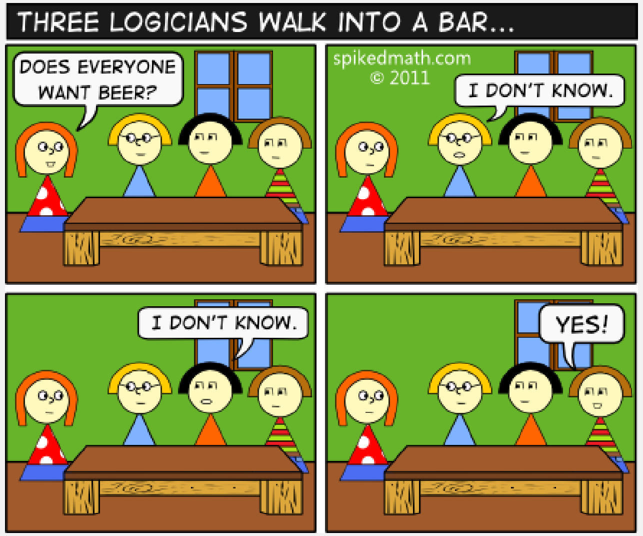
Cook-Levin, and other NP-Complete Problems

Wednesday, November 17, 2021



Announcements

- HW 8 due tonight

- HW9 out tomorrow
  - Due after break: 11/28 11:59pm EST

Last Time: **NP**-Completeness

DEFINITION

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

easy

hard????

Must prove for <u>all</u> langs, not just a single language

It's only hard to prove <u>the first</u> **NP-complete** problem!

(Just like figuring out <u>the first</u> **undecidable** problem was hard!)

Last Time: The Cook-Levin Theorem

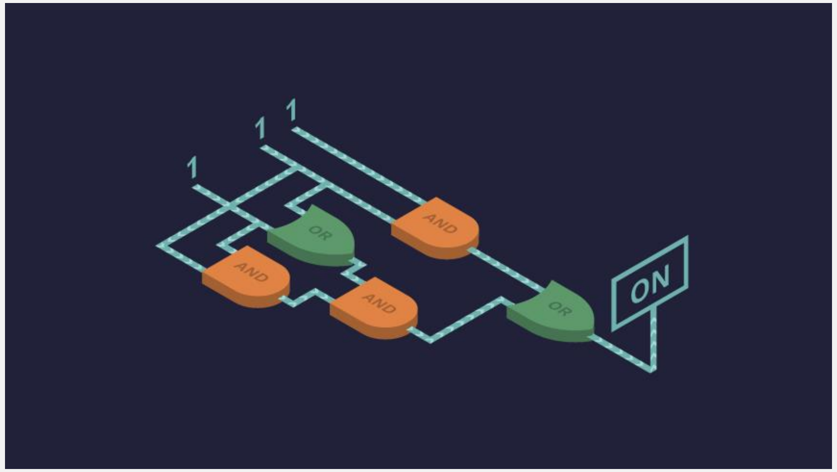
$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$

The first **NP**-Complete problem

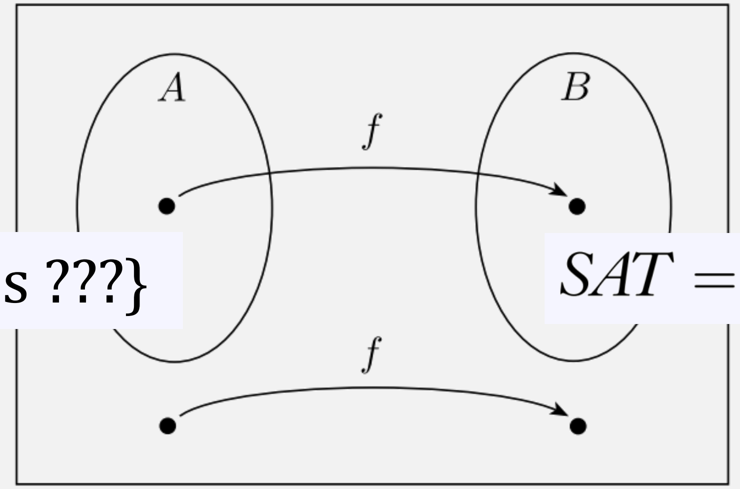
THEOREM

SAT is NP-complete.

