香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF DATA SCIENCE
數據科學學院

CSC 6011 - Theory of Computation

Lecture 12 - NP-Completeness

Time: Mon & Wed, 4:00 - 5:20 PM

Location: Teaching A, 107

Instructor: Tao LIN (林涛)

TA: Yefan GAO (高叶繁)

Fall 2026

Lecture 12 - NP-Completeness

Last time:

- $\mathrm{NTIME}(t(n))$, NP
- P vs NP question
- Dynamic Programming, $A_{\mathrm{CFG}} \in$ P
- Satisfiability Problem (SAT)

Today:

- Polynomial-time reduction
- NP-completeness

Terminology

$A =$ Does this graph has a Hamiltonian path?

$x = (G, s, t)$

$\mathcal{A}_{\text{HP}}$

$\mathcal{A}_{\text{HP}}$ says: "G has a Hamiltonian path from s to t"

(G, s, t) that has a Hamiltonian path from s to t

(Decision) Problem A	Language $L \subseteq \Sigma^*$
Input to problem: Instance x	String s (may or may not be in the language)
Solved by an algorithm	Decided by a Turing machine
Output "Yes"	Accept
Output "No"	Reject
"Yes" instance x	String $s \in L$
"No" instance x	String $s \notin L$

$L = \text{HAMPATH}$

$s = \langle G, s, t \rangle$

D_{HP}

D_{HP} accepts $\langle G, s, t \rangle$

$\langle G, s, t \rangle \in \text{HAMPATH}$

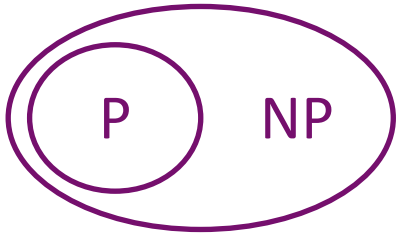
P and NP: language classes = classes of (decision) problems.

Quick Review

NP = All languages where we can verify membership quickly

P = All languages where we can decide membership quickly

P versus NP question: Is P = NP?



$\mathrm{SAT} = \{\langle\phi\rangle | \ \phi$ is a satisfiable Boolean formula$\}$

Cook-Levin Theorem: $\mathrm{SAT} \in$ P $\rightarrow$ P = NP

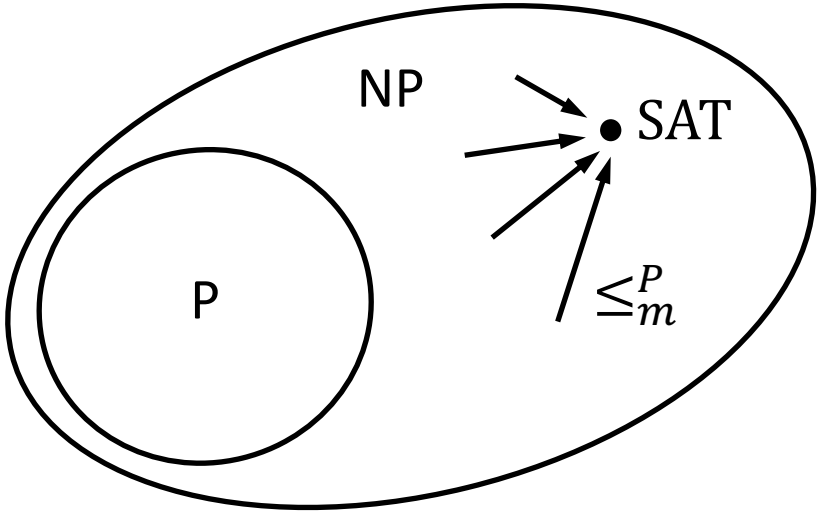
Proof plan: Show that every $A \in$ NP is polynomial-time reducible to SAT.

