NP-Completeness, NP-hardness for Optimization

Techniques for reductions, Proof writing guide, NP-hard optimization problems

Last Lecture

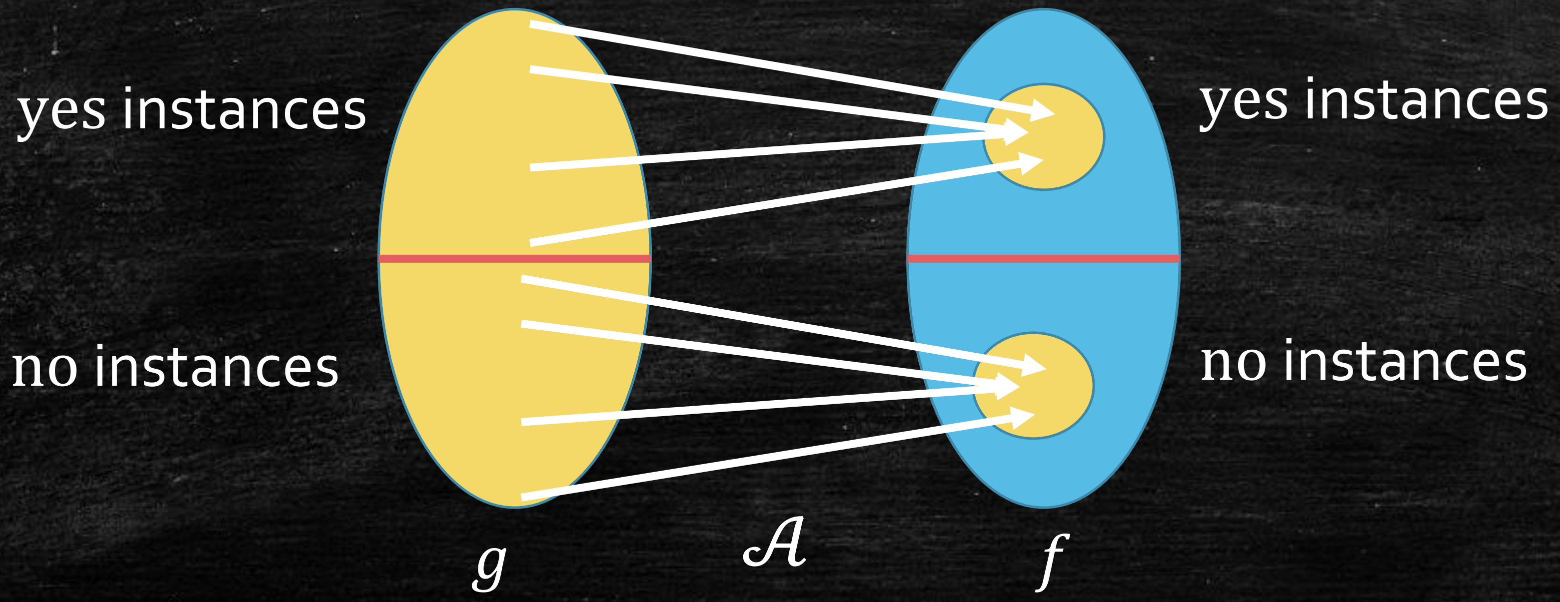
- **P**: decision problems that can be decided efficiently
- **NP**: decision problems that can be verified efficiently
- Reduction is an effective tool to show one problem is "weakly harder" than another.
- NP-Completeness describes the hardest problems in **NP**.
- Cook-Levin Theorem. SAT is NP-complete.
- 3SAT, VertexCover, IndependentSet, SubsetSum, HamiltonianPath are NP-complete.

Proving f is NP-complete

- Prove $f \in$ **NP**.
- Find an NP-complete problem g and prove $g \leq_k f$.

Reduction: $\mathcal{A}$ computes $g \leq_k f$

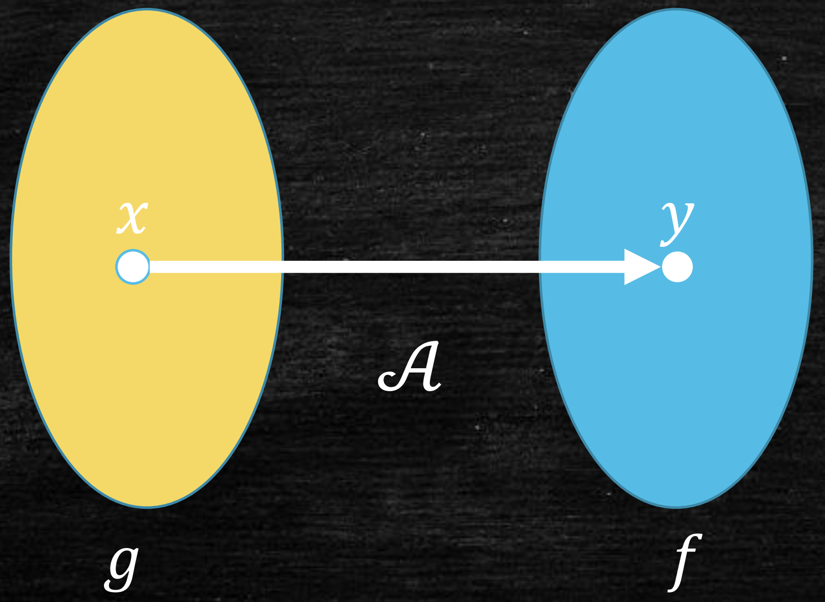
- $x \mapsto y$ under poly-time TM $\mathcal{A}$
- x is yes $\Rightarrow$ y is yes
- x is no $\Rightarrow$ y is no



Reduction: $\mathcal{A}$ computes $g \leq_k f$

- $x \mapsto y$ under poly-time TM $\mathcal{A}$
- x is yes $\Longrightarrow$ y is yes
- x is no $\Longrightarrow$ y is no

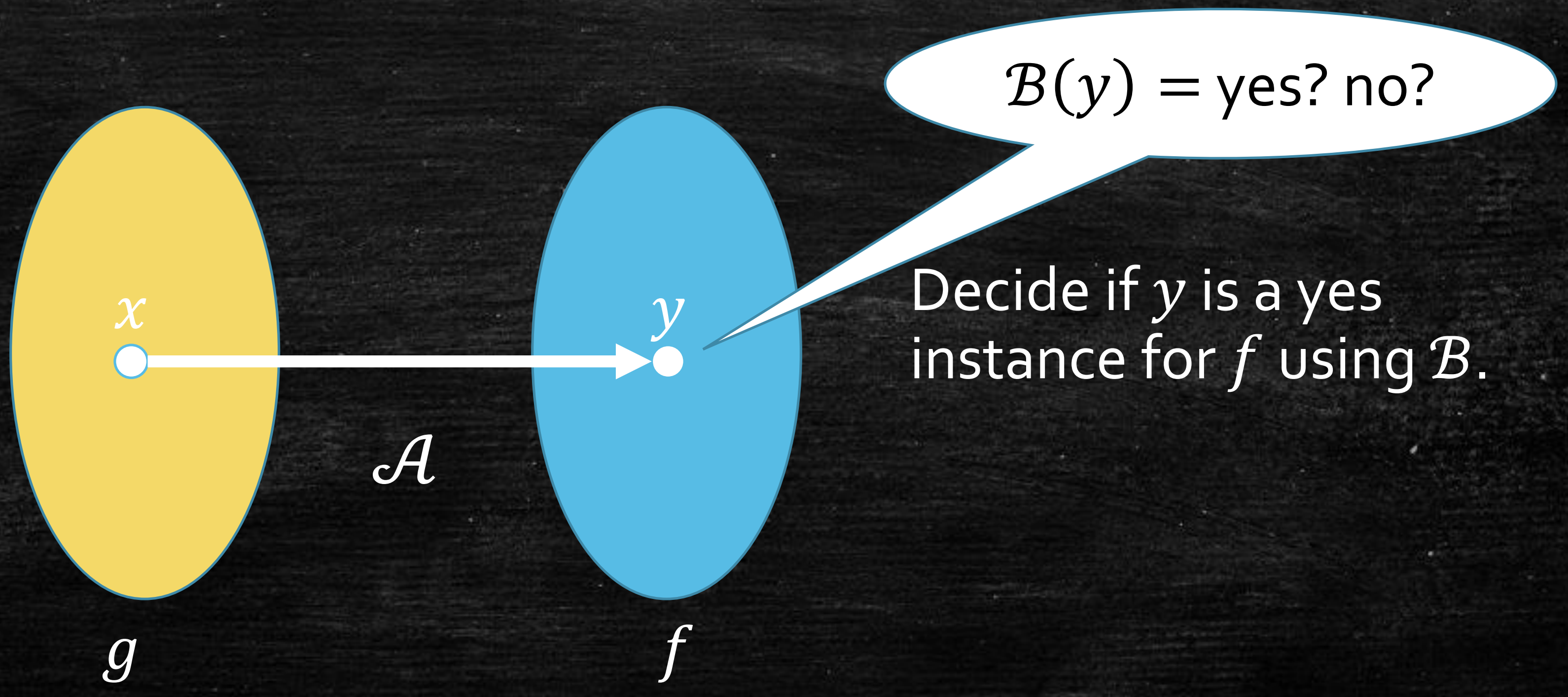
- A poly-time TM $\mathcal{B}$ solving f
- $\Longrightarrow$ The TM $\mathcal{B} \circ \mathcal{A}$ solves g



Given any g instance x,
Compute the f instance
$y = \mathcal{A}(x)$.

Reduction: $\mathcal{A}$ computes $g \leq_k f$

- $x \mapsto y$ under poly-time TM $\mathcal{A}$
- x is yes $\implies$ y is yes
- x is no $\implies$ y is no
- A poly-time TM $\mathcal{B}$ solving f
- $\implies$ The TM $\mathcal{B} \circ \mathcal{A}$ solves g



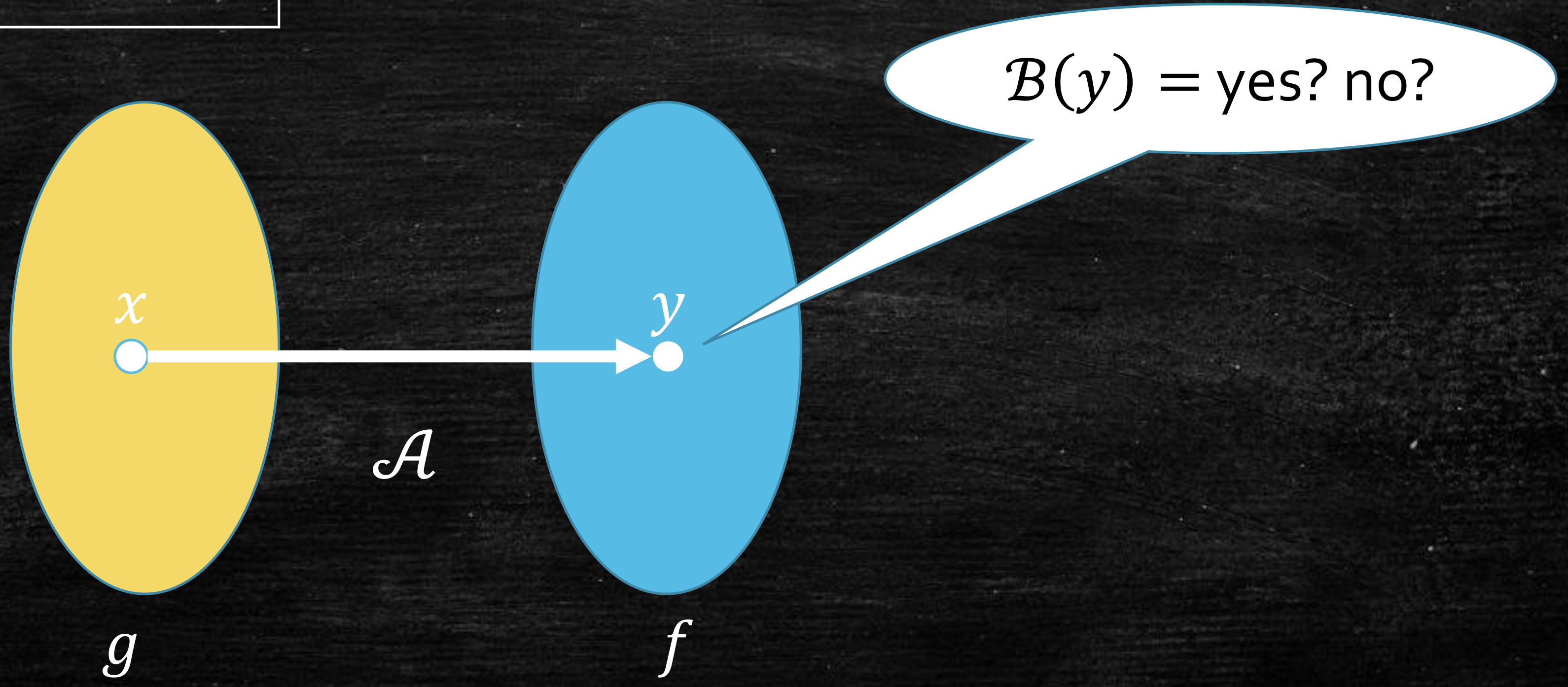
Reduction: $\mathcal{A}$ computes $g \leq_k f$