But it makes sense that every problem can be reduced to it …



Last Time: Reducing every **NP** lang to *SAT*

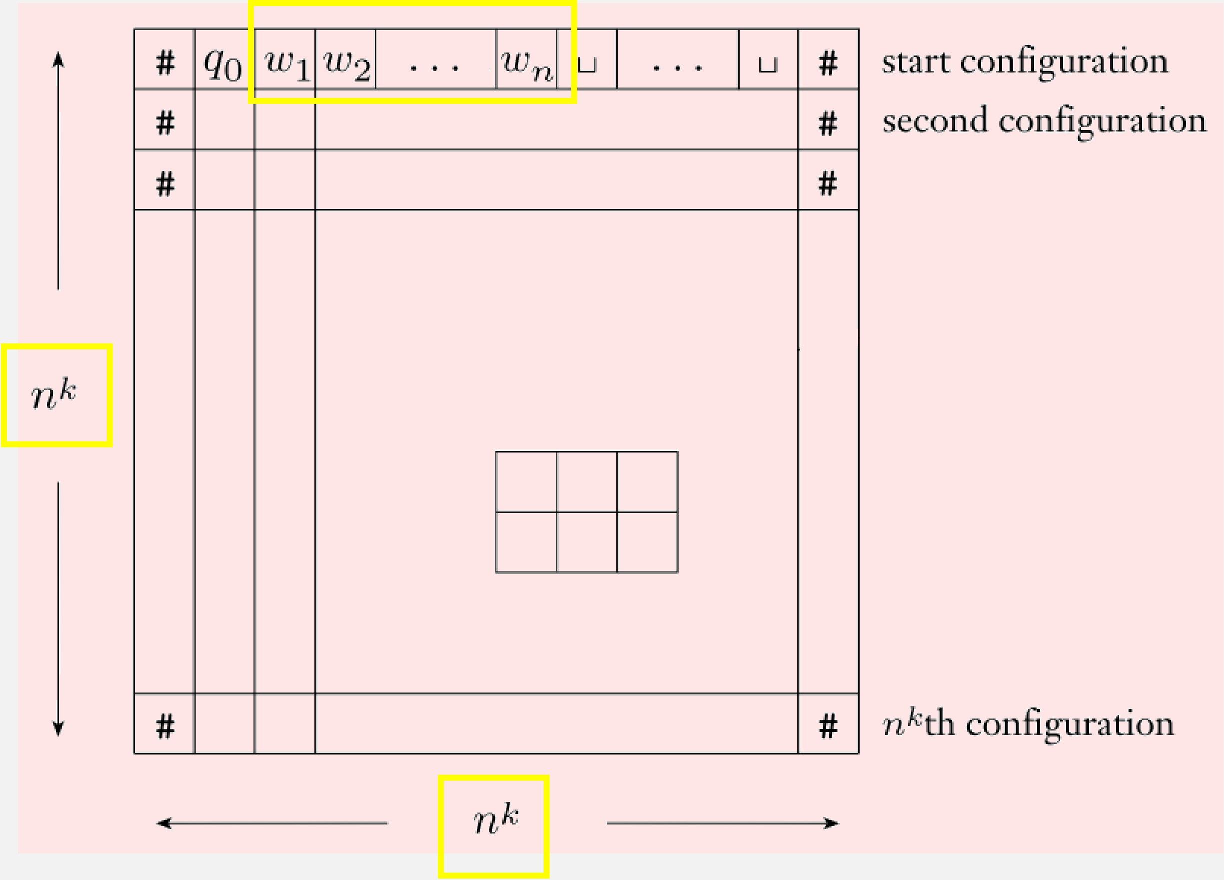


Some **NP** lang = {*w* | *w* is ???}

$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$

How can we reduce some *w* to a Boolean formula if we don't know *w*???

