete

1. To prove X is **NP-Complete**, show
   1. Show X is in **NP**.
      1. certificate/proof of polynomial size in input
      2. polynomial time certifier $C(s, t)$
   2. Reduction from a known **NP-Complete** problem such as **CSAT** or **SAT** to X
2. **SAT** $\leq_P$ X implies that every **NP** problem $Y \leq_P X$. Why?
   Transitivity of reductions:
3. $Y \leq_P$ **SAT** and **SAT** $\leq_P X$ and hence $Y \leq_P X$.

NP-Completeness via Reductions

- What we currently know:
  1. **CSAT** is **NP-Complete**.
  2. **CSAT** $\leq_P$ **SAT** and **SAT** is in **NP** and hence **SAT** is **NP-Complete**.
  3. **SAT** $\leq_P$ **3SAT** and hence **3SAT** is **NP-Complete**.
  4. **3SAT** $\leq_P$ **Independent Set** (which is in **NP**) and hence **Independent Set** is **NP-Complete**.
  5. **Vertex Cover** is **NP-Complete**.
  6. **Clique** is **NP-Complete**.
- Hundreds and thousands of different problems from many areas of science and engineering have been shown to be **NP-Complete**.
- A surprisingly frequent phenomenon!

NP-Completeness via Reductions

1. What we currently know:
   1. **CSAT** is **NP-Complete**.
   2. **CSAT** $\leq_P$ **SAT** and **SAT** is in **NP** and hence **SAT** is **NP-Complete**.
   3. **SAT** $\leq_P$ **3SAT** and hence **3SAT** is **NP-Complete**.
   4. **3SAT** $\leq_P$ **Independent Set** (which is in **NP**) and hence **Independent Set** is **NP-Complete**.
   5. **Vertex Cover** is **NP-Complete**.
   6. **Clique** is **NP-Complete**.
2. Hundreds and thousands of different problems from many areas of science and engineering have been shown to be **NP-Complete**.
3. A surprisingly frequent phenomenon!

NP-Completeness via Reductions

1. What we currently know:
   1. **CSAT** is **NP-Complete**.
   2. **CSAT** $\leq_P$ **SAT** and **SAT** is in **NP** and hence **SAT** is **NP-Complete**.
   3. **SAT** $\leq_P$ **3SAT** and hence **3SAT** is **NP-Complete**.
   4. **3SAT** $\leq_P$ **Independent Set** (which is in **NP**) and hence **Independent Set** is **NP-Complete**.
   5. **Vertex Cover** is **NP-Complete**.
   6. **Clique** is **NP-Complete**.
2. Hundreds and thousands of different problems from many areas of science and engineering have been shown to be **NP-Complete**.
3. A surprisingly frequent phenomenon!

Part III

More reductions...

Next...

Prove

- **Hamiltonian Cycle** Problem is **NP-Complete**.
- 3-Coloring is **NP-Complete**.
- **Subset Sum**.

Part IV

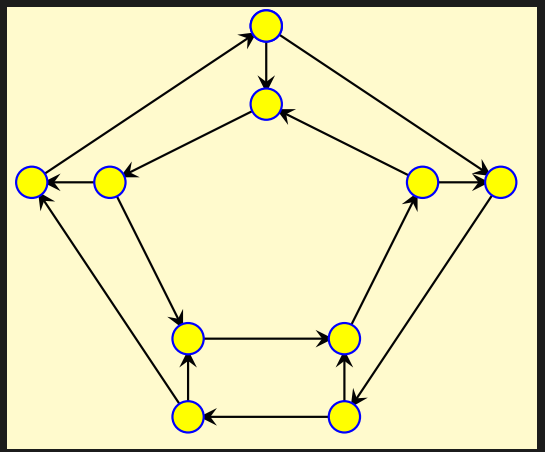
NP-Completeness of Hamiltonian Cycle

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

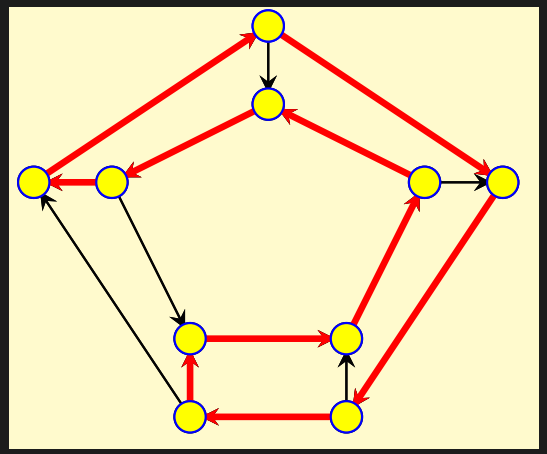


Directed Hamiltonian Cycle

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



Directed Hamiltonian Cycle is **NP-Complete**

- Directed Hamiltonian Cycle is in NP
  - Certificate: Sequence of vertices
  - Certifier: Check if every vertex (except the first) appears exactly once, and that consecutive vertices are connected by a directed edge
- Hardness: Will prove...
  3SAT $\leq_P$ Directed Hamiltonian Cycle.

Directed Hamiltonian Cycle is **NP-Complete**

- Directed Hamiltonian Cycle is in NP
  - Certificate: Sequence of vertices
  - Certifier: Check if every vertex (except the first) appears exactly once, and that consecutive vertices are connected by a directed edge
- Hardness: Will prove...
  3SAT $\leq_P$ Directed Hamiltonian Cycle.

Reduction

- **3SAT** formula φ create a graph G_φ such that
  - G_φ has a Hamiltonian cycle $\iff$ 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

- **3SAT** formula φ create a graph G_φ such that
  - G_φ has a Hamiltonian cycle $\iff$ 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

- **3SAT** formula φ create a graph G_φ such that
  - G_φ has a Hamiltonian cycle $\iff$ 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

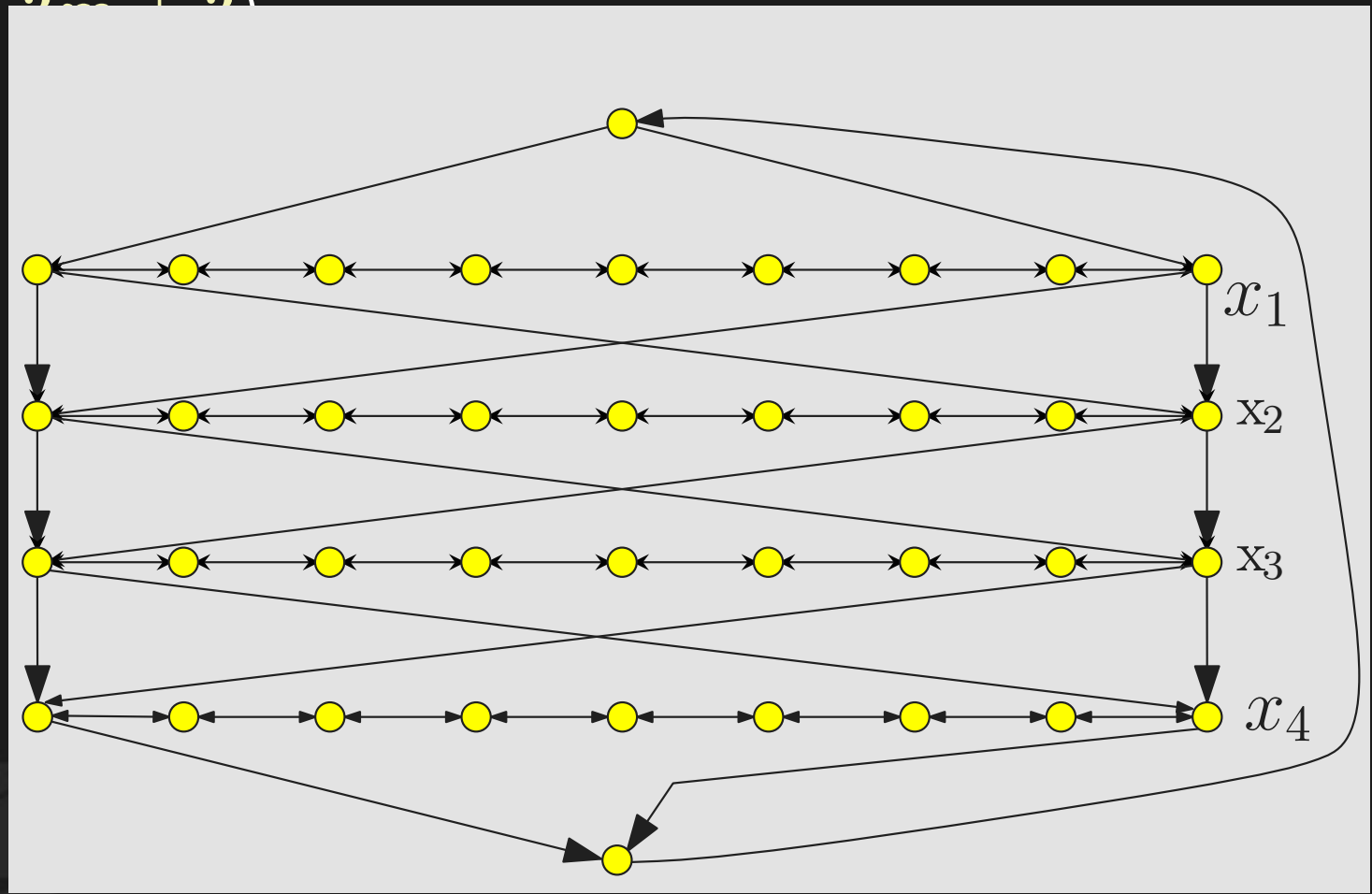
1. **3SAT** formula φ create a graph G_φ such that
   - G_φ has a Hamiltonian cycle $\iff$ φ is satisfiable
   - G_φ should be constructible from φ by a polynomial time algorithm $\mathcal{A}$
2. Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

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.