- $x \mapsto y$ under poly-time TM $\mathcal{A}$
- x is yes $\Longrightarrow$ y is yes
- x is no $\Longrightarrow$ y is no

This is crucial for a reduction to work!

y is yes $\Longrightarrow$ x is yes
y is no $\Longrightarrow$ x is no

- A poly-time TM $\mathcal{B}$ solving f
- $\Longrightarrow$ The TM $\mathcal{B} \circ \mathcal{A}$ solves g



Four Steps for a NP-completeness Proof

1. Prove $f \in \mathbf{NP}$.
2. Construct the reduction $g \leq_k f$.
   - Fix an instance x of g. Describe the corresponding f instance y.
3. [Completeness] x is yes $\Longrightarrow$ y is yes
4. [Soundness] x is no $\Longrightarrow$ y is no
   - Proving the contrapositive "y is yes $\Longrightarrow$ x is yes" is often easier.

This Lecture

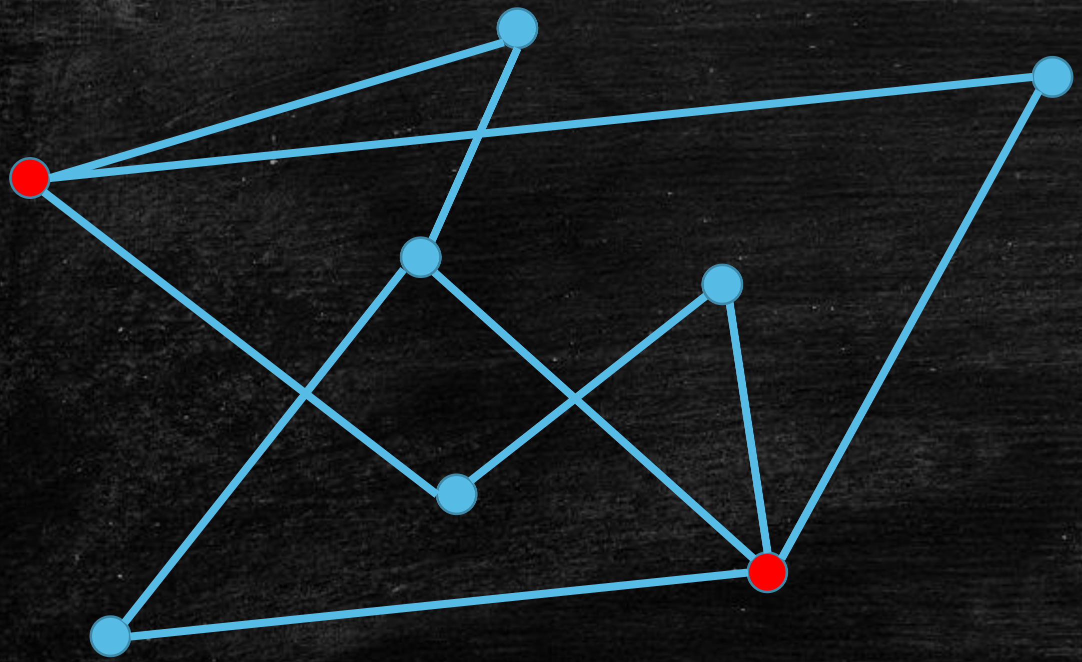
- Show more important NP-complete problems.
- Learn some elementary techniques for reduction.
- Learn how to write a formal proof for NP-completeness.
- NP-hard optimization problems.

Note 1: Choose the Right Problem to Reduce from.

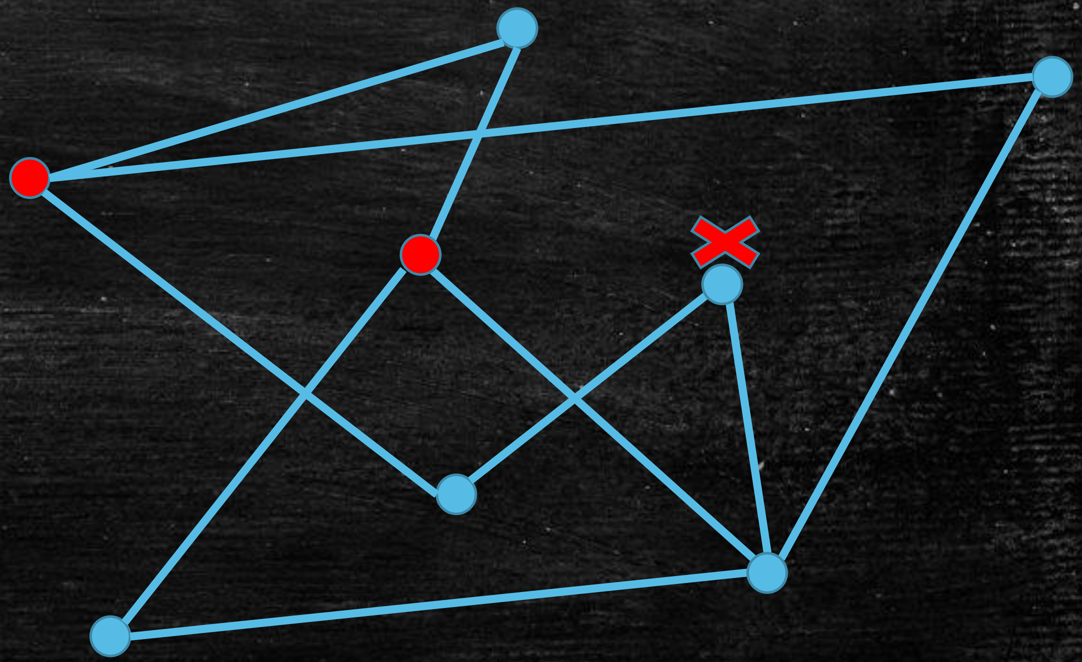
- Want to show an **NP** problem f is NP-complete.
- Need to show $g \leq_k f$ for some NP-complete problem g.
- Conceptually and in principle, $g \leq_k f$ should hold for any NP problem g.
  - Choosing any NP-complete problem should work, e.g., SAT.
- However, choosing a suitable problem makes your life much easier!
- If possible, choose g that "looks similar to" f.

Dominating Set

- Given an undirected graph $G = (V, E)$, a dominating set is a subset of vertices S such that, for any $v \in V \setminus S$, there is a vertex $u \in S$ that is adjacent to v.



a dominating set



not a dominating set

Dominating Set Problem

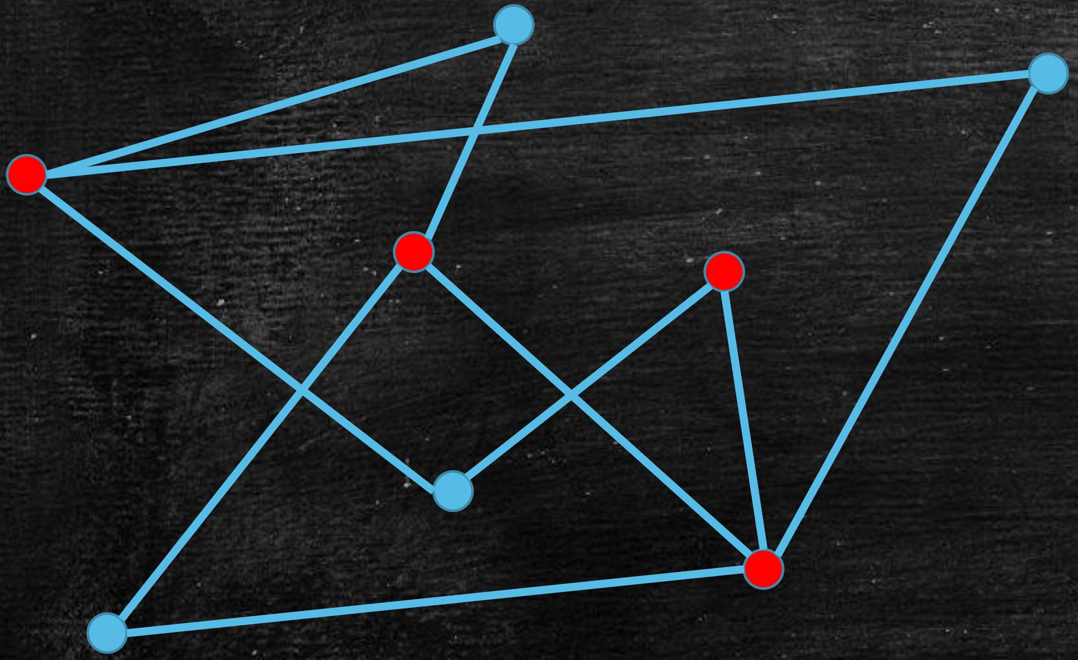
- [DominatingSet] Given an undirected graph $G = (V, E)$ and an integer $k \in \mathbb{Z}^+$, decide if G contains a dominating set with size k.
- Problem: Show that DominatingSet is NP-complete.
- Question: Which problem should we reduce from?

Reduction from VertexCover

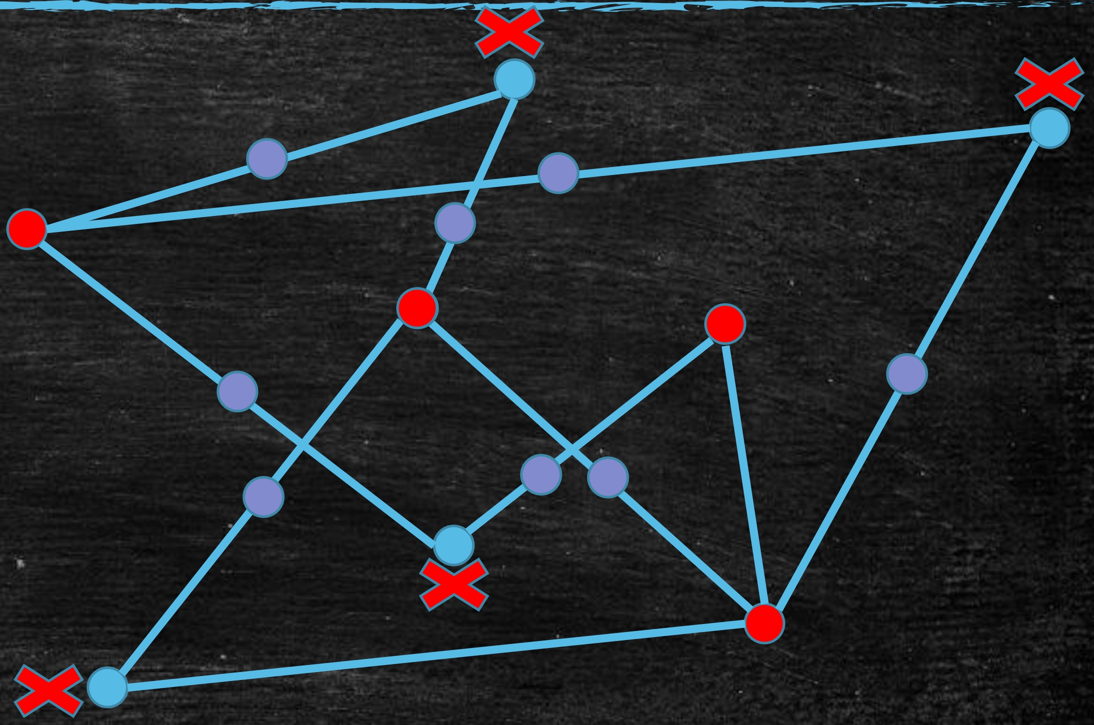
- A dominating set is similar to a vertex cover:
  - Vertex cover: S covers edges
  - Dominating set: S covers vertices
- An idea for reduction:
  - Introduce an intermediate vertex for each edge
  - cover the edge $\Longrightarrow$ cover the intermediate vertex

Does it work?