Accepting config sequence = “Tableau”

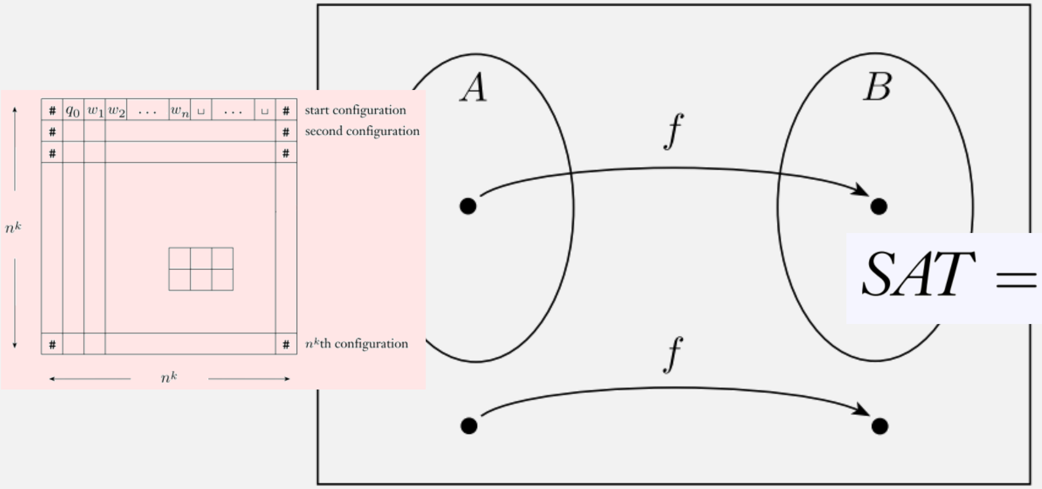


- input $w = w_1 \dots w_n$
- Assume configs start/end with #
- Must have an accepting config
- At most $\boldsymbol{n^k}$ configs
  - (why?)
- Each config has length $\boldsymbol{n^k}$
  - (why?)

Theorem: *SAT* is NP-complete

Proof idea:

- Create a reduction from accepting tableaus to satisfiable formulas
- And vice versa



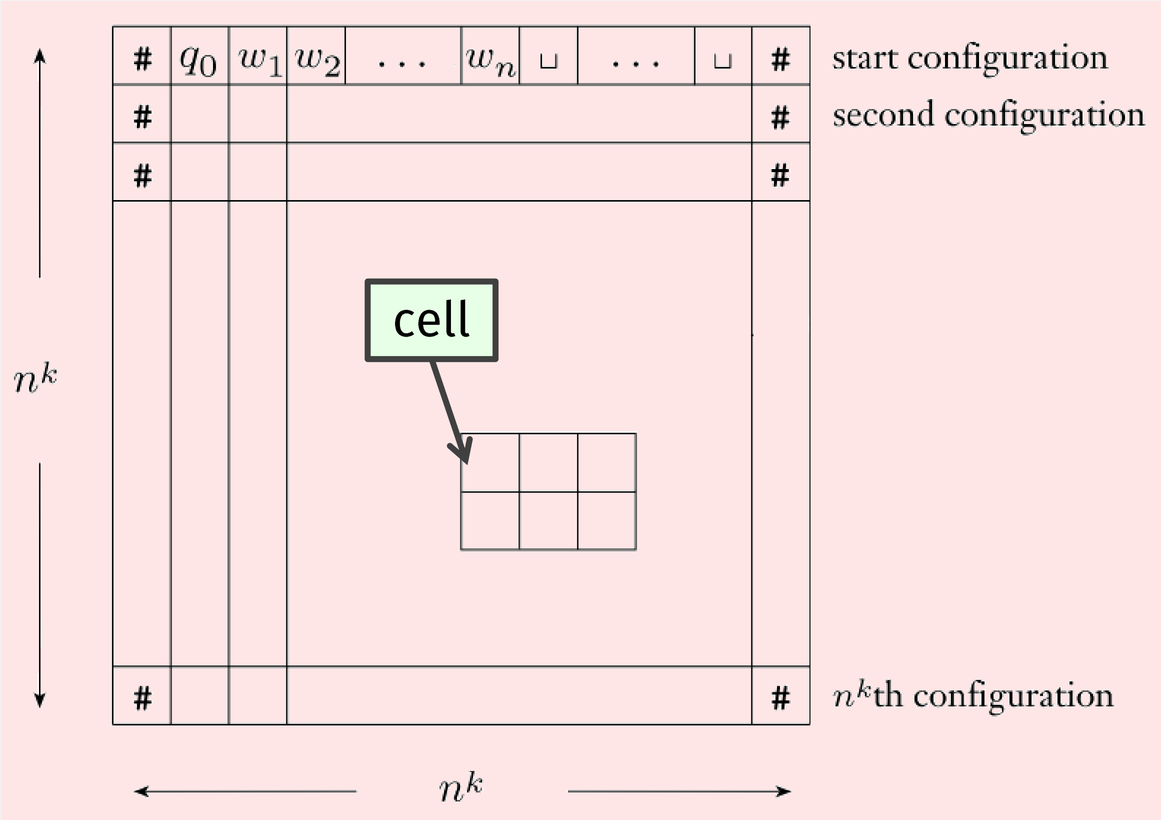
$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$