Polynomial-Time Reduction

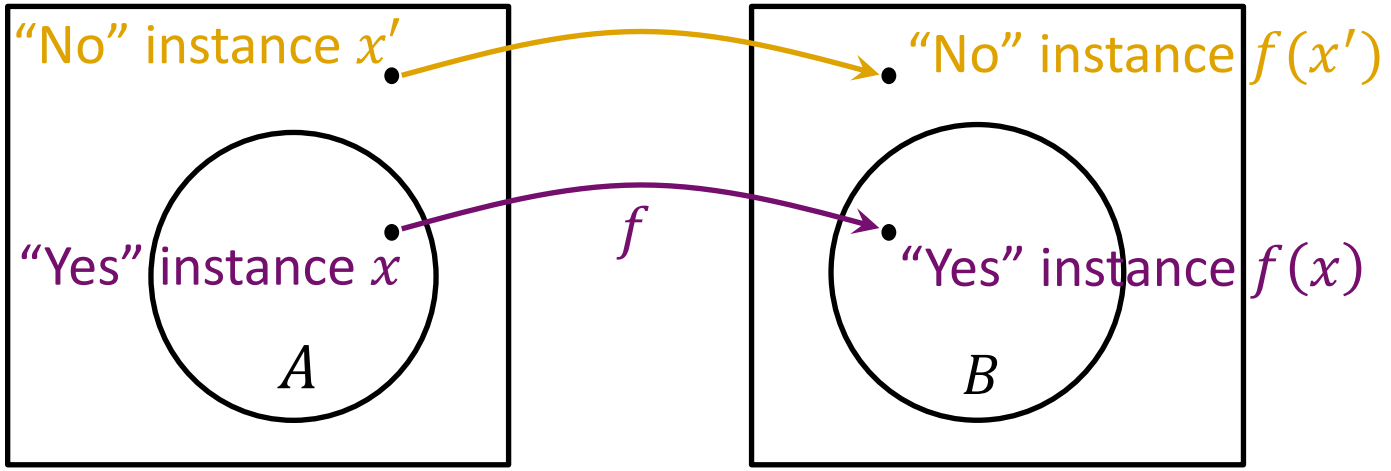
Defn: A is <u>polynomial-time (mapping) reducible</u> to B ($A \leq_m^P B$) if $A \leq_m B$ by a reduction function f that is computable in polynomial time. In particular,

x is a "Yes" instance for A iff $f(x)$ is a "Yes" instance for B

Theorem: If $A \leq_m^P B$ and $B \in$ P then $A \in$ P.



Idea to show $\mathrm{SAT} \in$ P $\rightarrow$ P = NP



f is computable in polynomial time

$\leq_m^P$ Example: 3SAT and CLIQUE

Defn: A Boolean formula ϕ is in Conjunctive Normal Form (CNF) if it has the form:

$$\phi = \underbrace{(x \vee \overline{y} \vee z)}_{\text{clause}} \wedge \underbrace{(\overline{x} \vee \overline{s} \vee z \vee u)}_{\text{clause}} \wedge \cdots \wedge \underbrace{(\overline{z} \vee \overline{u})}_{\text{clause}}$$

literals

Literal: a variable or a negated variable

Clause: an OR ($\vee$) of literals.

CNF: an AND ($\wedge$) of clauses.

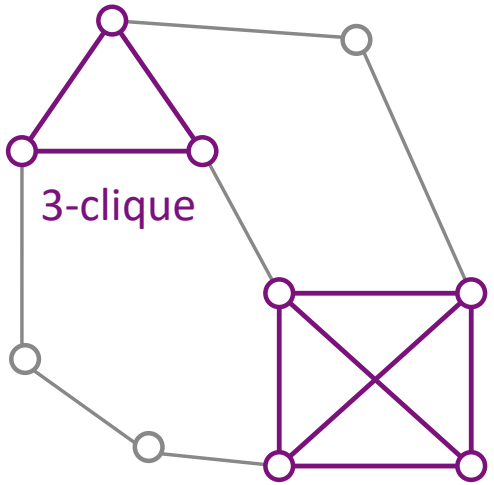
3CNF: a CNF with exactly 3 literals in each clause.

$3\text{SAT} = \{\langle\phi\rangle | \ \phi \text{ is a satisfiable 3CNF formula}\}$

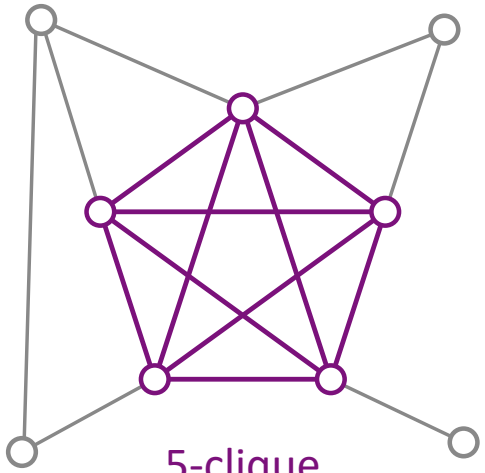
Defn: A k-clique in a graph is a subset of k nodes with an edge between every pair.