Does it work? NO!



a vertex cover

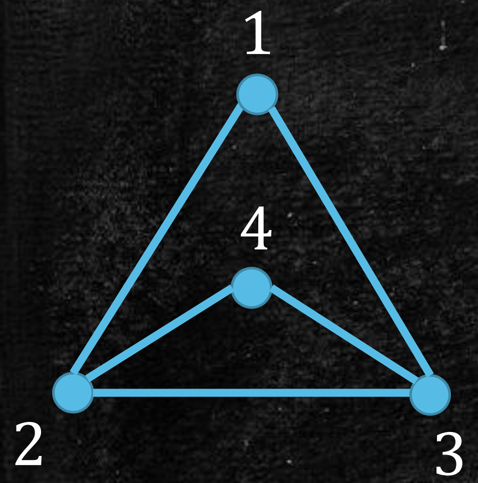


not a dominating set

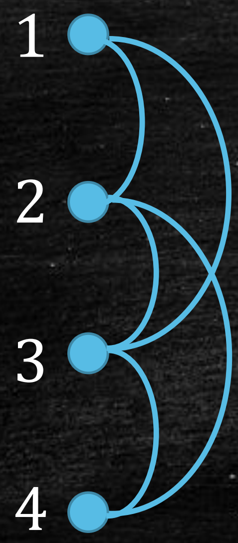
- New vertices are covered, but original vertices may not be covered!
- Can we fix it?

Note 2: Fix your reduction if it doesn't work.

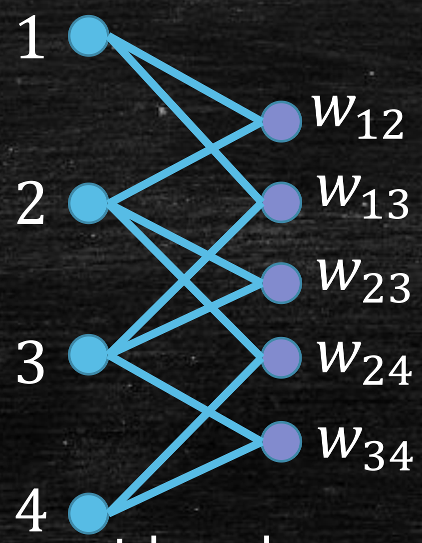
- All we have to do: make the original vertices a clique!
- Now, selecting a single vertex in the original vertex set covers all the original vertices.



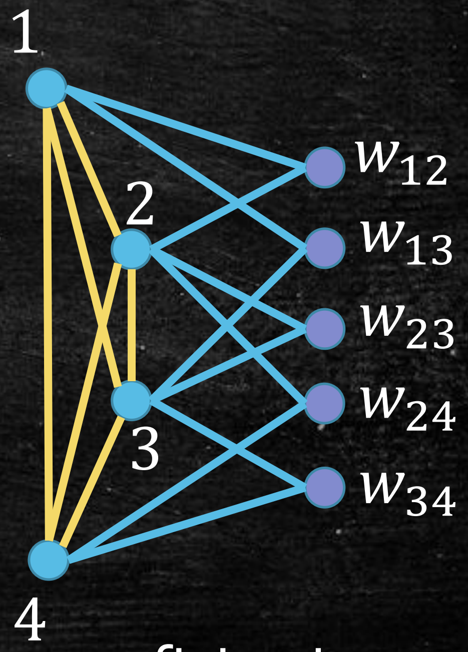
original graph



redraw it



our idea that doesn't work



fixing it

How to write a NP-Completeness Proof

Four Parts for proving f is NP-complete:

1. Prove that f is in **NP**
2. Present the reduction $g \leq_k f$ for an NP-complete problem g
3. Show that yes instances of g are mapped to yes instances of f
4. Show that no instances of g are mapped to no instances of f
   - Most of the time, it is easier to prove its contrapositive: if an instance x of g is mapped to a yes instance of f, then x is a yes instance of g.

DominatingSet is NP-complete – a formal proof

Proof. First of all, DominatingSet is in **NP**, as a dominating set S can be served as a certificate, and it can be verified in polynomial time whether S is a dominating set and whether $|S| = k$.

To show that DominatingSet is NP-complete, we present a reduction from VertexCover. Given a VertexCover instance $(G = (V, E), k)$, we construct a DominatingSet instance $(G' = (V', E'), k')$ as follows.

The vertex set is $V' = \overline{V} \cup \overline{E}$, which is defined as follows. For each vertex $v \in V$ in the VertexCover instance, construct a vertex $\overline{v} \in \overline{V} \subseteq V'$; for each edge $e \in E$ in the VertexCover instance, construct a vertex $w_e \in \overline{E} \subseteq V'$.

The edge set E' is defined as follows. For each edge $e = (u, v)$ in the VertexCover instance, build two edges $(\overline{u}, w_e), (\overline{v}, w_e) \in E'$. For any two vertices $\overline{u}, \overline{v}$ in $\overline{V}$, build an edge $(\overline{u}, \overline{v})$.

Define $k' = k$.

DominatingSet is NP-complete – a formal proof (continued)

Proof (Continued).

Suppose $(G = (V, E), k)$ is a yes VertexCover instance. There exists a vertex cover $S \subseteq V$ with $|S| = k$. We will prove $\overline{S}$ corresponding S is a dominating set in G'.

For each vertex in $\overline{V}$, it is covered by any vertex in $\overline{S}$ as $\overline{V}$ forms a clique.

For each vertex w_e in $\overline{E}$, let $e = (u, v) \in E$ be the corresponding edge in the VertexCover instance. We have either $u \in S$ or $v \in S$ (or both), as S is a vertex cover. This implies either $\overline{u} \in \overline{S}$ or $\overline{v} \in \overline{S}$ (or both), which further implies w_e is covered as $(\overline{u}, w_e), (\overline{v}, w_e) \in \overline{E}$ by our construction.

Since $|\overline{S}| = |S| = k = k'$, the DominatingSet instance we constructed is a yes instance.

DominatingSet is NP-complete – a formal proof (continued)

Suppose $(G' = (V', E'), k')$ is a yes DominatingSet instance. There exists a dominating set $S' \subseteq V' = \overline{V} \cup \overline{E}$ with $|S'| = k' = k$. We aim to show that $(G = (V, E), k)$ is a yes VertexCover instance.

First of all, we can assume $S' \subseteq \overline{V}$ without loss of generality. If we have $w_e \in S'$ for some $w_e \in \overline{E}$, we can replace w_e with either $\overline{u}$ or $\overline{v}$ for the edge $e = (u, v)$ in the VertexCover instance. (In the case $\overline{u}$ and $\overline{v}$ have already been included in S', we can replace w_e with any unpicked vertex in $\overline{V}$.) It is easy to see that S' is still a dominating set after the change, as the set of vertices covered by either $\overline{u}$ or $\overline{v}$ is a superset of the set of vertices covered by w_e (which is just $\{\overline{u}, \overline{v}\}$).

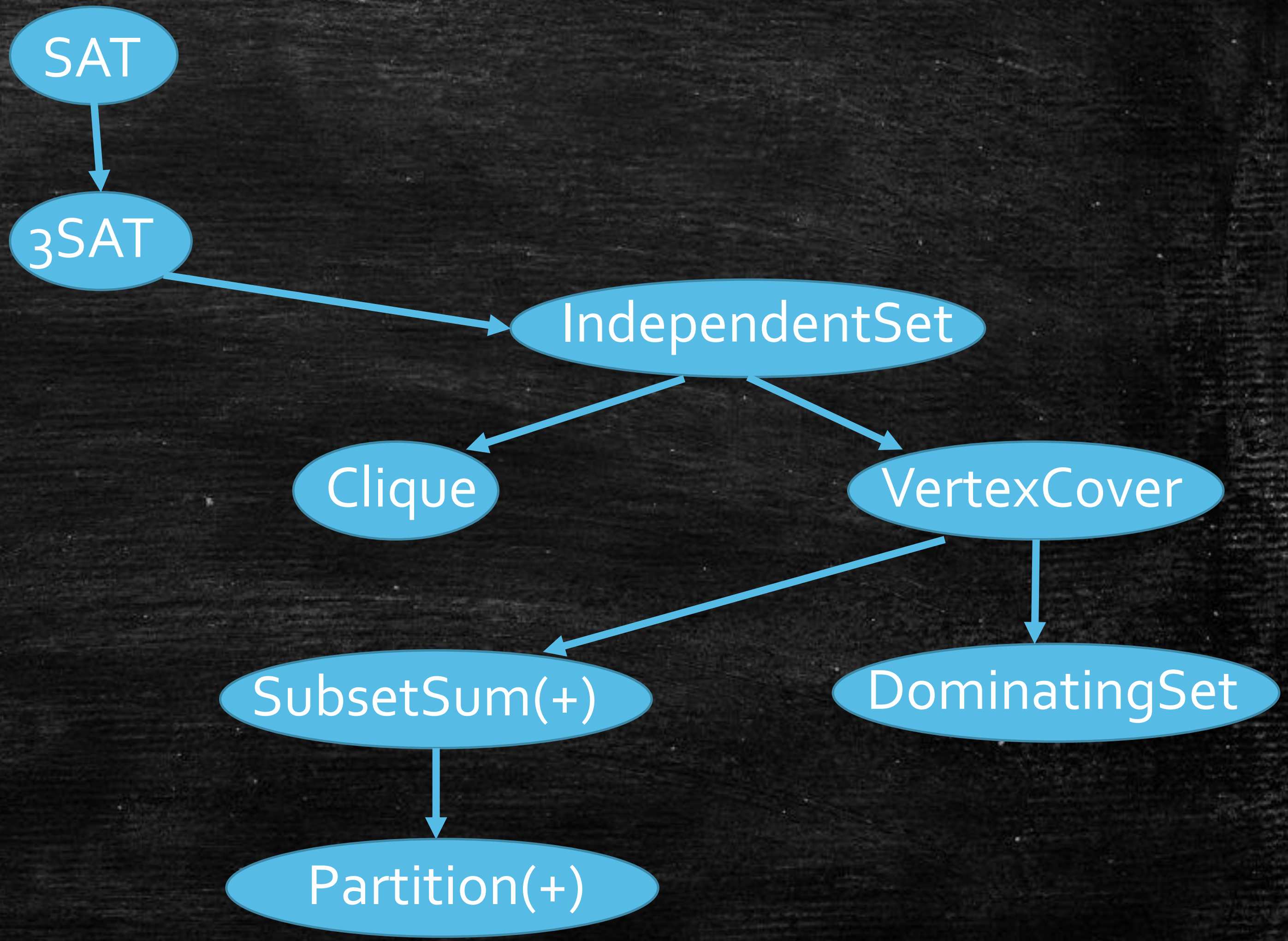
Next, since $S' \subseteq \overline{V}$, S' corresponds to a vertex set $S \subseteq V$ in the VertexCover instance with $|S| = |S'| = k' = k$. It remains to show S is a vertex cover.