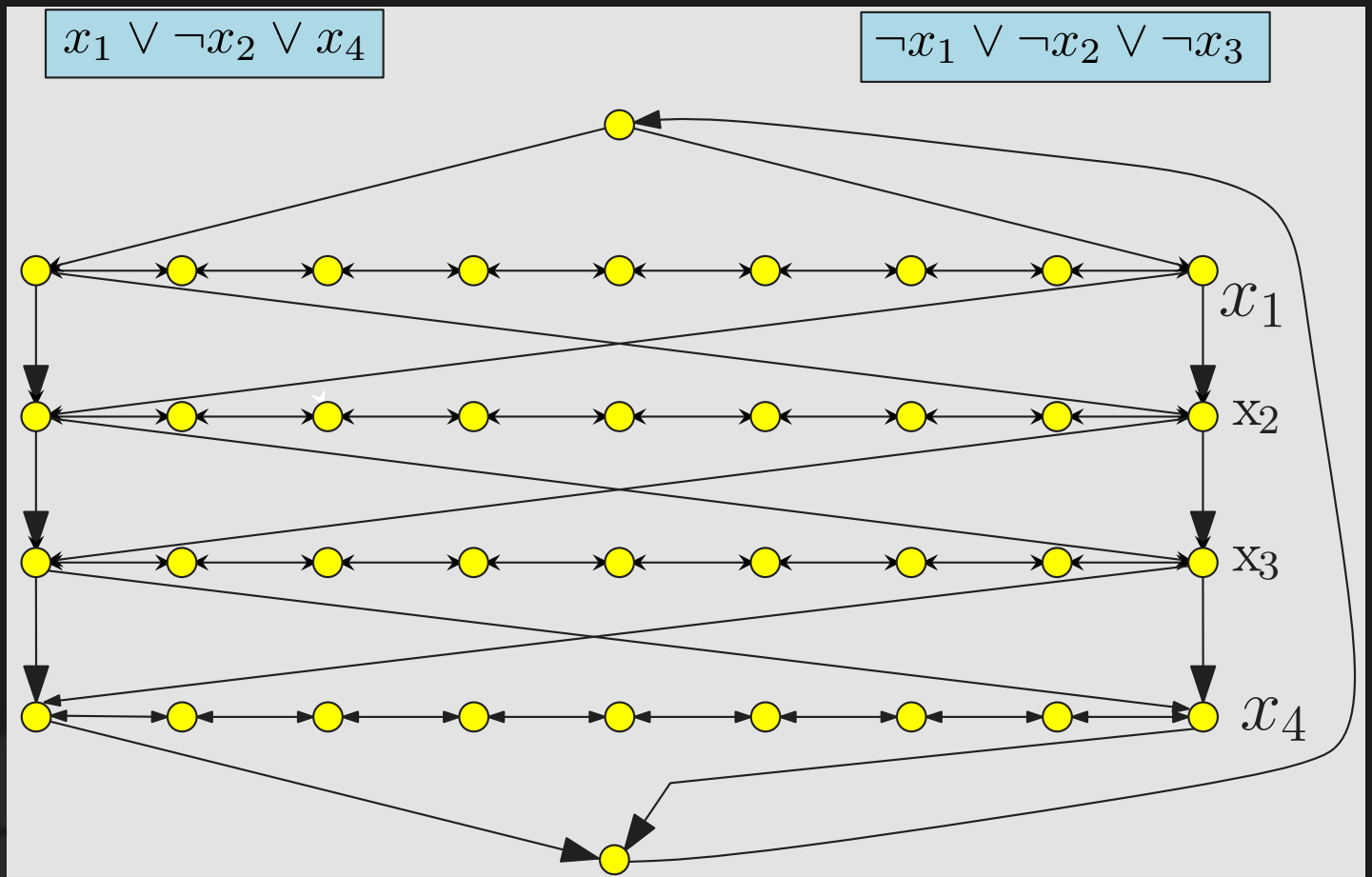
The Reduction: Phase I

- 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 $3m+3$)



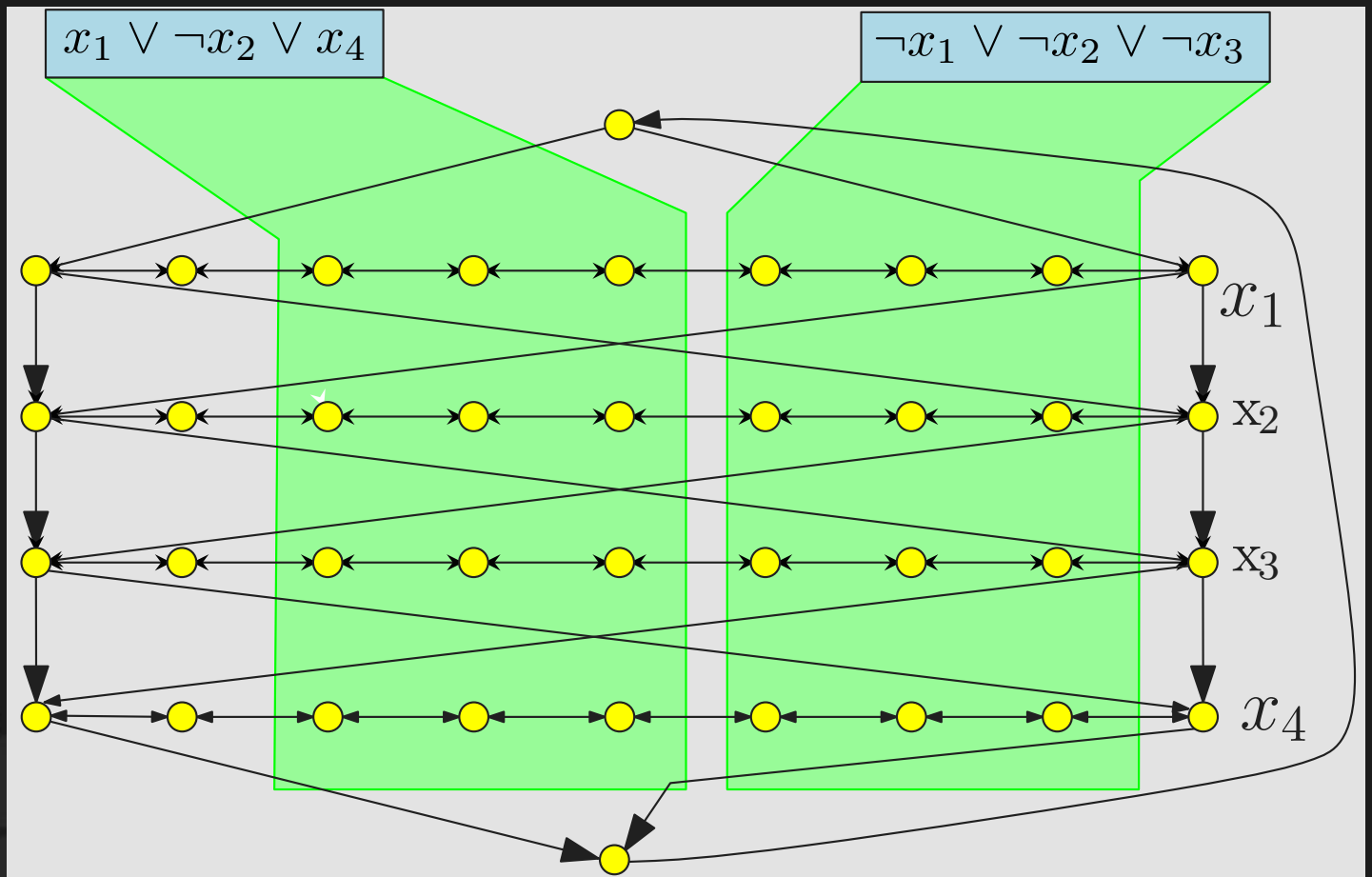
The Reduction: Phase II

- 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.