Tableau Terminology

- A tableau cell has coordinate *i,j*

- A cell has symbol:

$$s \in C = Q \cup \Gamma \cup \{\#\}$$

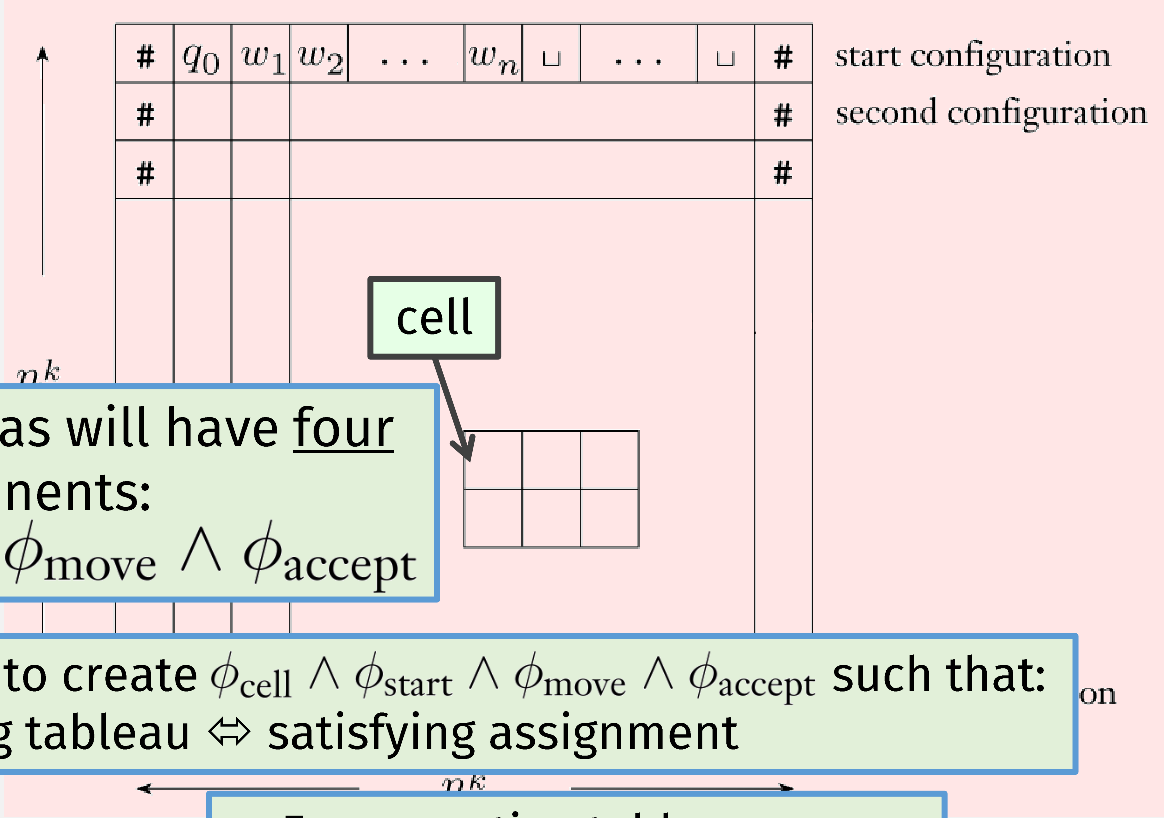


A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** ␣,
3. Γ is the tape alphabet, where ␣ $\in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$ e transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