For any edge $e = (u, v)$, we have either $\overline{u} \in S'$ or $\overline{v} \in S'$ (or both) since S' is a dominating set and $\overline{u}, \overline{v}$ are the only two vertices that can cover w_e. This implies $u \in S$ or $v \in S$ (or both), so S is a vertex cover.

Some Additional Notes

- Note 3: To prove a no instance is mapped to a no instance, we often prove the contrapositive.
- Note 4: When proving the above-mentioned contrapositive for $g \leq_k f$, a common technique is to show that we can assume the yes instance of f is "well-behaved" that corresponds to the yes instance of g.
  – E.g., we prove that we can assume $S' \subseteq \overline{V}$ just now.
- Note 5: Do not mess up with the direction: a common mistake is to construct a instance of g from f, which only shows $f \leq_k g$ (which is not helpful).

Web of NP-complete Problems



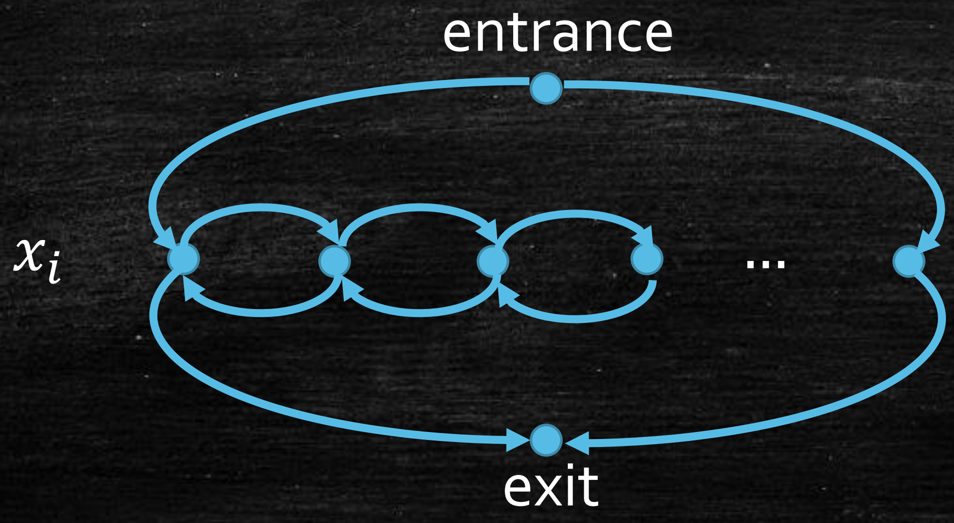
HamiltonianPath is NP-complete

- We have seen HamiltonianPath ∈ **NP**. It remains to show its NP-hardness.
- Intermediate problem: DirectedHamiltonianPath
  - [DirectedHamiltonianPath] Given a directed graph $G = (V, E)$, a source $s \in V$ and a sink $t \in V$, decide if there is a Hamiltonian path from s to t.
- We will show:
  1. $\text{3SAT} \leq_k \text{DirectedHamiltonianPath}$
  2. $\text{DirectedHamiltonianPath} \leq_k \text{HamiltonianPath}$

Note 7: constructing "gadgets" – be creative!

$3\mathrm{SAT} \leq_k \mathrm{DirectedHamiltonianPath}$

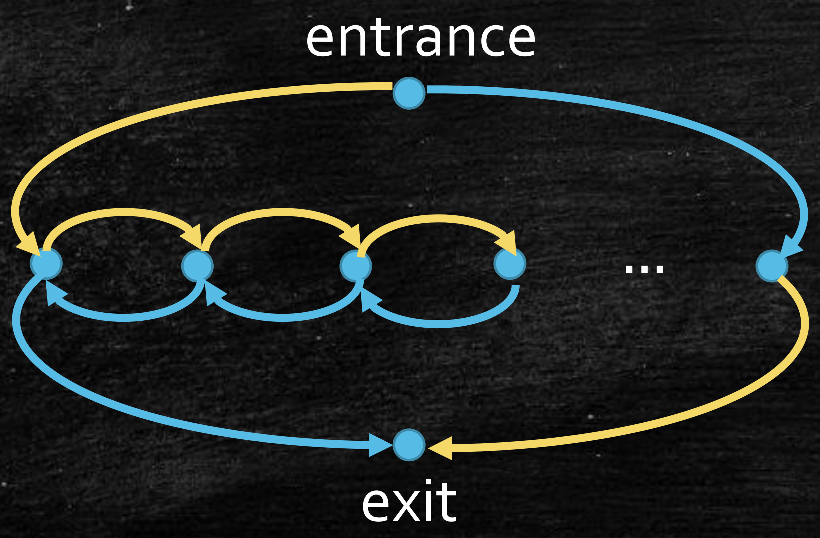
- Given a $3\mathrm{SAT}$ instance ϕ, we will construct a $\mathrm{DirectedHamiltonianPath}$ instance.
- Let n and m be the number of variables and clauses respectively.
- "Variable Gadget"



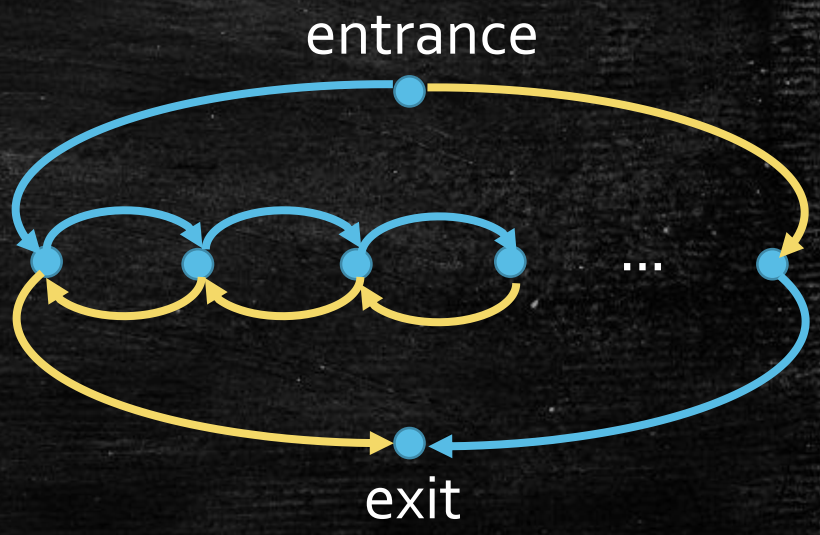
$3SAT \leq_k \text{DirectedHamiltonianPath}$

- There are two ways to go from "entrance" to "exit" that visit the middle vertices.
- They will represent $x_i = \text{true}$ and $x_i = \text{false}$ respectively.