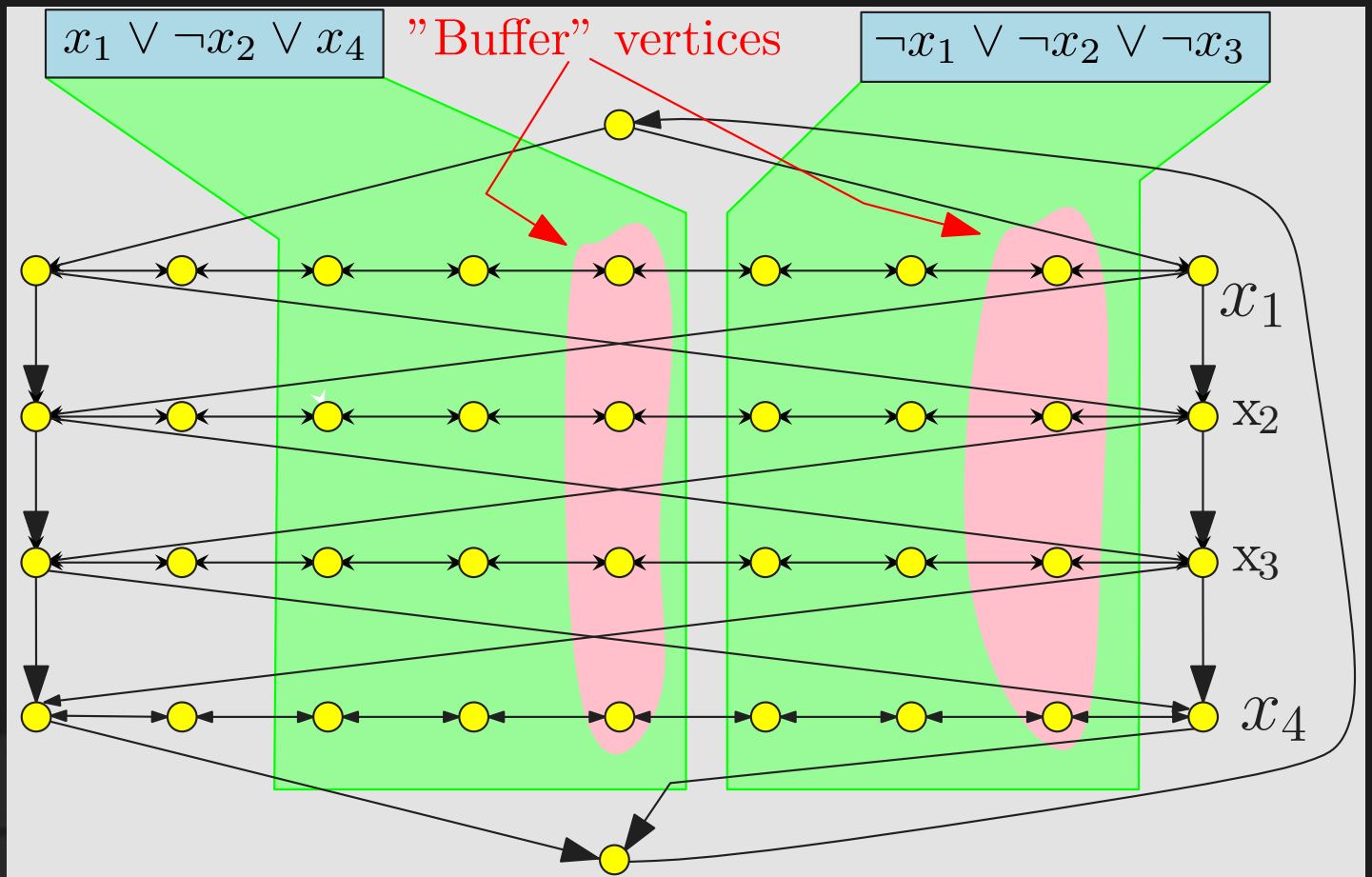
The Reduction: Phase II

- 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.



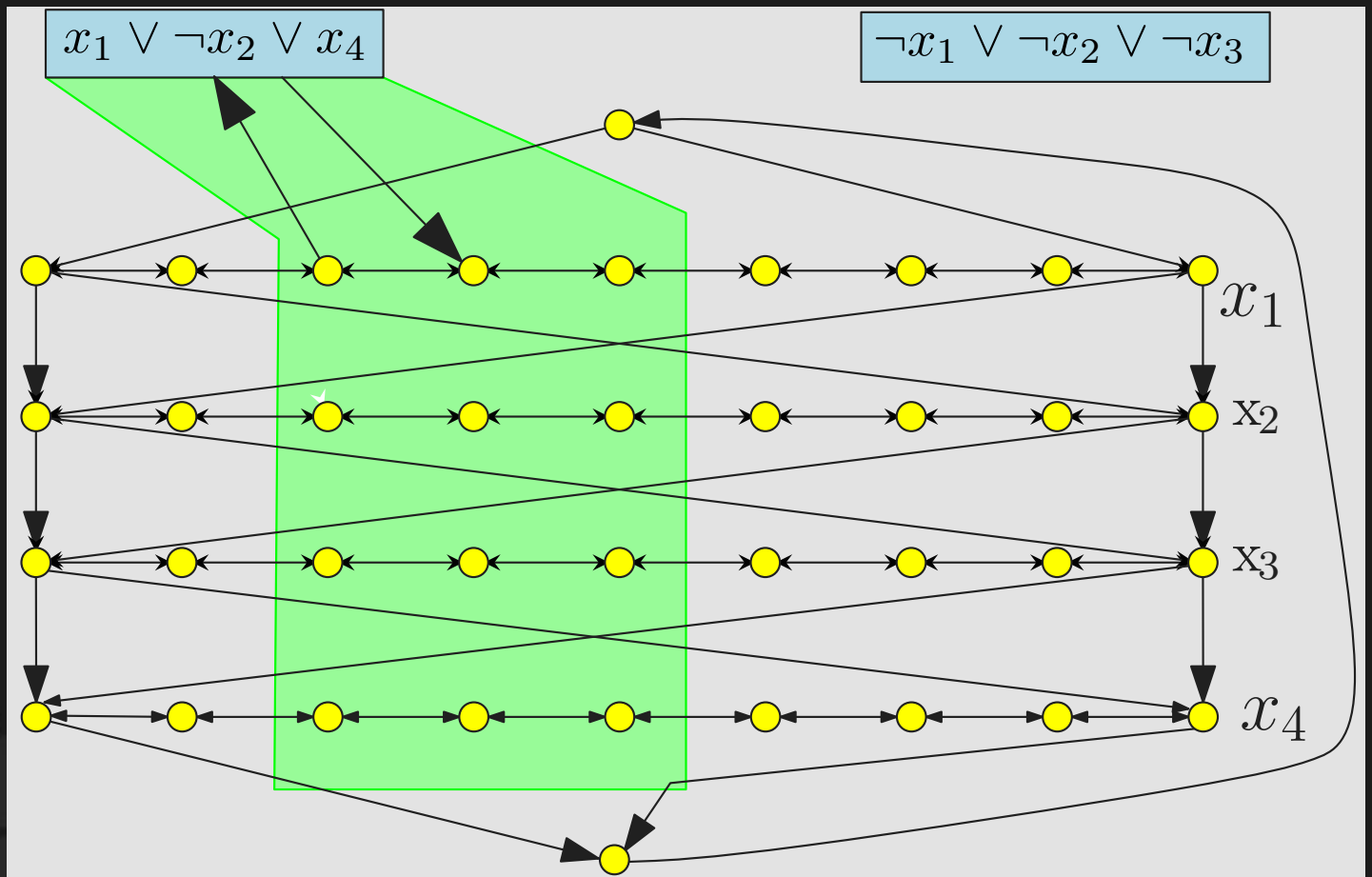
The Reduction: Phase II

- 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.



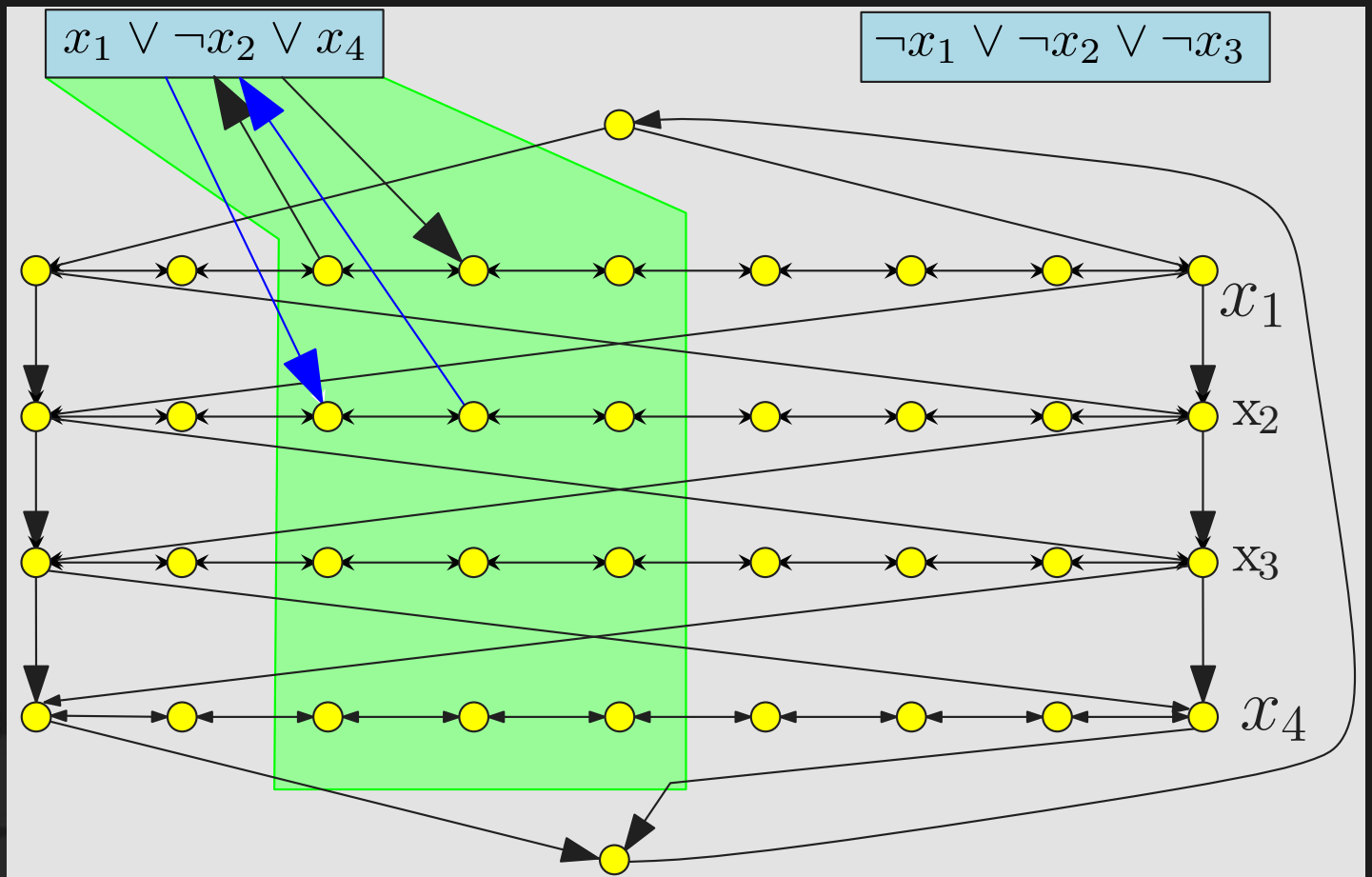
The Reduction: Phase II

- 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.