$\text{CLIQUE} = \{\langle G, k\rangle | \text{ graph } G \text{ contains a } k\text{-clique}\}$

Will show: $3\text{SAT} \leq_m^P \text{CLIQUE}$



4-clique



5-clique

$3\text{SAT} \leq_m^P \text{CLIQUE}$

Theorem: $3\text{SAT} \leq_m^P \text{CLIQUE}$

Proof: Give polynomial-time reduction f that maps 3CNF formula ϕ to $\langle G, k \rangle$ such that ϕ is satisfiable ("Yes" instance) iff G has a k-clique ("Yes" instance).

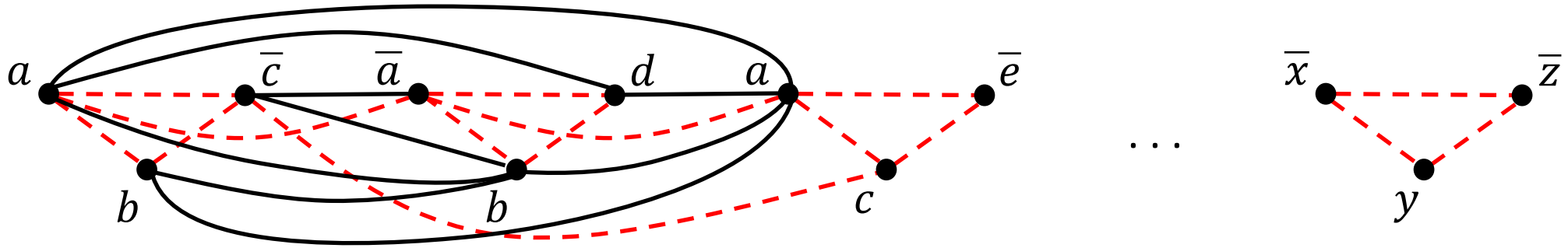
A satisfying assignment to a CNF formula has ≥1 true literal in every clause.

$$\phi = (a \vee b \vee \overline{c}) \wedge (\overline{a} \vee b \vee d) \wedge (a \vee c \vee \overline{e}) \wedge \cdots \wedge (\overline{x} \vee y \vee \overline{z})$$

f ↓

G =

k = # clauses



Forbidden edges:

1) between literals within a clause
2) opposite literals (a and $\overline{a}$)

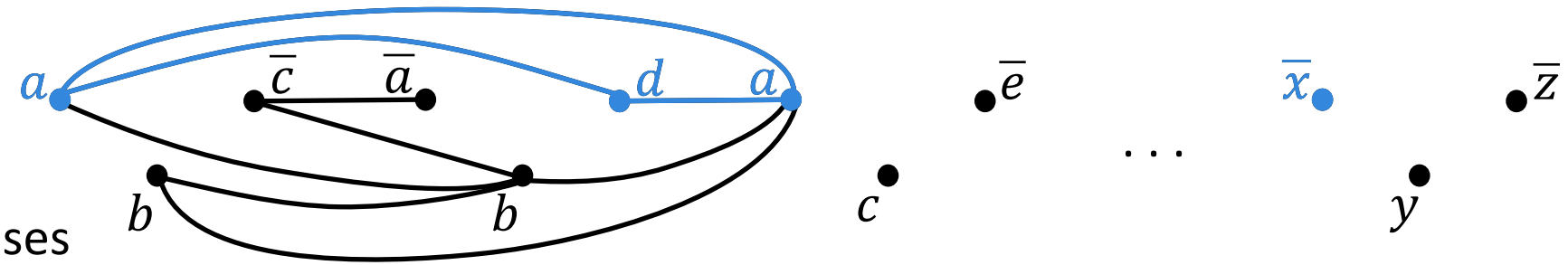
Let G contain <u>all</u> non-forbidden edges

$3\text{SAT} \leq_m^P$ CLIQUE conclusion

$$\phi = (a \vee b \vee \overline{c}) \wedge (\overline{a} \vee b \vee d) \wedge (a \vee c \vee \overline{e}) \wedge \cdots \wedge (\overline{x} \vee y \vee \overline{z})$$

$f \downarrow$

$\boxed{\begin{matrix} G \\ k \end{matrix}} = \begin{matrix} \\ = \#\text{ clauses} \end{matrix}$



Claim: ϕ is satisfiable iff G has a k-clique.