$x_i = \text{true}$

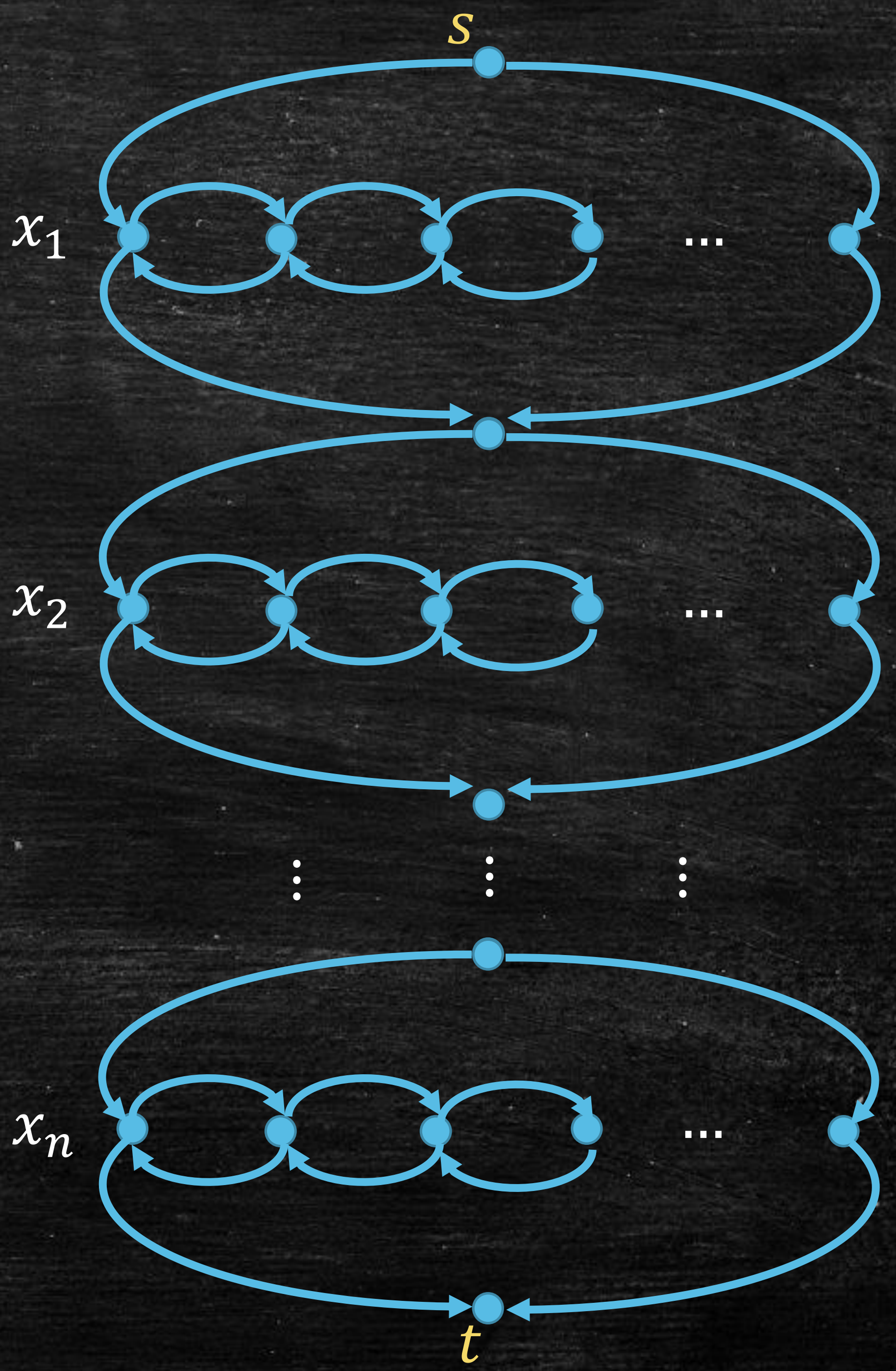


$x_i = \text{false}$



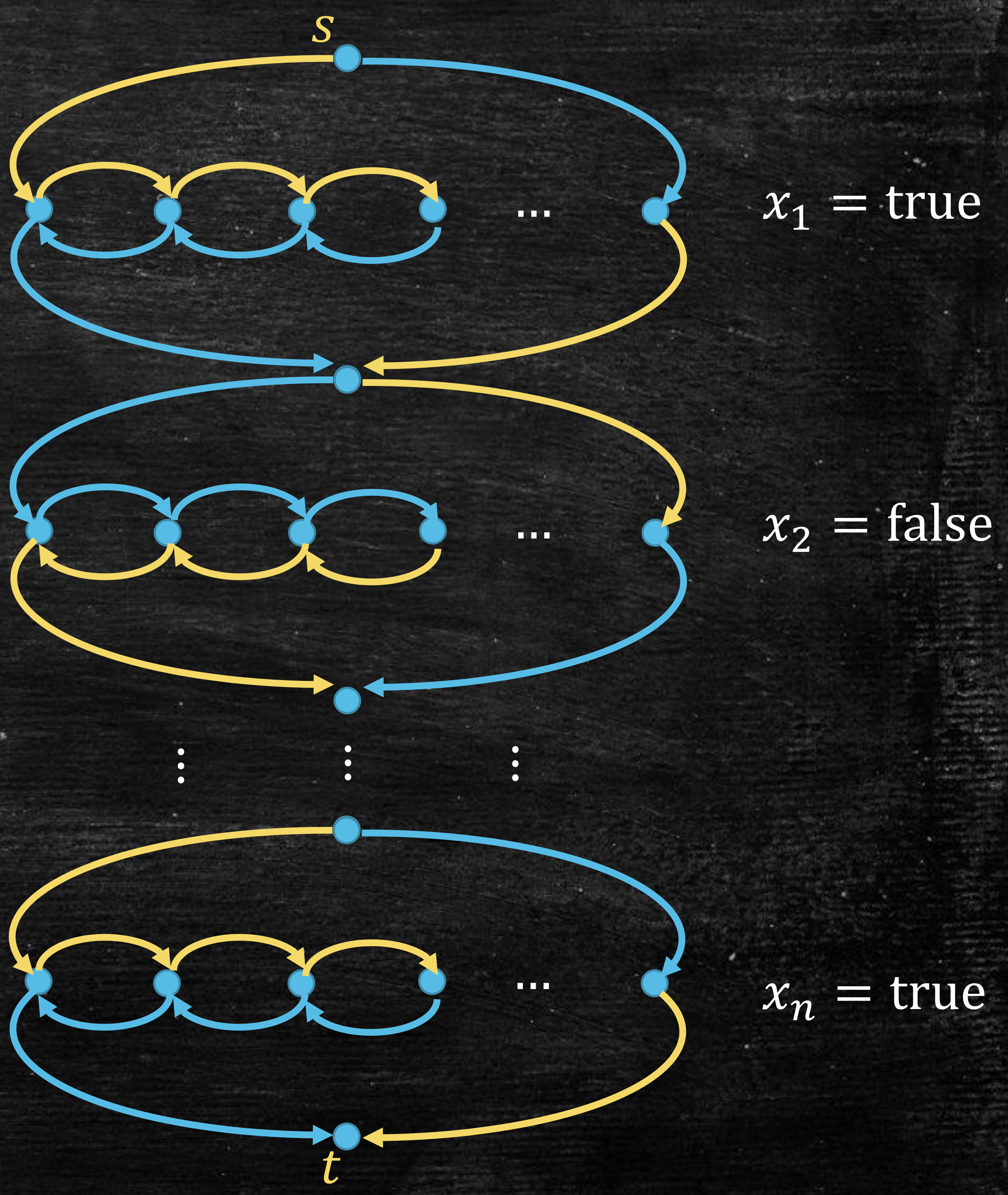
$3\mathrm{SAT} \leq_k \mathrm{DirectedHamiltonianPath}$

- Connect all the variable gadgets.



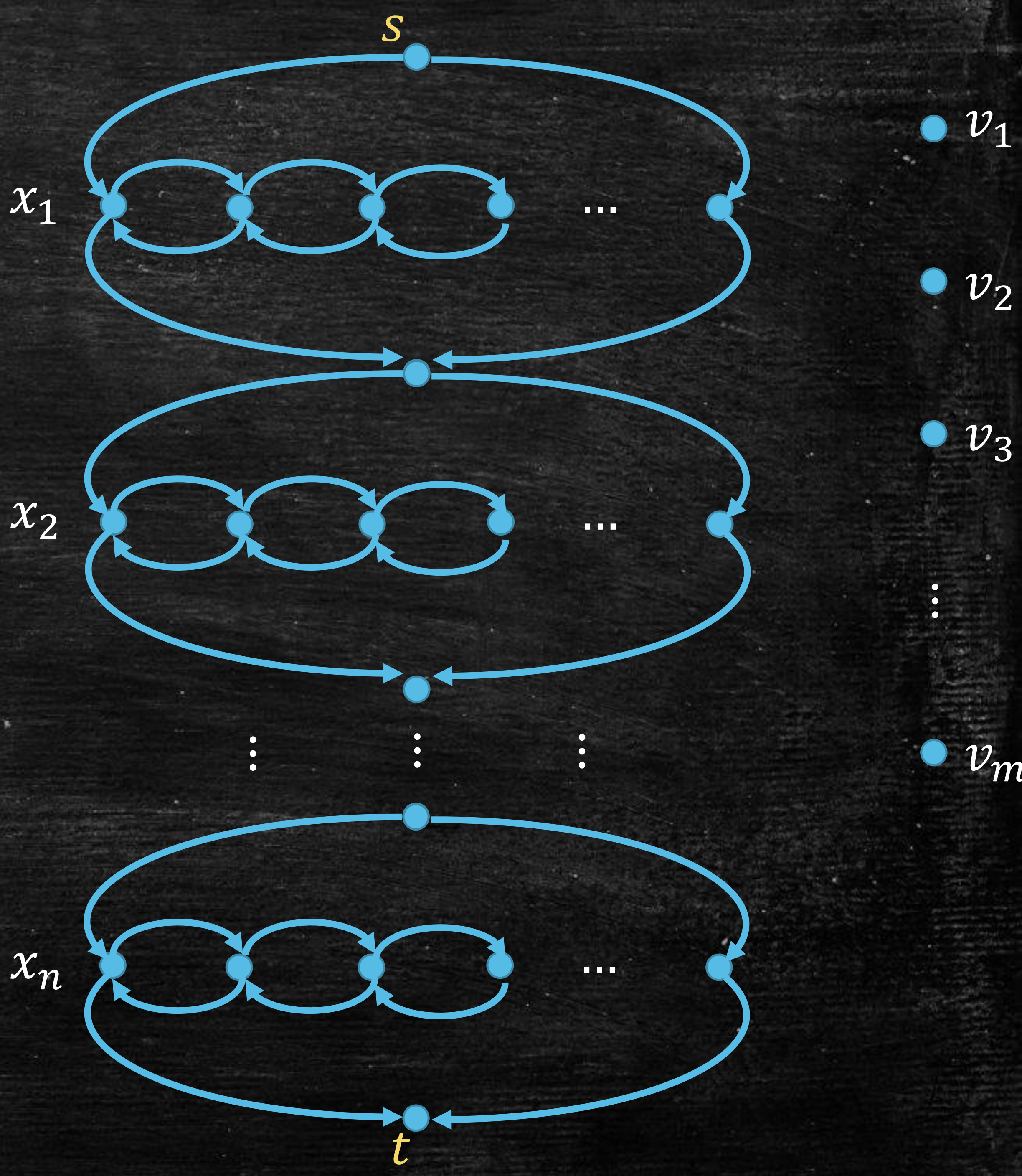
$3SAT \leq_k$ DirectedHamiltonianPath

- Connect all the variable gadgets.
- An s-t simple path visiting all middle vertices corresponds to an assignment to all variables.



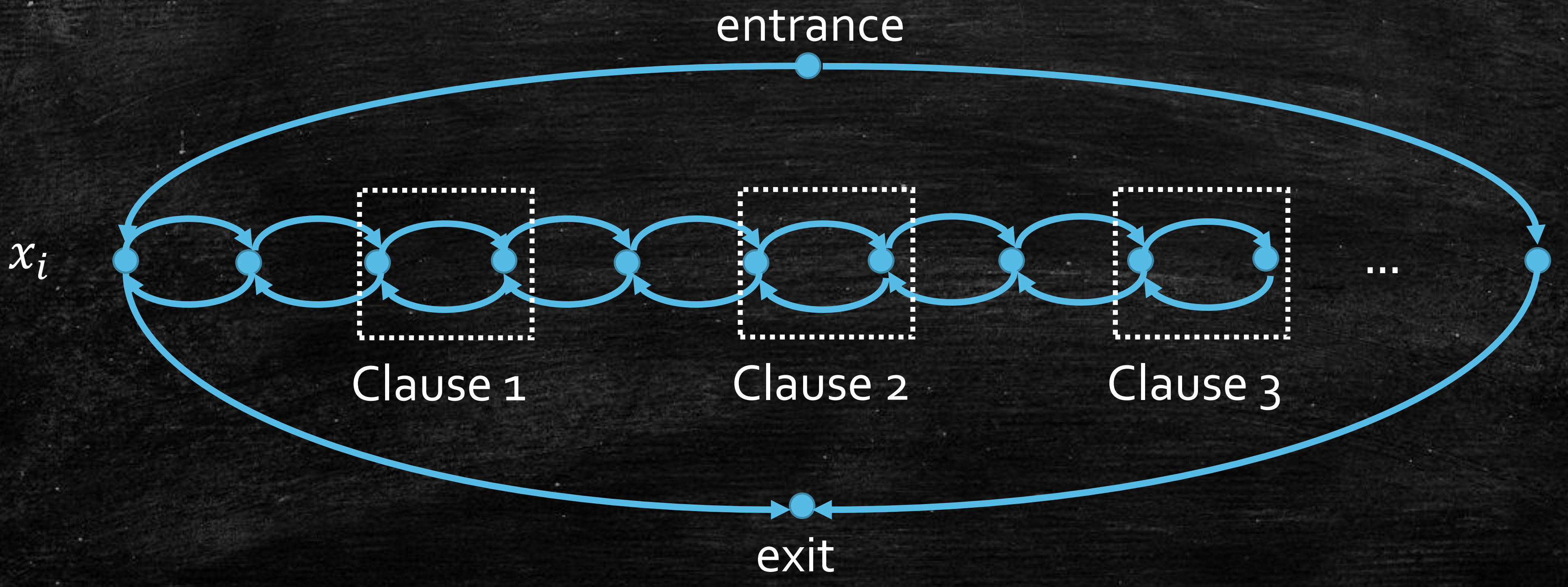
3SAT $\leq_k$ DirectedHamiltonianPath

- Connect all the variable gadgets.
- An s-t simple path visiting all middle vertices corresponds to an assignment to all variables.
- Build a vertex v_j for each clause j.



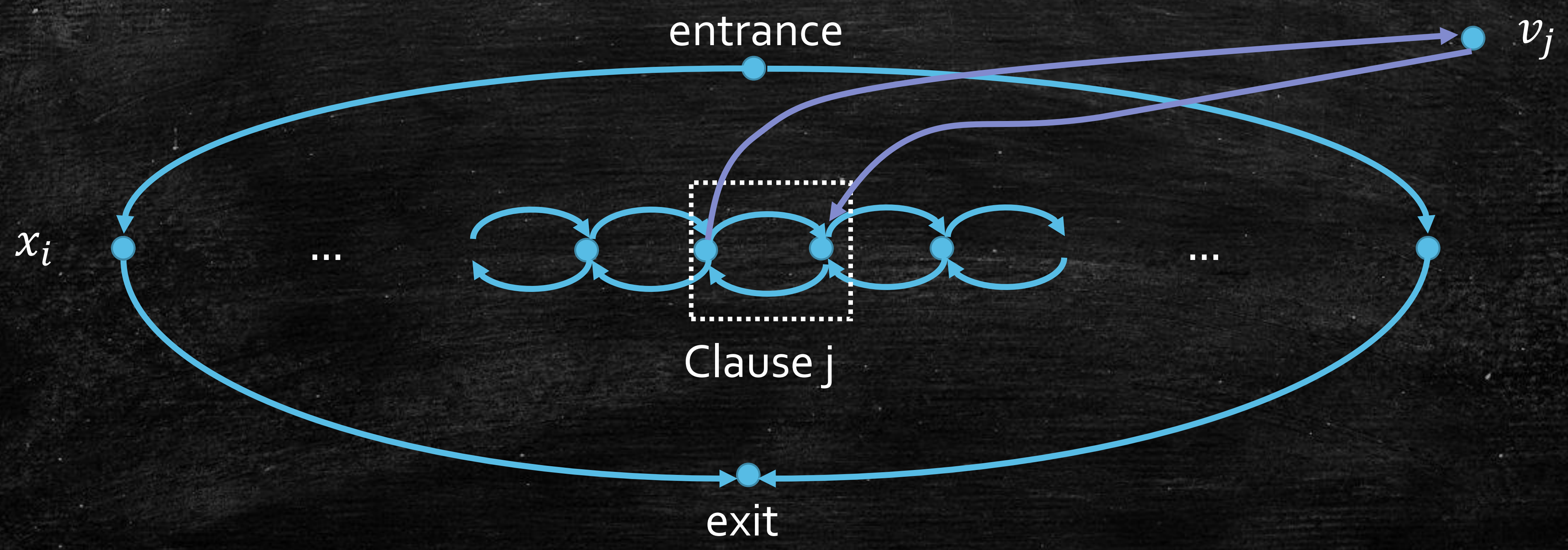
$3SAT \le_k$ DirectedHamiltonianPath

- Inside the variable gadget, build $3m + 1$ middle vertices such that every two vertices corresponds to a clause separated by a "separator".