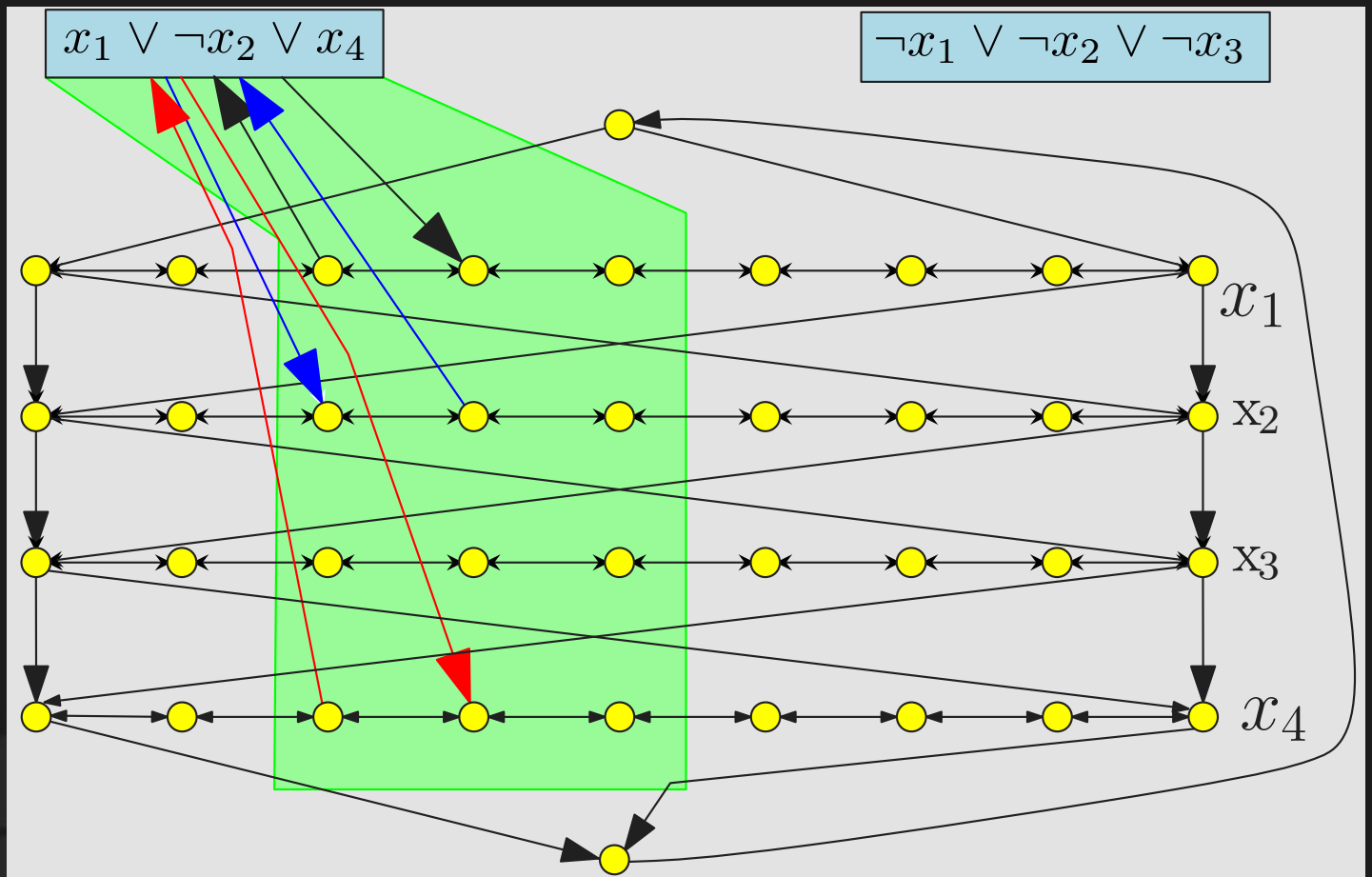
The Reduction: Phase II

- 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.



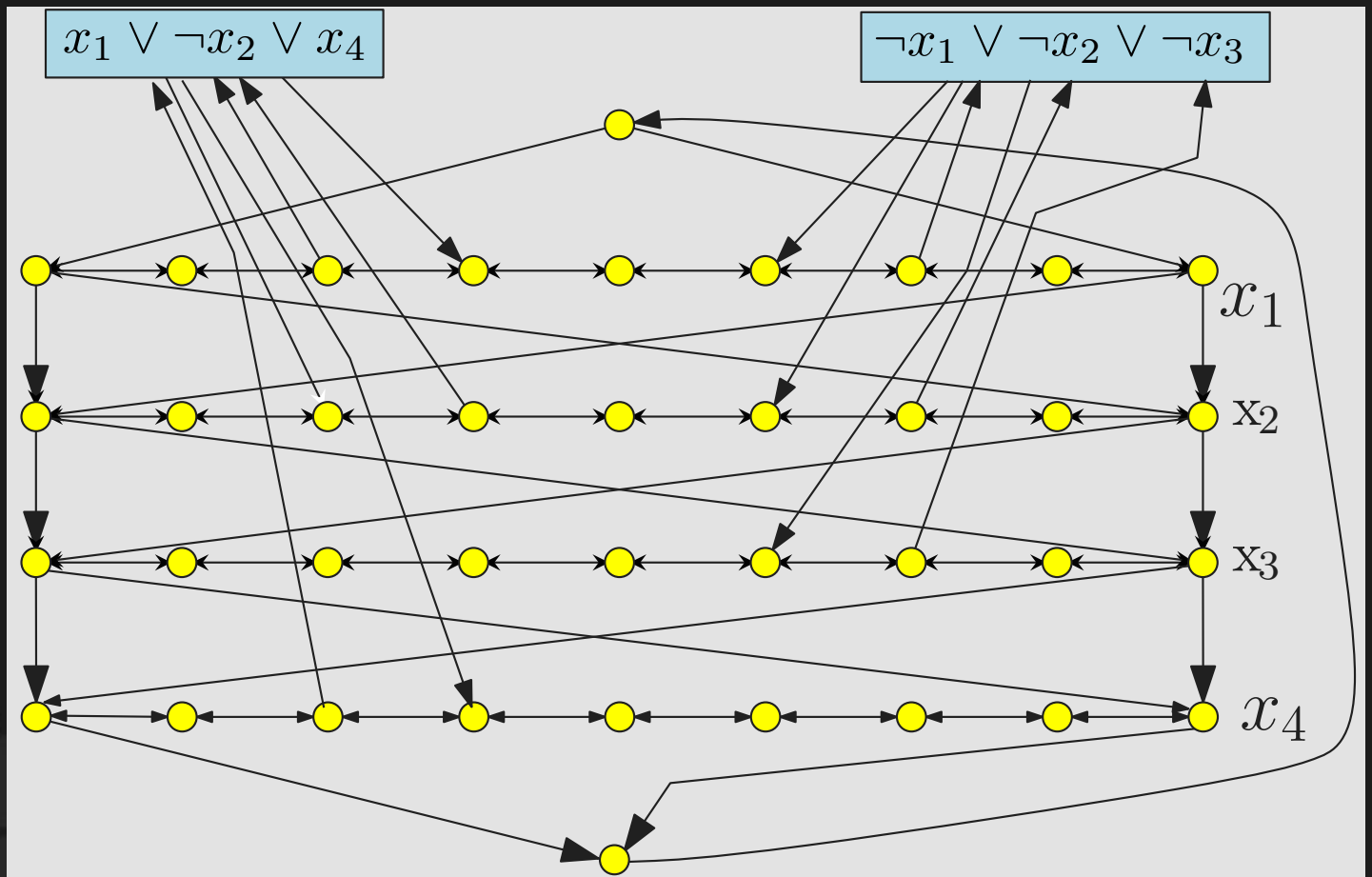
The Reduction: Phase II

- 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: Phase 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

Proposition

φ has satisfying assignment $\iff$ G_φ has Hamiltonian cycle.

Proof.

$\Rightarrow$ Let α be the satisfying assignment for φ. Define Hamiltonian cycle as follows

- If $\alpha(x_i) = 1$ then traverse path i from left to right
- If $\alpha(x_i) = 0$ 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. □