(→) **Take any satisfying assignment to ϕ.** Pick 1 true literal in each clause.
The corresponding nodes in G are a k-clique, because they don't have forbidden edges.

(←) **Take any k-clique in G.** It must have exactly 1 node in each clause.
Set each corresponding literal TRUE. That gives a satisfying assignment to ϕ.

The reduction f is computable in polynomial time.

Thus, $3\text{SAT} \leq_m^P \text{CLIQUE}$ **Corollary:** $\text{CLIQUE} \in \text{P} \rightarrow 3\text{SAT} \in \text{P}$

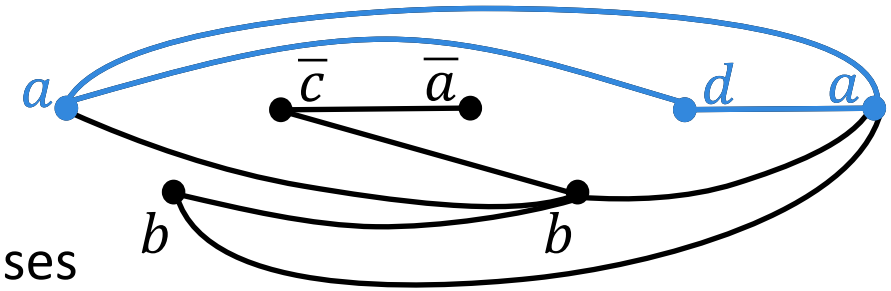
$3\text{SAT} \leq_m^P$ CLIQUE conclusion

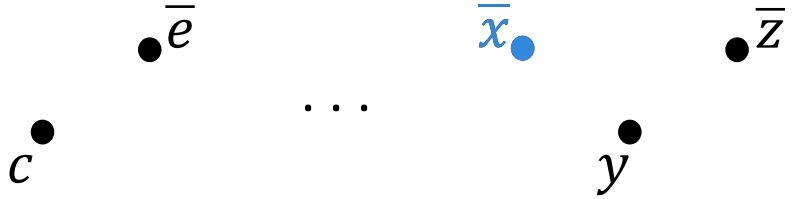
$$\phi = (a \vee b \vee \overline{c}) \wedge (\overline{a} \vee b \vee d) \wedge (a \vee c \vee \overline{e}) \wedge \cdots \wedge (\overline{x} \vee y \vee \overline{z})$$

f ↓

G =

k = # clauses





Claim: ϕ is satisfiable iff G has a k-clique.

(→) **Take any satisfying assignment to ϕ.** Pick 1 true literal in each clause. The corresponding nodes in G are a k-clique, because they don't have

(←) **Take any k-clique in G.** It must have exactly 1 node in each clause. Set each corresponding literal TRUE. That gives a satisfying assignment

The reduction f is computable in polynomial time.

Thus, $3\text{SAT} \leq_m^P \text{CLIQUE}$ **Corollary:** $\text{CLIQUE} \in \text{P} \rightarrow 3\text{SAT} \in \text{P}$

Mini-Quiz 12.1

Does this proof require 3 literals per clause?

(a) Yes, to prove the claim.

(b) Yes, to ensure poly-time reduction.

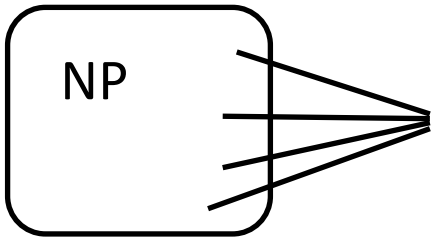
(c) No, it works for any clause sizes.

NP-hardness and NP-completeness

Defn: B is NP-hard if:

for any $A \in$ NP, $A \leq_m^P B$

(Harder than all NP problems)



next lecture today

$\leq_m^P$ SAT $\leq_m^P$ 3SAT $\leq_m^P$ CLIQUE

3SAT $\leq_m^P$ HAMPATH

Defn: B is NP-complete if

1) $B \in$ NP

2) B is NP-hard.

Cook-Levin Theorem:

Previously: if SAT $\in$ P, then P = NP

Now: SAT is NP-complete

Proof: Next lecture. Assume for now.