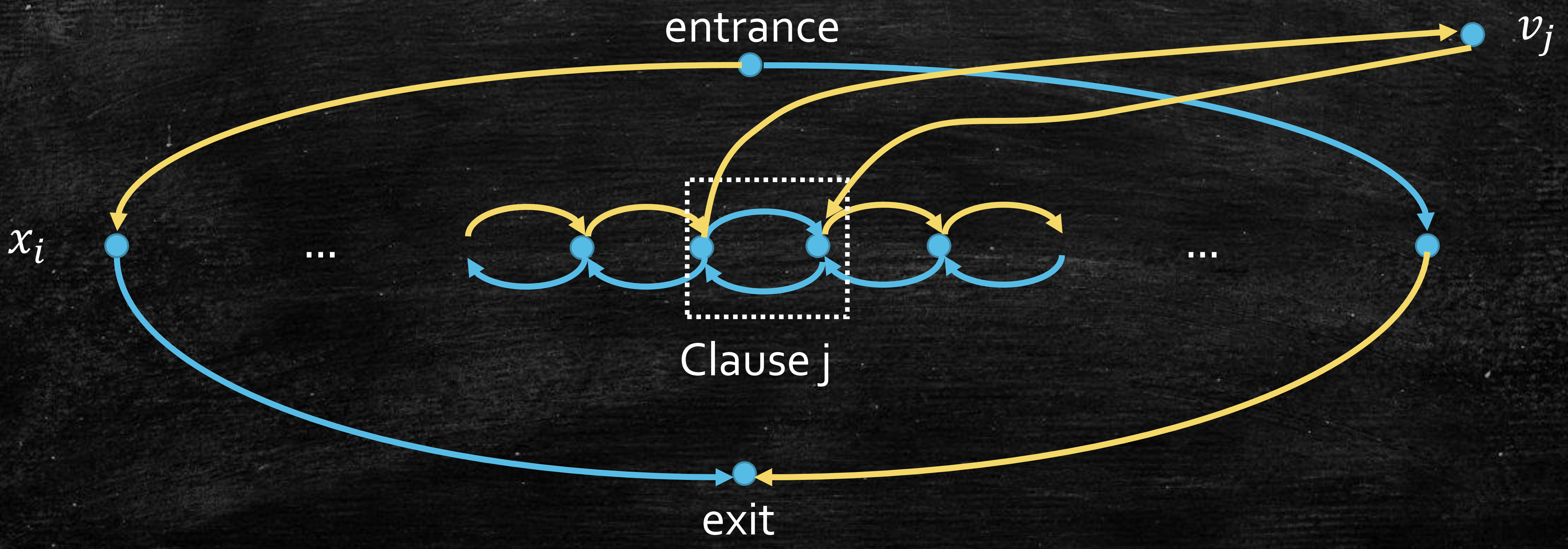
$3SAT \leq_k$ DirectedHamiltonianPath

- If x_i is in j-th clause, connect the gadget to v_j as follows.



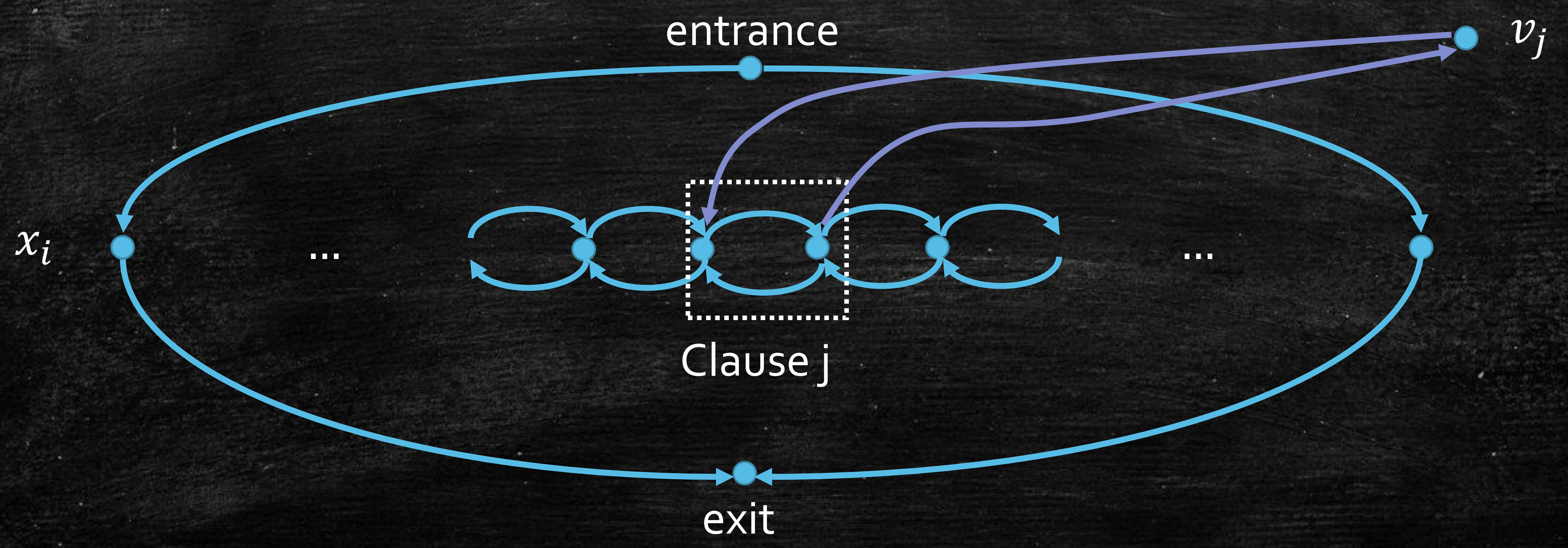
$3\mathrm{SAT} \leq_k \mathrm{DirectedHamiltonianPath}$

- If x_i is in j-th clause, connect the gadget to v_j as follows.
- If $x_i = \mathrm{true}$, j-th clause is satisfied, we can take a detour and visit v_j.



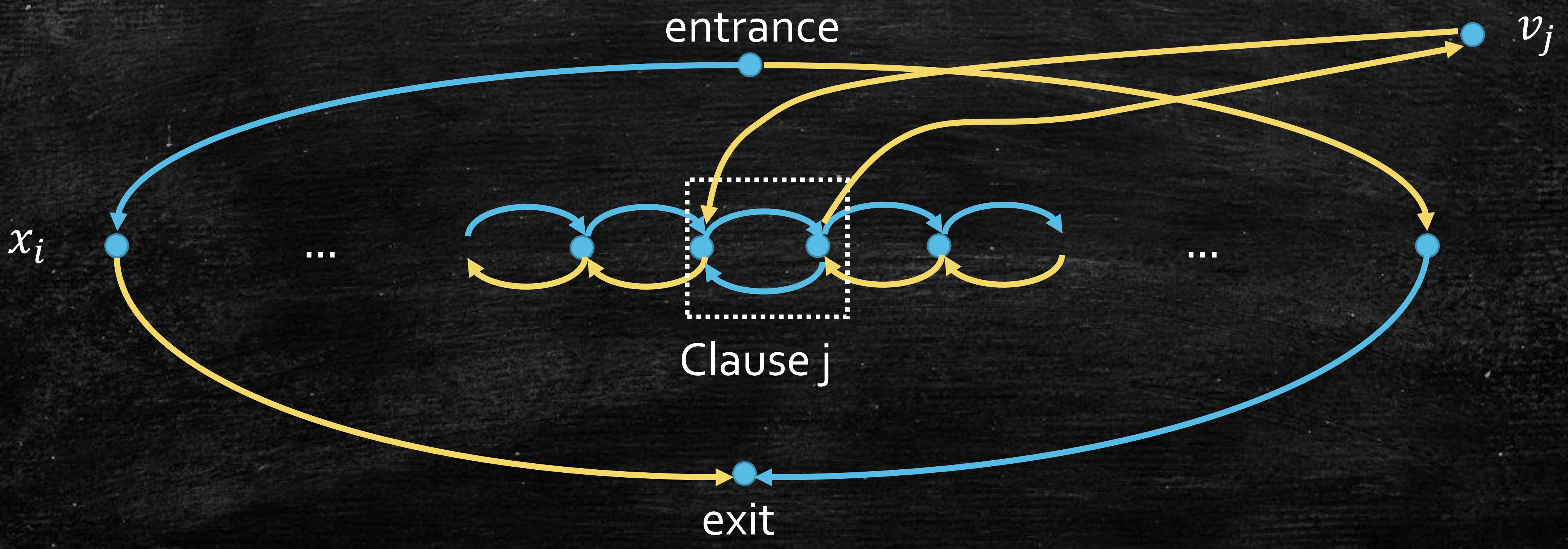
$3SAT \leq_k$ DirectedHamiltonianPath

- If $\neg x_i$ is in j-th clause, connect the gadget to v_j as follows.



$3\text{SAT} \leq_k \text{DirectedHamiltonianPath}$

- If $\neg x_i$ is in j-th clause, connect the gadget to v_j as follows.
- If $x_i = \text{false}$, j-th clause is satisfied, we can take a detour and visit v_j.



$3SAT \leq_k$ DirectedHamiltonianPath

If ϕ is a yes instance, the graph has a Hamiltonian path:

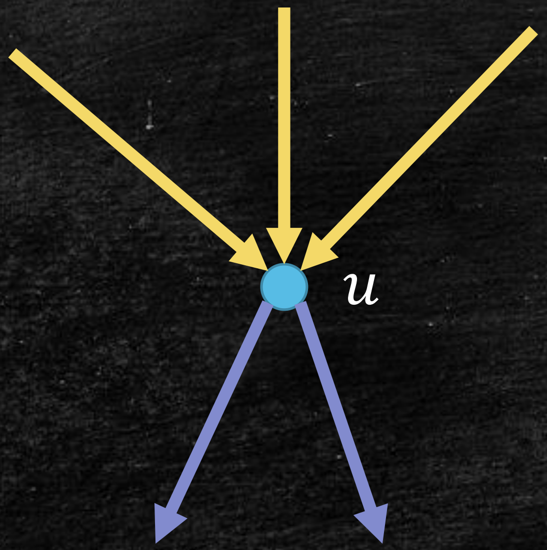
- For each clause, choose a representative true literature.
- Go from s to t, and visit each v_j from its representative by taking a detour.