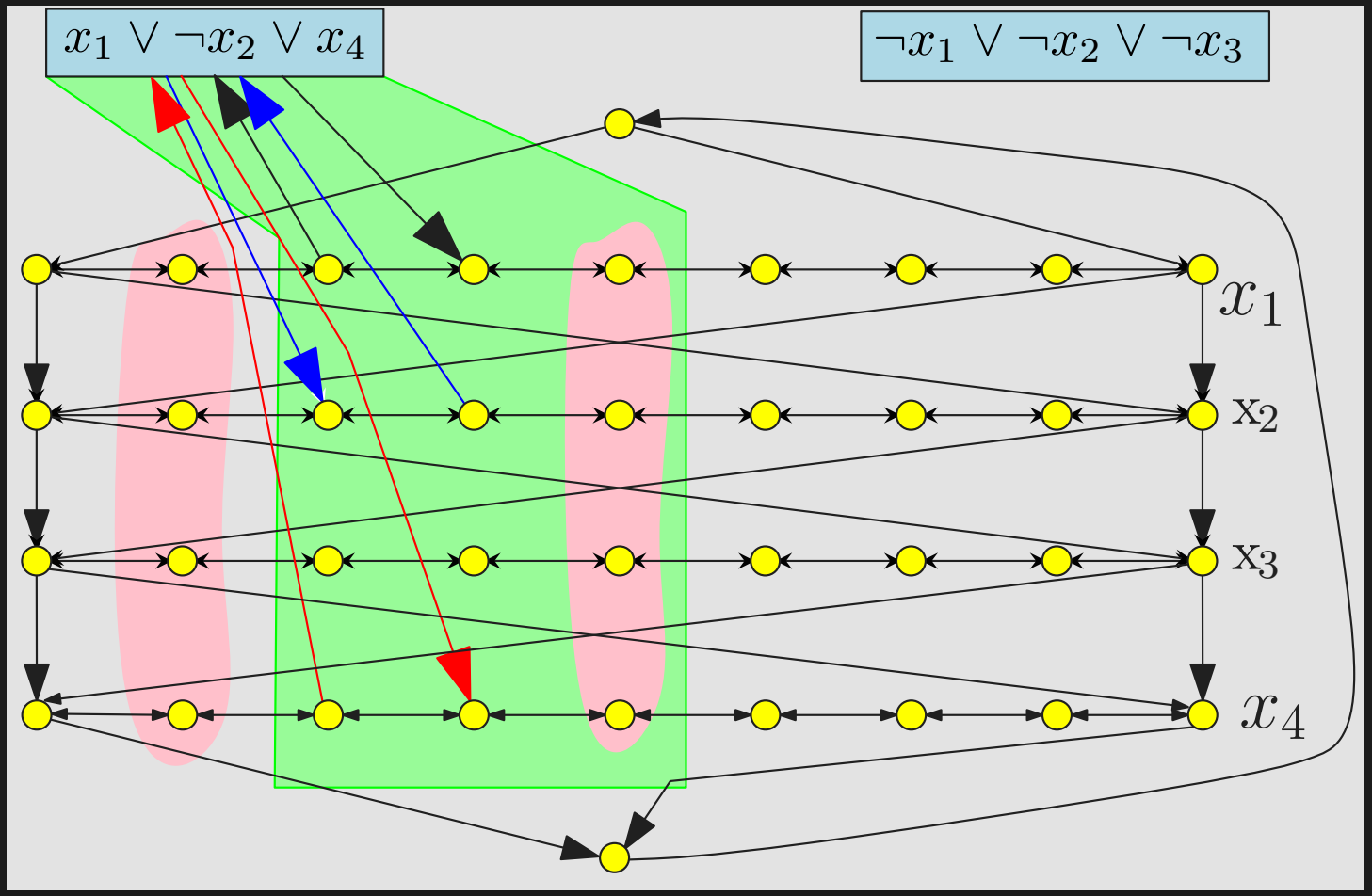
Hamiltonian Cycle $\Rightarrow$ Satisfying assignment

Proof continued

Suppose Π is a Hamiltonian cycle in G_φ

- 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

Example



Hamiltonian Cycle $\Longrightarrow$ Satisfying assignment (contd)

- Thus, vertices visited immediately before and after C_i are connected by an edge
- We can remove c_j from cycle, and get Hamiltonian cycle in $G - c_j$
- Consider Hamiltonian cycle in $G - \{c_1, \ldots c_m\}$; it traverses each path in only one direction, which determines the truth assignment

(Undirected) Hamiltonian Cycle

Problem (Undirected Hamiltonian Cycle)

Input: *Given undirected graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$.

Goal: *Does* $\mathbf{G}$ *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.

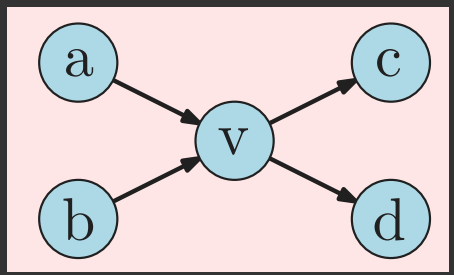
- The problem is in **NP**; proof left as exercise.
- Hardness proved by reducing Directed Hamiltonian Cycle to this problem. □

Reduction Sketch

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

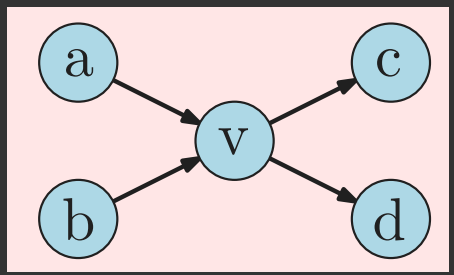


Reduction Sketch

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

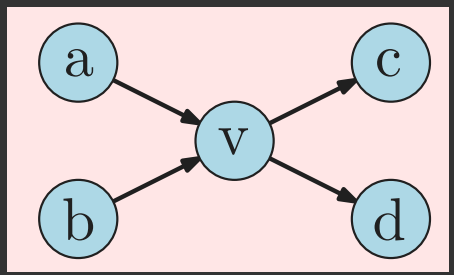


Reduction Sketch

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

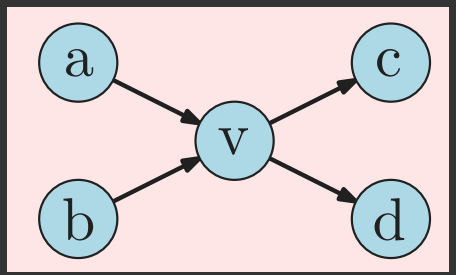


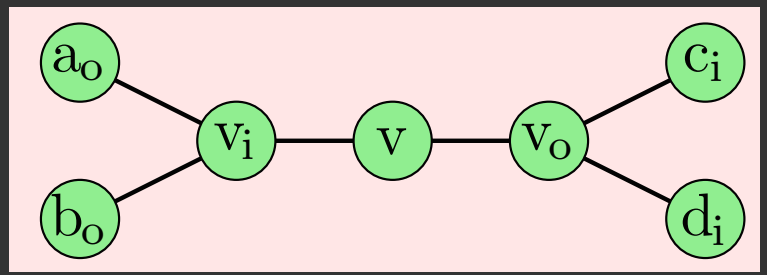
Reduction Sketch

Goal: Given directed graph G, need to construct undirected graph G' such that G has Hamiltonian Path if and only if 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: Wrapup

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

Part V

NP-Completeness of Graph Coloring

Graph Coloring

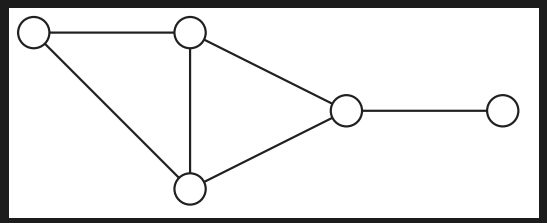
Graph Coloring

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

3 Coloring

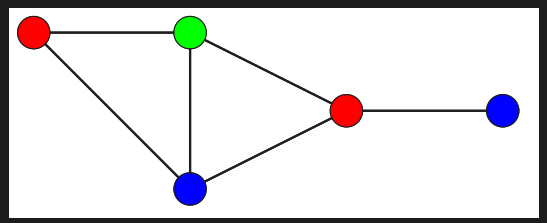
Instance: $G = (V, E)$: Undirected graph.
Question: Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?



Graph 3-Coloring

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 **BFS** (we saw this earlier).

Graph Coloring

1. 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.
2. Graph 2-Coloring can be decided in polynomial time.
3. G is 2-colorable $\iff$ G is bipartite! There is a linear time algorithm to check if G is bipartite using **BFS** (we saw this earlier).

Graph Coloring

1. 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.
2. Graph 2-Coloring can be decided in polynomial time.
3. G is 2-colorable $\iff$ G is bipartite! There is a linear time algorithm to check if G is bipartite using **BFS** (we saw this earlier).

Graph Coloring

1. 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.
2. Graph 2-Coloring can be decided in polynomial time.
3. G is 2-colorable $\iff$ G is bipartite! There is a linear time algorithm to check if G is bipartite using **BFS** (we saw this earlier).

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
- ... **3-COLOR** $\leq_P$ k**Register Allocation**, for any $k \geq 3$

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
- ... **3-COLOR** $\leq_P$ k**Register Allocation**, for any $k \geq 3$

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
- ... **3-COLOR** $\leq_P$ k**Register Allocation**, for any $k \geq 3$

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
- ... **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.

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.

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.

Class Room Scheduling

1. Given n classes and their meeting times, are k rooms sufficient?
   Reduce to Graph k-Coloring problem
2. 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*
3. Exercise: G is k-colorable $\iff$ k rooms are sufficient.

Class Room Scheduling

1. Given n classes and their meeting times, are k rooms sufficient?
   Reduce to Graph k-Coloring problem
2. 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*
3. 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
- 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.

Frequency Assignments in Cellular Networks

1. 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
2. Problem: given k bands and some region with n towers, is there a way to assign the bands to avoid interference?
3. Can reduce to k-coloring by creating interference/conflict graph on towers.

Frequency Assignments in Cellular Networks

1. 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
2. Problem: given k bands and some region with n towers, is there a way to assign the bands to avoid interference?
3. Can reduce to k-coloring by creating interference/conflict graph on towers.

3-Coloring is NP-Complete

- **3-Coloring** is in **NP**.
  - Certificate: for each node a color from $\{1, 2, 3\}$.
  - Certifier: Check if for each edge (u, v), the color of u is different from that of v.
- Hardness: Show...
  3SAT $\leq_P$ **3-Coloring**.