Proof template:

To prove B is NP-complete:

(1) show $B \in$ NP

(2) show B is NP-hard:
by reducing a known NP-complete (or NP-hard) problem to B.
(e.g., 3SAT)

Importance of NP-hardness and NP-completeness

1) Evidence that a problem is computationally intractable.

2) A candidate for proving P ≠ NP.

HAMPATH is NP-complete

Theorem: HAMPATH is NP-complete

(1) In NP: HAMPATH $\in$ NP (Lecture 11).

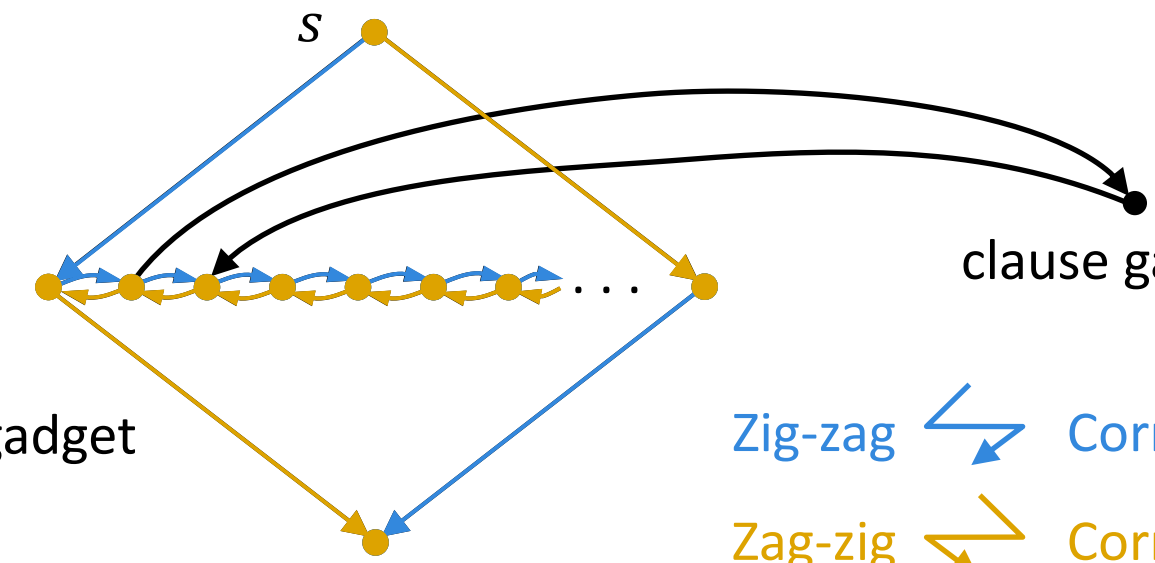
(2) NP-Hardness: Show 3SAT $\leq_m^P$ HAMPATH (assuming 3SAT is NP-complete)

Idea: "Simulate" variables and clauses with "gadgets"

$$\phi = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_4) \wedge \cdots \wedge (\quad)$$