If the graph has a Hamiltonian path, ϕ is a yes instance:

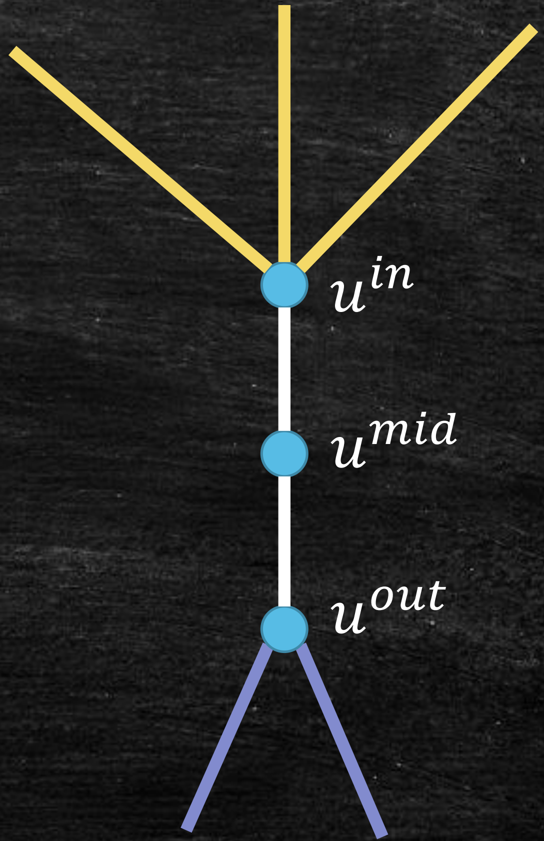
- The Hamiltonian path has to go from s to t.
- Each v_j has to be visited by a detour from a variable.
- The variable's value is then determined.

DirectedHamiltonianPath $\leq_k$ HamiltonianPath

- Vertex Gadget:



a vertex and its incident edges
DirectedHamiltonianPath instance



a vertex gadget and its incident edges
HamiltonianPath instance

DirectedHamiltonianPath $\leq_k$ HamiltonianPath

If G is a yes DirectedHamiltonianPath instance, G' is a yes HamiltonianPath instance:

- Hamiltonian path in G: $s \to u_1 \to u_2 \to \cdots \to u_n \to t$
- Hamiltonian path in G': $s^{in} \to s^{mid} \to s^{out} \to u_1^{in} \to u_1^{mid} \to u_1^{out} \to u_2^{in} \to \cdots \to u_n^{out} \to t^{in} \to t^{mid} \to t^{out}$

$\text{DirectedHamiltonianPath} \leq_k \text{HamiltonianPath}$

If G' is a yes HamiltonianPath instance, G is a yes $\text{DirectedHamiltonianPath}$ instance:

Show that the yes HamiltonianPath instance is "well-behaved"

- **Lemma 1**. The path in G' must start at s^{in} and end at t^{out}.

$\text{DirectedHamiltonianPath} \leq_k \text{HamiltonianPath}$

If G' is a yes HamiltonianPath instance, G is a yes $\text{DirectedHamiltonianPath}$ instance:

Show that the yes HamiltonianPath instance is "well-behaved"

- **Lemma 1**. The path in G' must start at s^{in} and end at t^{out}.
- Proof. s^{in} and t^{out} have degree 1, so they must be starting and ending vertices.
- We can assume the path goes from s^{in} to t^{out}
  - Going from t^{out} to s^{in} is equivalent, as the graph is undirected.

$\text{DirectedHamiltonianPath} \leq_k \text{HamiltonianPath}$

If G' is a yes HamiltonianPath instance, G is a yes $\text{DirectedHamiltonianPath}$ instance:

Show that the yes HamiltonianPath instance is "well-behaved"

- **Lemma 1**. The path in G' must start at s^{in} and end at t^{out}.
- **Lemma 2**. If we first enter a vertex gadget at u^{in} (or u^{out}) we must proceed to u^{mid} and then to u^{out} (or u^{in}).

$\text{DirectedHamiltonianPath} \leq_k \text{HamiltonianPath}$

If G' is a yes HamiltonianPath instance, G is a yes $\text{DirectedHamiltonianPath}$ instance:

Show that the yes HamiltonianPath instance is "well-behaved"

- **Lemma 1**. The path in G' must start at s^{in} and end at t^{out}.
- **Lemma 2**. If we first enter a vertex gadget at u^{in} (or u^{out}) we must proceed to u^{mid} and then to u^{out} (or u^{in}).
- Proof. If we go to u^{in} and do not proceed to u^{mid}, we have nowhere to go when we reach u^{mid} in the future.
- u^{mid} must be an endpoint of the path, contradicting to Lemma 1.

$\text{DirectedHamiltonianPath} \leq_k \text{HamiltonianPath}$

If G' is a yes HamiltonianPath instance, G is a yes $\text{DirectedHamiltonianPath}$ instance:

Show that the yes HamiltonianPath instance is "well-behaved"

- **Lemma 1**. The path in G' must start at s^{in} and end at t^{out}.
- **Lemma 2**. If we first enter a vertex gadget at u^{in} (or u^{out}) we must proceed to u^{mid} and then to u^{out} (or u^{in}).
- **Lemma 3**. The pattern of the path must be $in \to mid \to out \to in \to mid \to out \to \cdots$

$\text{DirectedHamiltonianPath} \leq_k \text{HamiltonianPath}$

If G' is a yes HamiltonianPath instance, G is a yes $\text{DirectedHamiltonianPath}$ instance:

Show that the yes HamiltonianPath instance is "well-behaved"

- **Lemma 1**. The path in G' must start at s^{in} and end at t^{out}.
- **Lemma 2**. If we first enter a vertex gadget at u^{in} (or u^{out}) we must proceed to u^{mid} and then to u^{out} (or u^{in}).
- **Lemma 3**. The pattern of the path must be $in \to mid \to out \to in \to mid \to out \to \cdots$
  - Proof. We start at s^{in} (Lemma 1) and we must go to s^{mid} and s^{out} (Lemma 2).
  - Each u^{out} is only connected to an v^{in}, and we need to proceed to v^{mid} and v^{out} (Lemma 2).

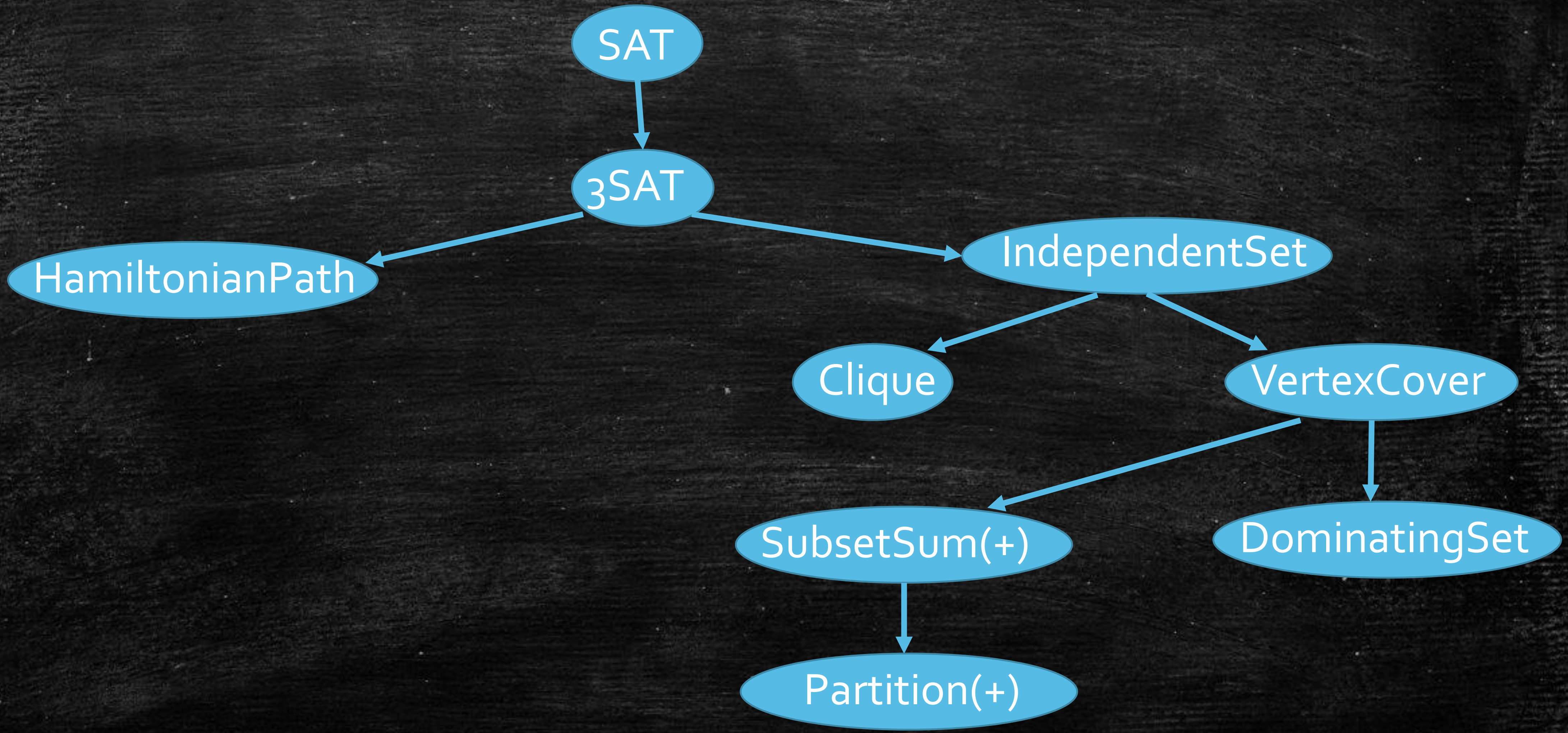
$\text{DirectedHamiltonianPath} \leq_k \text{HamiltonianPath}$

If G' is a yes HamiltonianPath instance, G is a yes $\text{DirectedHamiltonianPath}$ instance:

Show that the yes HamiltonianPath instance is "well-behaved"

- **Lemma 1**. The path in G' must start at s^{in} and end at t^{out}.
- **Lemma 2**. If we first enter a vertex gadget at u^{in} (or u^{out}) we must proceed to u^{mid} and then to u^{out} (or u^{in}).
- **Lemma 3**. The pattern of the path must be $in \to mid \to out \to in \to mid \to out \to \cdots$
- Now we have a Hamiltonian path in G' corresponding to a path in G.