3-Coloring is NP-Complete

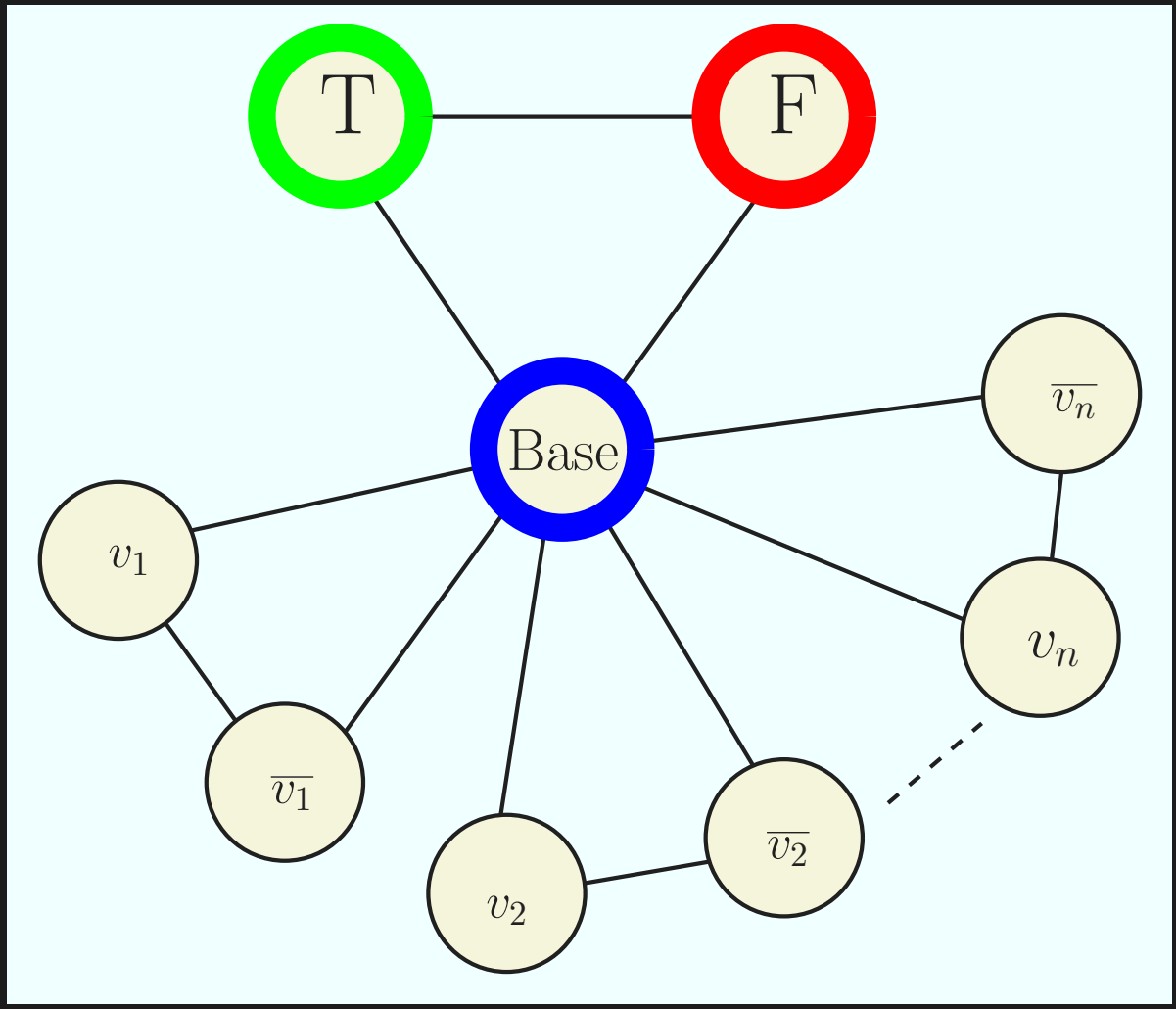
- **3-Coloring** is in **NP**.
  - Certificate: for each node a color from $\{1, 2, 3\}$.
  - Certifier: Check if for each edge (u, v), the color of u is different from that of v.
- Hardness: Show...
  3SAT $\leq_P$ **3-Coloring**.

Reduction Idea

Start with **3SAT** formula (i.e., $3\mathrm{CNF}$ 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

1. Need to establish truth assignment for $x_1, \ldots, x_n$ via colors for some nodes in G_φ.
2. Create triangle with nodes **true**, **false**, **base**.
3. For each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with the special node **base**.
4. 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.
5. Need to add constraints to ensure clauses are satisfied (next phase).

Figure

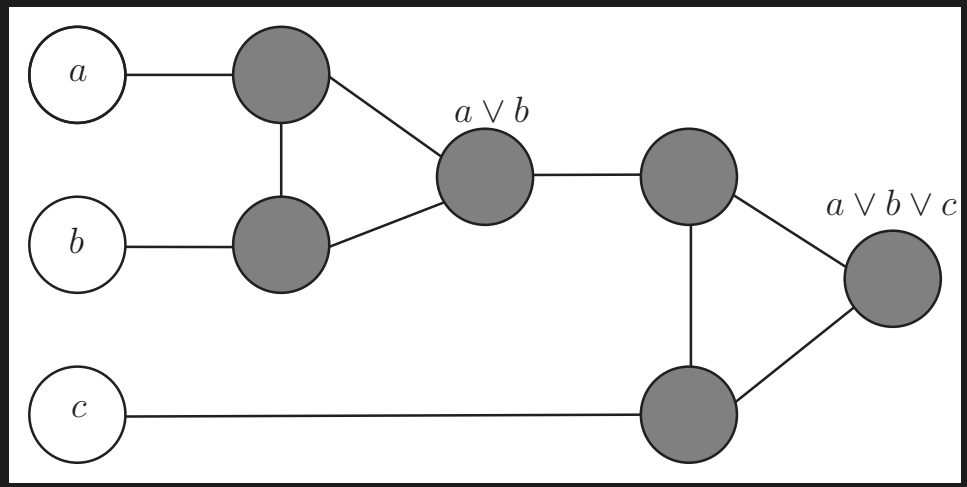


Clause Satisfiability Gadget

- 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:

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

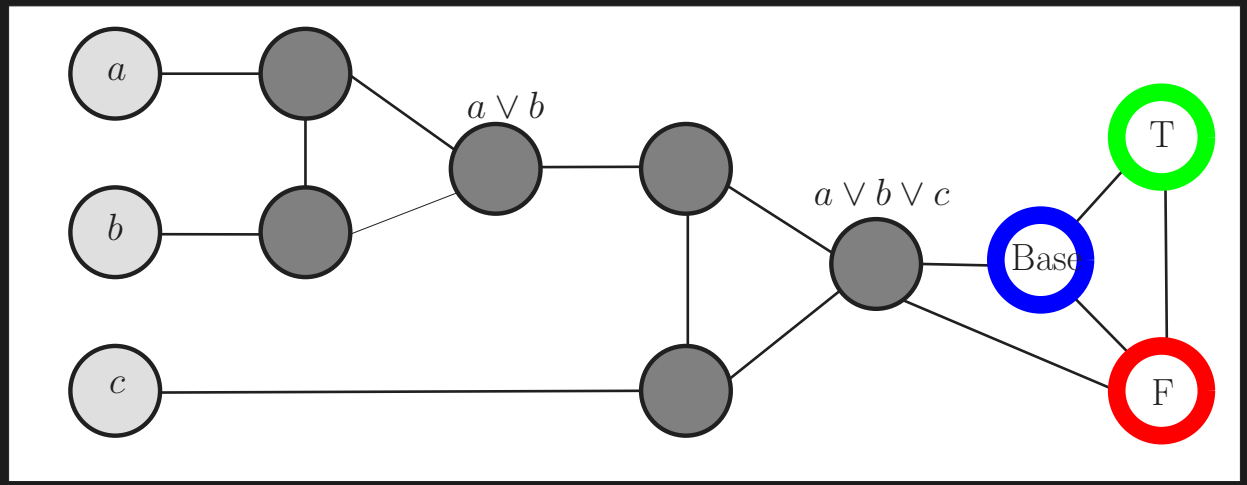
- Property: if a, b, c are colored **false** in a 3-coloring then output node of OR-gadget has to be colored **false**.
- Property: if one of a, b, c is colored **true** then OR-gadget can be 3-colored such that output node of OR-gadget is colored **true**.

OR-Gadget Graph

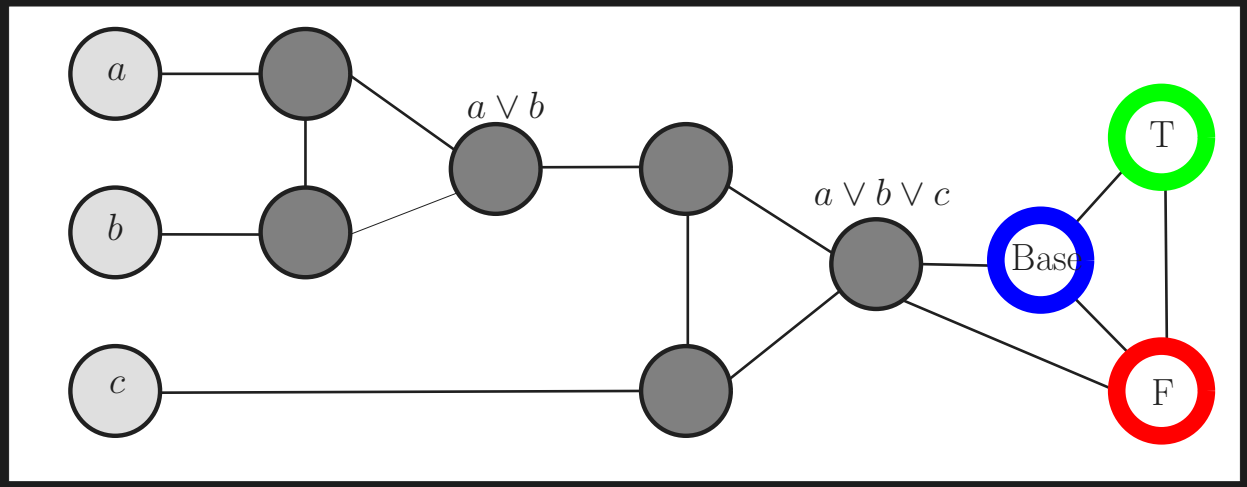
- Property: if a, b, c are colored **false** in a 3-coloring then output node of OR-gadget has to be colored **false**.
- Property: if one of a, b, c is colored **true** then OR-gadget can be 3-colored such that output node of OR-gadget is colored **true**.