$f \downarrow$

$\langle G, s, t \rangle$



Zig-zag Corresponds to setting x_1 TRUE

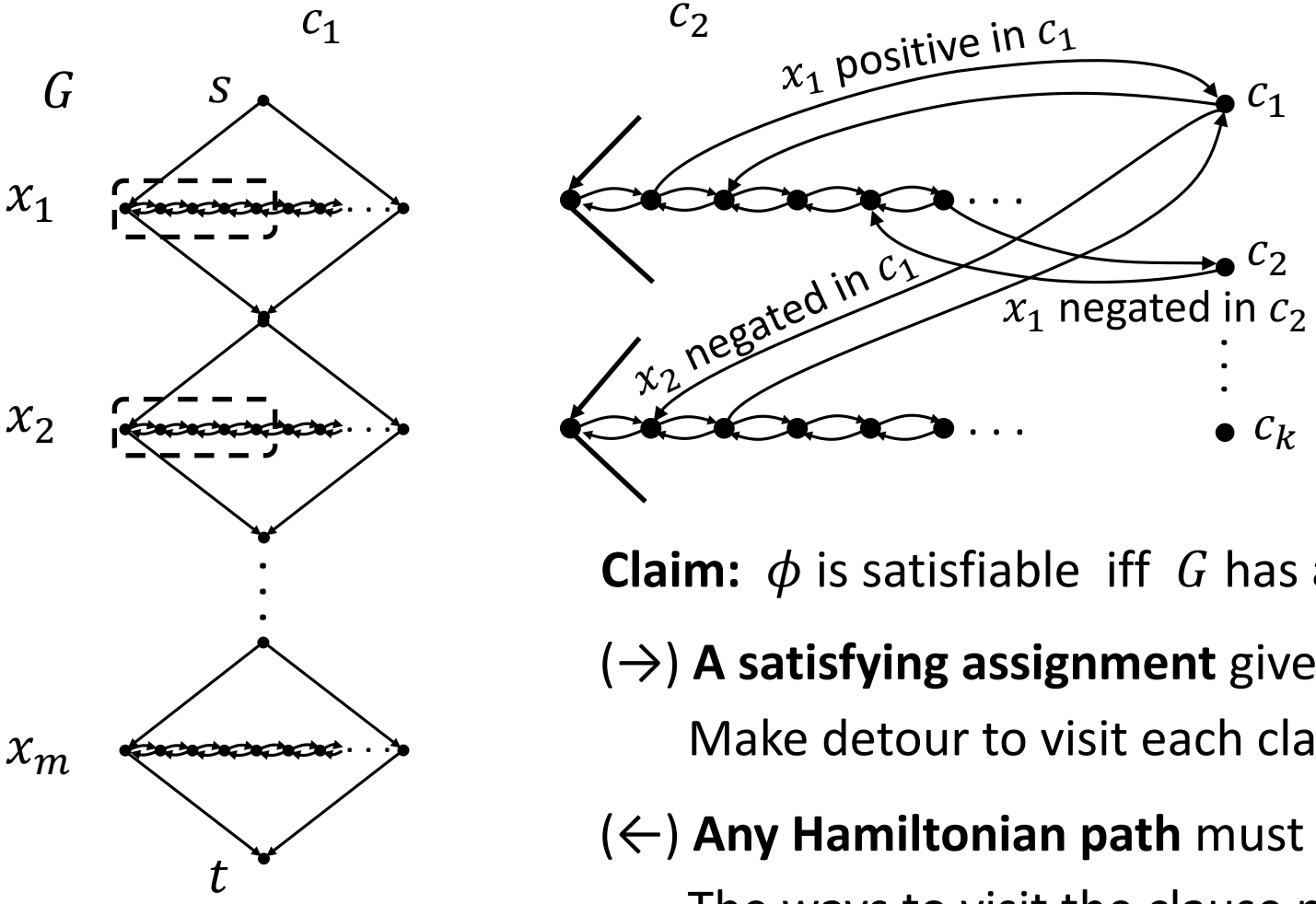
Zag-zig Corresponds to setting x_1 FALSE

Construction of G

$$\phi = \underbrace{(x_1 \vee \overline{x}_2 \vee x_3)}_{c_1} \wedge \underbrace{(\overline{x}_1 \vee x_2 \vee x_4)}_{c_2} \wedge \cdots \wedge \underbrace{(\quad x_m \quad)}_{c_k}$$

m variables

k clauses



The reduction f is computable in polynomial time.

Claim: ϕ is satisfiable iff G has a Hamiltonian path from s to t.

(→) **A satisfying assignment** gives each variable's Zig-zag or Zag-zig choice.
Make detour to visit each clause node c_i.