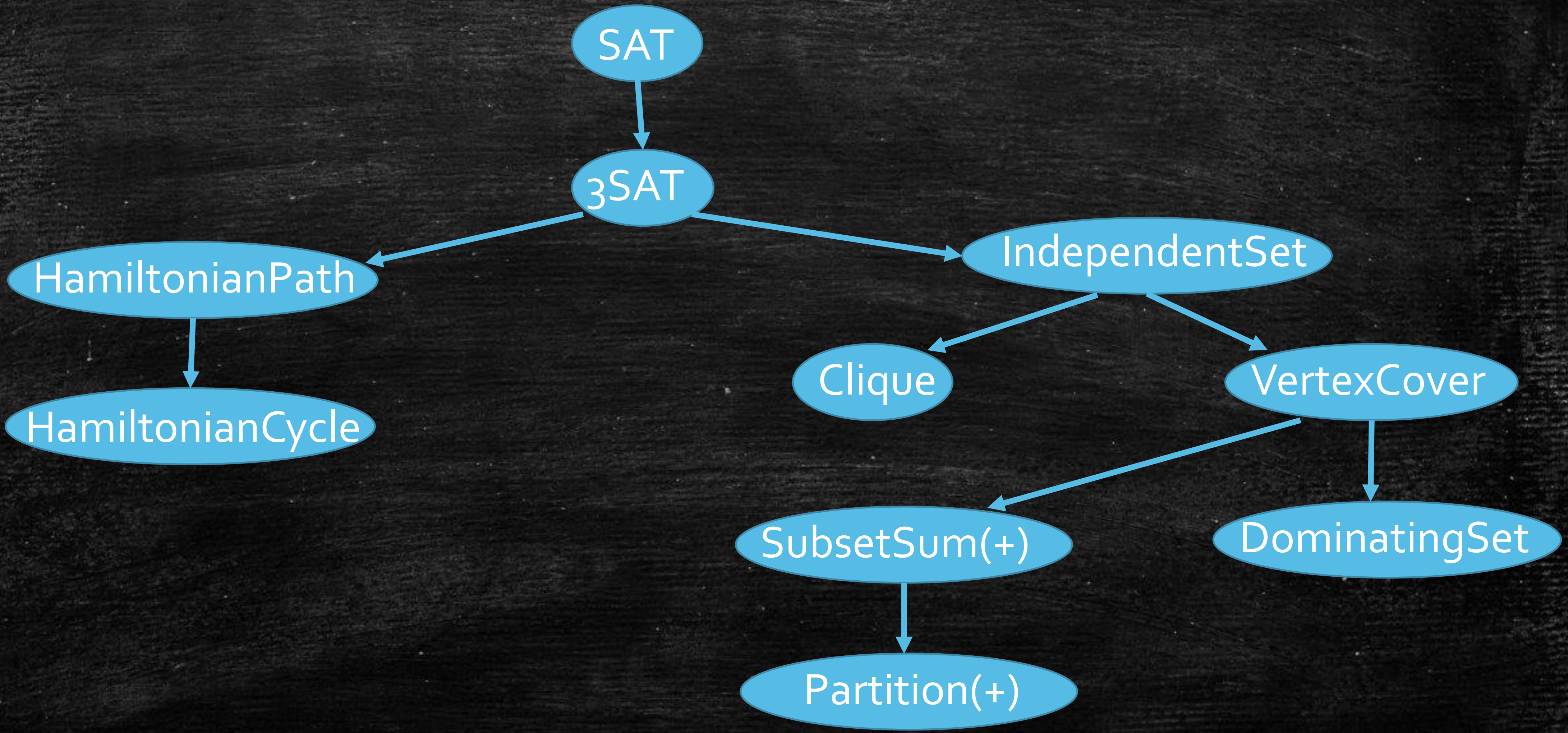
Web of NP-complete Problems



Hamiltonian Cycle

- Given an undirected graph $G = (V, E)$, a Hamiltonian cycle is a cycle that visits each vertex exactly once.
- [HamiltonianCycle] Given an undirected graph $G = (V, E)$, decide if G contains a Hamiltonian cycle.
- Exercise: Prove that HamiltonianCycle is NP-complete.

Web of NP-complete Problems



Five Most Important NP-Complete Problems

Most NP-complete problems can be reduced from...

- 3SAT
- IndependentSet (Clique)
- VertexCover
- SubsetSum (Partition)
- HamiltonianPath (HamiltonianCycle)

Techniques we have seen…

1. Choose the right problem to reduce from
2. Fix the reduction by minor modifications
3. Show the contrapositive for the mapping of no instances
4. Show the yes instance being reduced to is “well-behaved”
5. Do not mess-up the direction
6. Introduce intermediate problems
7. Use gadgets – be creative

NP-Hard vs NP-Complete

Difference between NP-hardness and NP-completeness:

- For decision problems: NP-complete = NP-hard + (in **NP**)
  - There are NP-hard problems that are not in **NP**; these problems are even harder than NP-complete problems.
- NP-hardness can describe optimization/search problems

NP-hard Optimization Problems (Informal)

- A maximization problem is NP-hard if there exists $k \in \mathbb{R}$ such that deciding whether $\mathrm{OPT} \geq k$ is NP-hard.
- A minimization problem is NP-hard if there exists $k \in \mathbb{R}$ such that deciding whether $\mathrm{OPT} \leq k$ is NP-hard.
- If there exists a polynomial time algorithm to solve an NP-hard optimization problem, then **P** = **NP**.
  - If OPT can be computed in polynomial time, whether $\mathrm{OPT} \geq k$ ($\mathrm{OPT} \leq k$) can also be decided in polynomial time.
  - Solving an NP-hard optimization problem in polynomial time implies **P** = **NP**.

NP-hard Optimization Problem Examples

- [Max-3SAT] Maximizing the number of satisfying clauses.
  - NP-hard to decide if $\mathrm{OPT} \geq \mathrm{NumOfClauses}$
- [Max-IndependentSet] Maximizing the size of the independent set.
  - NP-hard to decide if $\mathrm{OPT} \geq k$
  - Note: existence of k-independent set implies $\mathrm{OPT} \geq k$.
- [Min-VertexCover] Minimizing the size of the vertex cover.
  - NP-hard to decide if $\mathrm{OPT} \leq k$
  - Note: existence of k-vertex cover implies $\mathrm{OPT} \leq k$.
- [LongestPath] Maximizing the length of a simple path.
  - NP-hard to decide if $\mathrm{OPT} \geq |V|$ (HamiltonianPath)

Knapsack (Revisited)

- Knapsack is NP-hard.
- Reduction from SubsetSum+
- Given a SubsetSum+ instance $(\{a_1, \dots, a_n\}, k)$, construct a Knapsack instance as follows.
- n items: $w_i = v_i = a_i$
- Capacity = k
- Yes SubsetSum+ instance $\Longrightarrow \mathrm{OPT}_{\mathrm{Knapsack}} = k$
- No SubsetSum+ instance $\Longrightarrow \mathrm{OPT}_{\mathrm{Knapsack}} < k$
- Deciding if $\mathrm{OPT}_{\mathrm{Knapsack}} \geq k$ is NP-hard, so Knapsack is NP-hard

Travelling Salesman Problem (TSP)

- [TSP] Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?
- [TSP (Formulation)] Given a weighted and complete undirected graph $G = (V, E = V \times V, w)$, find a Hamiltonian cycle with minimum length.
- Differences with HamiltonianCycle:
  - A Hamiltonian cycle always exists for TSP
  - But the graph is weighted, we need to optimize the path length

TSP is NP-hard

- Given a HamiltonianPath instance $G = (V, E)$, we construct a TSP instance $G = (V', E', w)$ such that
  - $V' = V$
  - $w(u, v) = 1$ if $(u, v) \in E$
  - $w(u, v) = |V|^{2615}$ is a very large number if $(u, v) \notin E$
- It's NP-hard to decide if optimal tour has length at most $|V|$.

TSP is even hard to "approximate"!

- **Theorem.** Suppose **P** $\neq$ **NP**. There is no polynomial time α-approximation algorithm for TSP for any $\alpha \geq 1$ that may depend on the instance.
- Theorem holds for exponentially large α, e.g., $\alpha = (2615|V|)^{2615|V|}$.
- Proof. Change $|V|^{2615}$ to $\alpha|V| + 1$ in the previous reduction.
- Yes HamiltonianCycle instance $\Longrightarrow \mathrm{OPT}_{\mathrm{TSP}} = |V|$
- No HamiltonianCycle instance $\Longrightarrow \mathrm{OPT}_{\mathrm{TSP}} \geq \alpha|V| + 1$
- Let ALG be the output of an α-approximation algorithm $\mathcal{A}$.
- $\mathrm{ALG} \leq \alpha|V| \quad \Longrightarrow$ yes HamiltonianCycle instance
- $\mathrm{ALG} \geq \alpha|V| + 1 \quad \Longrightarrow$ no HamiltonianCycle instance

This Lecture

- Show more important NP-complete problems.
  - DominatingSet
  - SubsetSum (Partition)
  - HamiltonianPath (HamiltonianCycle)
- Learn some elementary techniques for reduction.
- Learn how to write a formal proof for NP-completeness.
- NP-hard optimization problems
  - Knapsack
  - TSP

Extra – Naming for **P** and **NP**

- **P**: polynomial-time
- **NP**: non-deterministic polynomial-time

- Deterministic Turing Machine (the normal TM we have seen):
  - Transition $\delta: Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$
- Non-deterministic Turing Machine
  - Specify two transitions δ_1, δ_2 for each state-alphabet tuple.

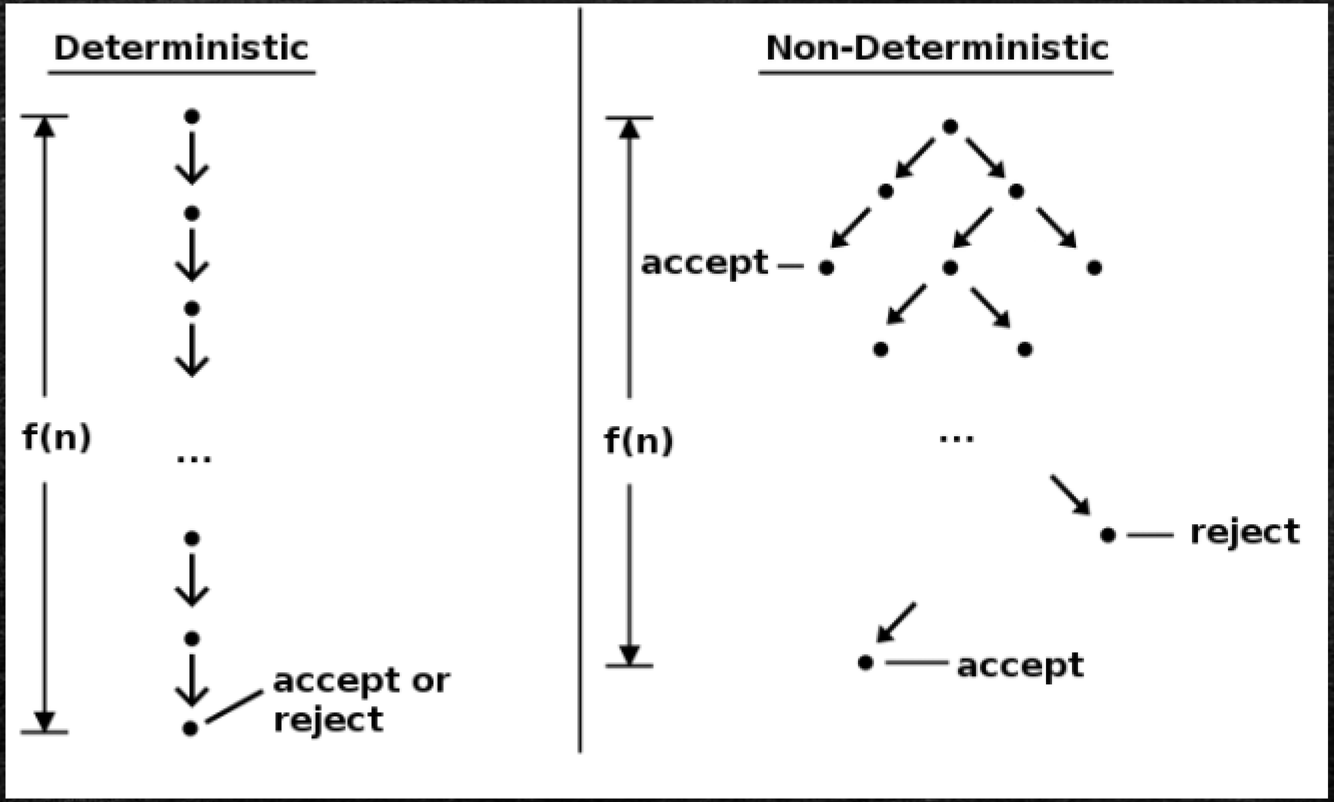


Image from: https://en.wikipedia.org/wiki/Nondeterministic_Turing_machine

Polynomial Time NTM

- A non-deterministic Turing machine runs in polynomial time if, upon receiving input x, all branches reach halting states within $O(|x|^c)$ steps for some constant $c > 0$.

Original Definition for **NP**

- Definition. A decision problem $f: \Sigma^* \to \{0,1\}$ is in **NP** if there is a polynomial time NTM $\mathcal{A}$ such that
  - There is a branch of $\mathcal{A}(x)$ that reaches the accepting state if $f(x) = 1$
  - All branches of $\mathcal{A}(x)$ reach the rejecting state if $f(x) = 0$

- This definition is equivalent to the "certificate definition":
  - Each bit of the certificate corresponds to the "instruction" for which of δ_1, δ_2 we are following.
  - For the yes instance, the certificate "instructs" us to move along the branch that reach the accepting state.
  - For the no instance, no "instruction" can help us reach the accepting state.

SAT $\in$ **NP**

- We consider the NTM that enumerates the values of $x_1, \dots, x_n$ in the first n steps.
- Now we have 2^n “terminals” after first n steps.
- For each terminal, verify if ϕ is satisfied; go to the accepting state if it is, and go to the rejecting state if not.