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 the above **base** vertex.
- 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**.



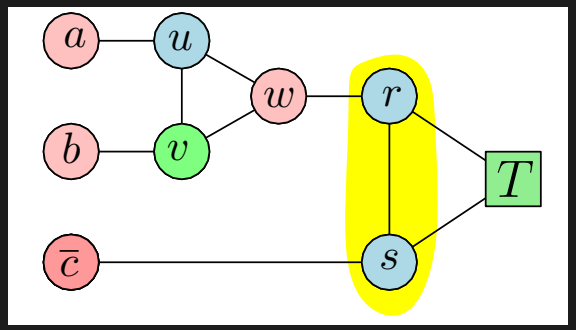
Reduction



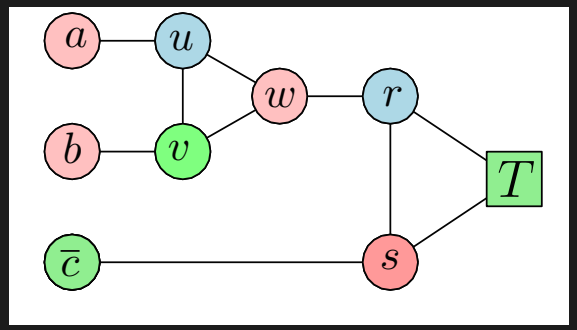
Claim

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.*

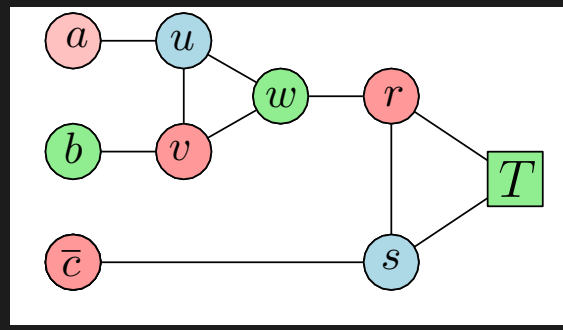
3 coloring of the clause gadget



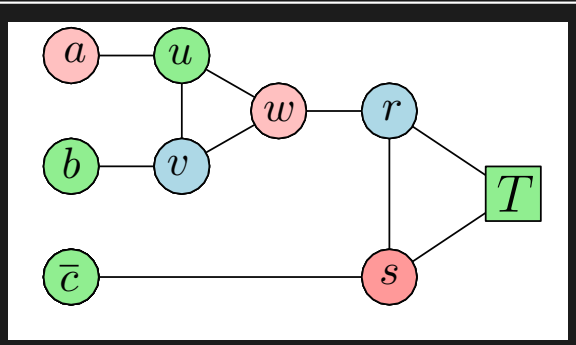
FFF - **BAD**



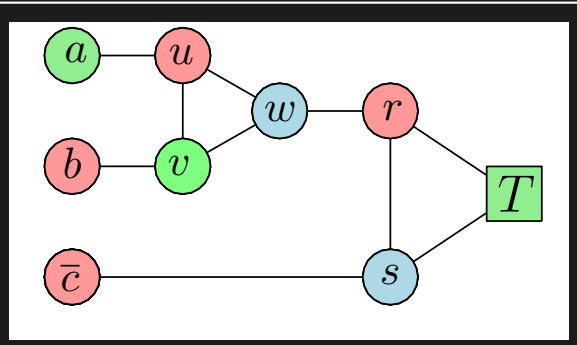
FFT



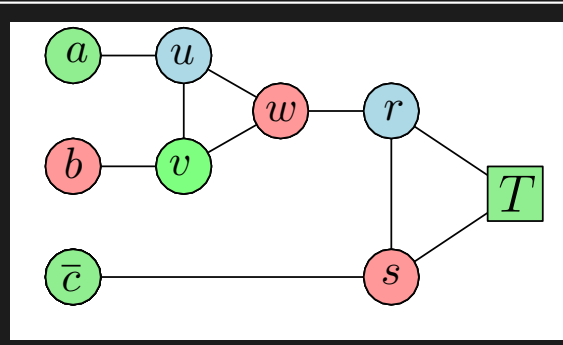
FTF



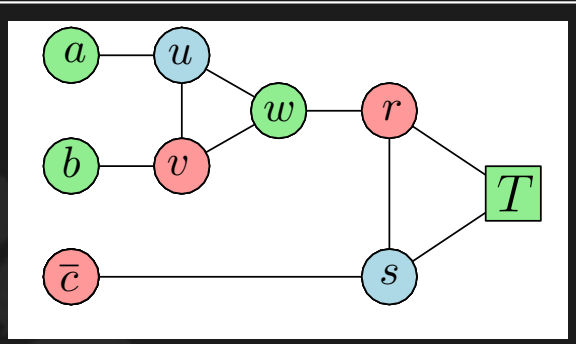
FTT



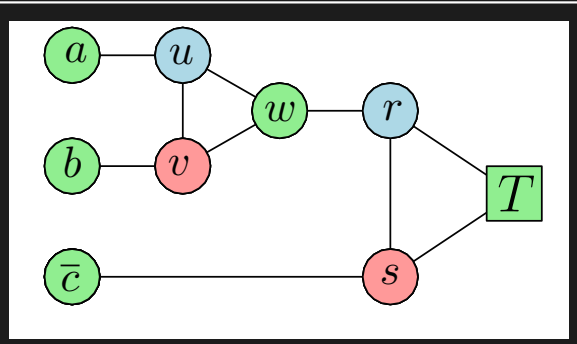
TFF



TFT



TTF

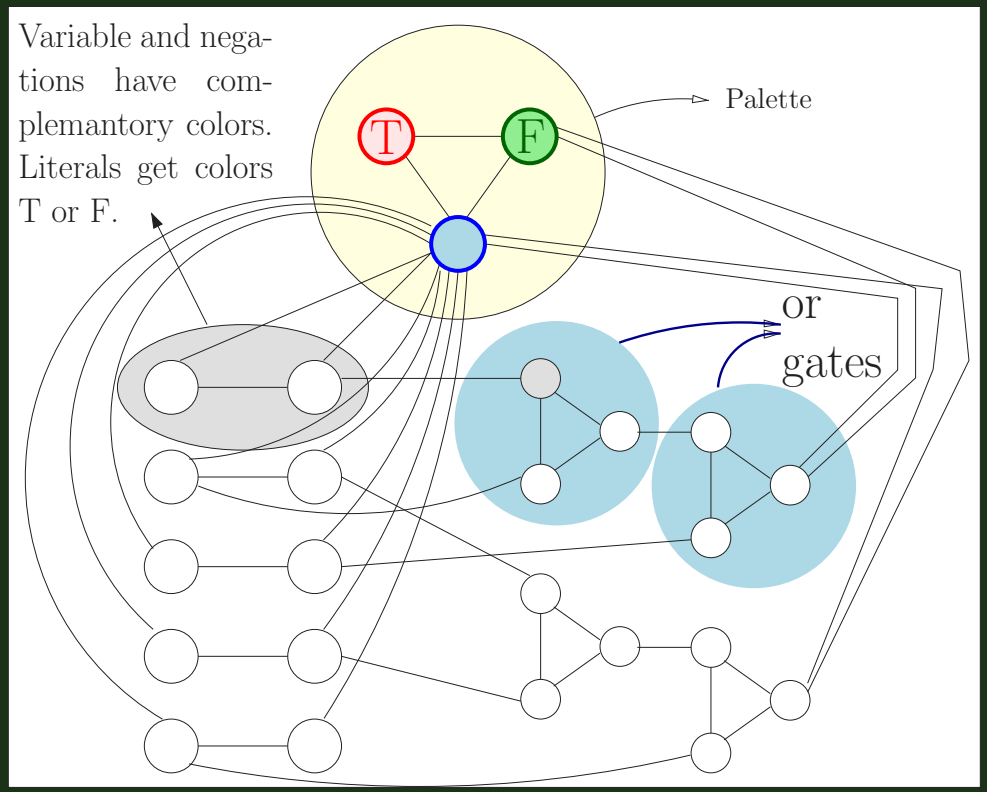


TTT

Reduction Outline

Example

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



Correctness of Reduction

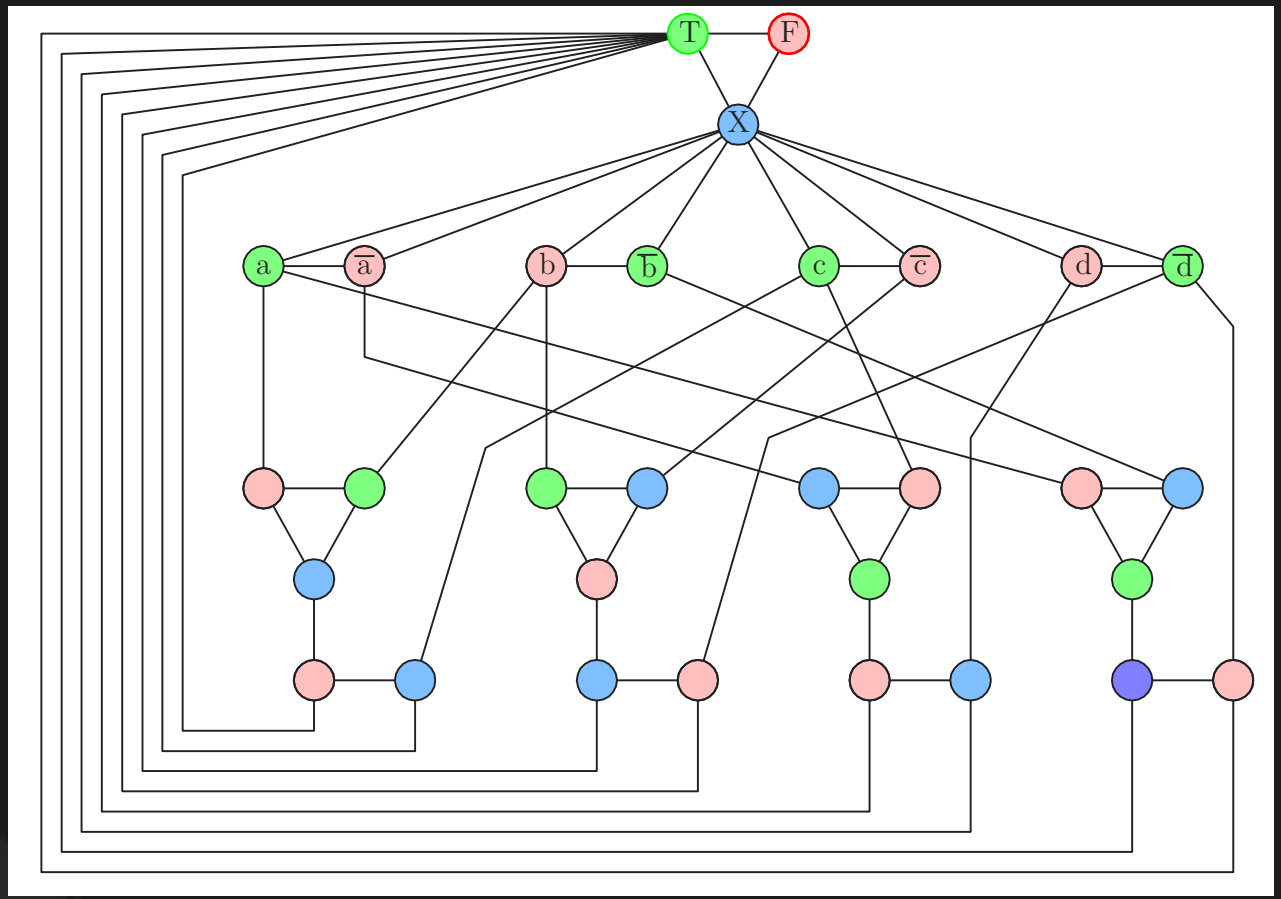
1. φ is satisfiable implies G_φ is 3-colorable
   1. if x_i is assigned 1, color v_i **true** and $\bar{v}_i$ **false**.
   2. for each clause $C_j = (a \vee b \vee c)$ at least one of a, b, c is colored True. OR-gadget for C_j can be 3-colored such that output is True.
2. G_φ is 3-colorable implies φ is satisfiable
   1. If v_i is colored **true** then set x_i to be 1, this is a legal truth assignment.
   2. Consider any clause $C_j = (a \vee b \vee c)$. it cannot be that all a, b, c are all colored **false**. If so, output of OR-gadget for C_j has to be colored **false** but output is connected to **base** and **false**!

Correctness of Reduction

1. φ is satisfiable implies G_φ is 3-colorable
   1. if x_i is assigned 1, color v_i **true** and $\bar{v}_i$ **false**.
   2. 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.
2. G_φ is 3-colorable implies φ is satisfiable
   1. If v_i is colored **true** then set x_i to be 1, this is a legal truth assignment.
   2. Consider any clause $C_j = (a \vee b \vee c)$. it cannot be that all a, b, c are all colored **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**



Part VI

Hardness of **Subset Sum**

Subset Sum

Subset Sum

Instance: S - set of positive integers, t: - an integer number (Target)
Question: Is there a subset $X \subseteq S$ such that $\sum_{x \in X} x = t$?

Claim

Subset Sum *is* **NP-Complete**.

Vector Subset Sum

We will prove following problem is **NP-Complete**...

Vec Subset Sum

Instance: S - set of n vectors of dimension k, each vector has non-negative numbers for its coordinates, and a target vector $\overrightarrow{t}$.
Question: Is there a subset $X \subseteq S$ such that $\sum_{\overrightarrow{x} \in X} \overrightarrow{x} = \overrightarrow{t}$?

Reduction from **3SAT**.

Vector Subset Sum

Handling a single clause

Think about vectors as being lines in a table.

First gadget

Selecting between two lines.

Target	??	??	01	???
a_1	??	??	01	??
a_2	??	??	01	??

Two rows for every variable x: selecting either $x = 0$ or $x = 1$.

Handling a clause...

We will have a column for every clause...

numbers	...	$C \equiv a \vee b \vee \overline{c}$	...
a	...	01	...
$\overline{a}$	...	00	...
b	...	01	...
$\overline{b}$	...	00	...
c	...	00	...
$\overline{c}$	...	01	...
C fix-up 1	000	07	000
C fix-up 2	000	08	000
C fix-up 3	000	09	000
TARGET		10	

3SAT to Vec Subset Sum

numbers	$a \vee \overline{a}$	$b \vee \overline{b}$	$c \vee \overline{c}$	$d \vee \overline{d}$	$D \equiv \overline{b} \vee c \vee \overline{d}$	$C \equiv a \vee b \vee \overline{c}$
a	1	0	0	0	00	01
$\overline{a}$	1	0	0	0	00	00
b	0	1	0	0	00	01
$\overline{b}$	0	1	0	0	01	00
c	0	0	1	0	01	00
$\overline{c}$	0	0	1	0	00	01
d	0	0	0	1	00	00
$\overline{d}$	0	0	0	1	01	01
C fix-up 1	0	0	0	0	00	07
C fix-up 2	0	0	0	0	00	08
C fix-up 3	0	0	0	0	00	09
D fix-up 1	0	0	0	0	07	00
D fix-up 2	0	0	0	0	08	00
D fix-up 3	0	0	0	0	09	00
TARGET	1	1	1	1	10	10

Vec Subset Sum to Subset Sum

numbers
010000000001