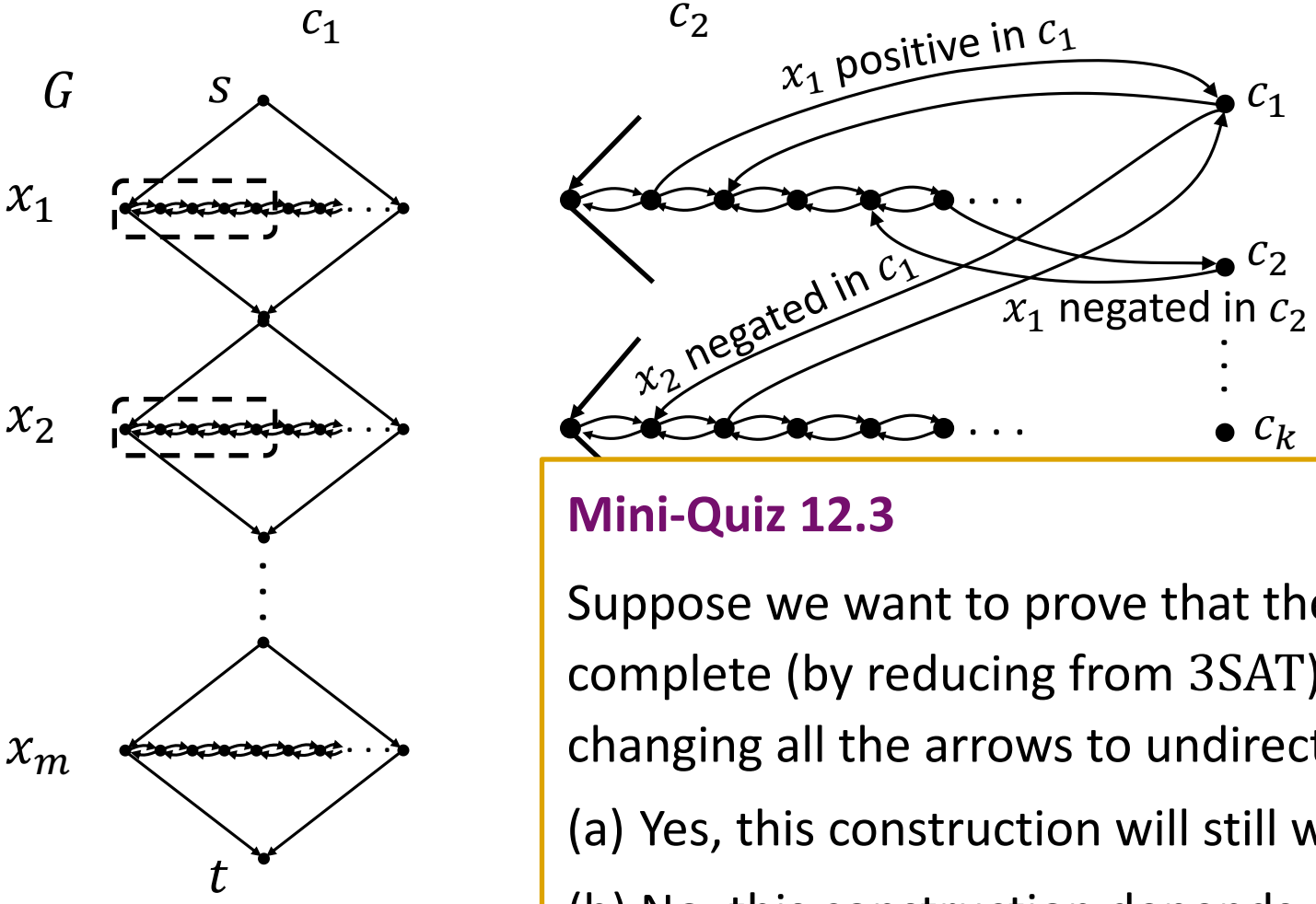
(←) **Any Hamiltonian path** must have Zig-zag and Zag-zig patterns.
The ways to visit the clause nodes give a satisfying assignment.

Construction of G

$$\phi = \underbrace{(x_1 \vee \overline{x}_2 \vee x_3)}_{c_1} \wedge \underbrace{(\overline{x}_1 \vee x_2 \vee x_4)}_{c_2} \wedge \cdots \wedge \underbrace{(\quad x_m \quad)}_{c_k}$$

m variables

k clauses



The reduction f is computable in polynomial time.

Mini-Quiz 12.3

Suppose we want to prove that the **Undirected**-HAMPATH problem is NP-complete (by reducing from 3SAT). Can we still use this construction by changing all the arrows to undirected lines?

(a) Yes, this construction will still work.

(b) No, this construction depends on G being directed.

Quick review of today

1. NP-hardness and NP-completeness

2. SAT and $3\mathrm{SAT}$

3. $3\mathrm{SAT} \leq_m^P \mathrm{CLIQUE}$

4. $3\mathrm{SAT} \leq_m^P \mathrm{HAMPATH}$

5. Strategy to prove NP-completeness:
   Show membership in NP.
   Reduce from a known NP-complete problem using gadgets.

These slides are adapted from Michael Sipser, *18.404J Theory of Computation*, Fall 2020. Massachusetts Institute of Technology:

- Source: MIT OpenCourseWare https://ocw.mit.edu

Except for separately identified third-party material, this adapted presentation is licensed under CC BY-NC-SA 4.0.