010000000000
000100000001
000100000100
000001000100
000001000001
000000010000
000000010101
000000000007
000000000008
000000000009
000000000700
000000000800
000000000900
010101011010

Other NP-Complete Problems

- 3-Dimensional Matching
- Subset Sum

Read book.

Need to Know NP-Complete Problems

- **3SAT**.
- **Circuit-SAT**.
- **Independent Set**.
- **Vertex Cover**.
- **Clique**.
- **Set Cover**.
- **Hamiltonian Cycle** (in Directed/Undirected Graphs).
- **3Coloring**.
- **3-D Matching**.
- **Subset Sum**.

Subset Sum and Knapsack

- Subset Sum Problem: Given n integers $a_1, a_2, \ldots, a_n$ and a target B, is there a subset of S of $\{a_1, \ldots, a_n\}$ such that the numbers in S add up *precisely* to B?
- Subset Sum is **NP-Complete**— see book.
- Knapsack: Given n items with item i having size s_i and profit p_i, a knapsack of capacity B, and a target profit P, is there a subset S of items that can be packed in the knapsack and the profit of S is at least P?
- Show Knapsack problem is **NP-Complete** via reduction from Subset Sum (exercise).

Subset Sum and Knapsack

1. Subset Sum Problem: Given n integers $a_1, a_2, \ldots, a_n$ and a target B, is there a subset of S of $\{a_1, \ldots, a_n\}$ such that the numbers in S add up *precisely* to B?
2. Subset Sum is **NP-Complete**— see book.
3. Knapsack: Given n items with item i having size s_i and profit p_i, a knapsack of capacity B, and a target profit P, is there a subset S of items that can be packed in the knapsack and the profit of S is at least P?
4. Show Knapsack problem is **NP-Complete** via reduction from Subset Sum (exercise).

Subset Sum and Knapsack

1. Subset Sum Problem: Given n integers $a_1, a_2, \ldots, a_n$ and a target B, is there a subset of S of $\{a_1, \ldots, a_n\}$ such that the numbers in S add up *precisely* to B?
2. Subset Sum is **NP-Complete**— see book.
3. Knapsack: Given n items with item i having size s_i and profit p_i, a knapsack of capacity B, and a target profit P, is there a subset S of items that can be packed in the knapsack and the profit of S is at least P?
4. Show Knapsack problem is **NP-Complete** via reduction from Subset Sum (exercise).

Subset Sum and Knapsack

1. Subset Sum Problem: Given n integers $a_1, a_2, \ldots, a_n$ and a target B, is there a subset of S of $\{a_1, \ldots, a_n\}$ such that the numbers in S add up *precisely* to B?
2. Subset Sum is **NP-Complete**— see book.
3. Knapsack: Given n items with item i having size s_i and profit p_i, a knapsack of capacity B, and a target profit P, is there a subset S of items that can be packed in the knapsack and the profit of S is at least P?
4. Show Knapsack problem is **NP-Complete** via reduction from Subset Sum (exercise).

Subset Sum and Knapsack

- Subset Sum can be solved in $O(nB)$ time using dynamic programming (exercise).
- Implies that problem is hard only when numbers $a_1, a_2, \ldots, a_n$ are exponentially large compared to n. That is, each a_i requires polynomial in n bits.
- *Number problems* of the above type are said to be ***weakly*** **NP-Complete**.

Subset Sum and Knapsack

- Subset Sum can be solved in $O(nB)$ time using dynamic programming (exercise).
- Implies that problem is hard only when numbers $a_1, a_2, \ldots, a_n$ are exponentially large compared to n. That is, each a_i requires polynomial in n bits.
- *Number problems* of the above type are said to be ***weakly*** **NP-Complete**.

Subset Sum and Knapsack

1. Subset Sum can be solved in $O(nB)$ time using dynamic programming (exercise).
2. Implies that problem is hard only when numbers $a_1, a_2, \ldots, a_n$ are exponentially large compared to n. That is, each a_i requires polynomial in n bits.
3. *Number problems* of the above type are said to be ***weakly*** **NP-Complete**.

Notes

Notes

Notes

Notes