Formula Variables

- A tableau <u>cell</u> has coordinate *i,j*
- A cell has <u>symbol</u>:
  $s \in C = Q \cup \Gamma \cup \{\#\}$
- For every *i,j,s* create <u>variable</u> $x_{i,j,s}$
  - i.e., one var for every possible symbol/cell combination
- <u>Total</u> variables =
  - # cells * # symbols =
  - $n^k * n^k * |C| = O(\mathbf{n^{2k}})$



Resulting formulas will have <u>four</u> components:
$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$

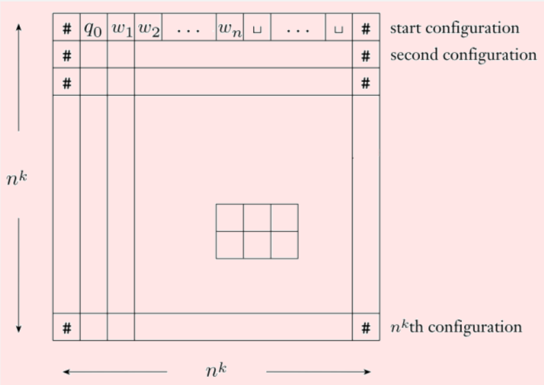
Use these variables to create $\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$ such that:
accepting tableau ⇔ satisfying assignment

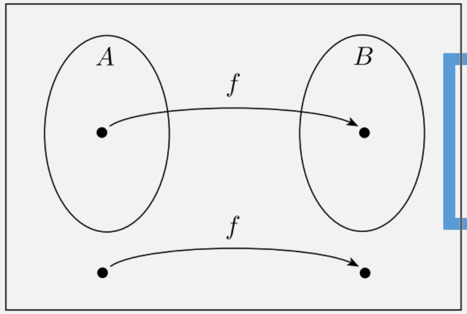
⇒ For <u>accepting tableau</u>:
- **all four parts** must be TRUE

⇐ For <u>non-accepting tableau</u>
- **only one part** must be FALSE

A ***Turing m*** ... $_{\text{ject}})$, where
Q, Σ, Γ are a ...

1. Q is the ...
2. Σ is the input alphabet not containing the ***blank symbol*** ␣,
3. Γ is the tape alphabet, where ␣ $\in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$ e transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.





⇒ accepting tableau: **all four** must be TRUE
⇐ nonaccepting tableau: **one** must be FALSE

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

$$C = Q \cup \Gamma \cup \{\#\}$$

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i,j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

"The following must be TRUE for every cell *i,j*"

"The variable for one *s* must be TRUE"

And only one variable for some *s* must be TRUE

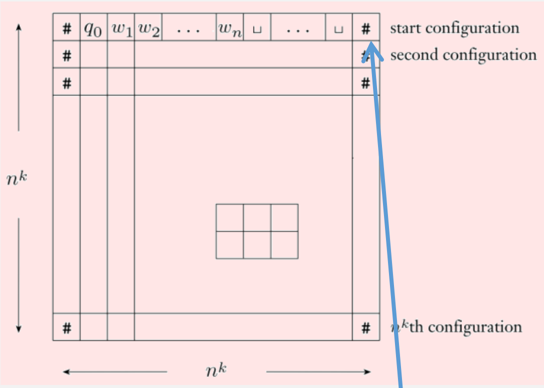
i.e., **every cell has a valid character**

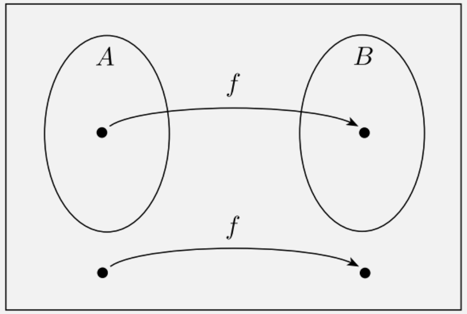
⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **Yes**, assign $x_{i,j,s}$ = TRUE if it's in the tableau,
- and assign other vars = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?

- Not necessarily





⇒ accepting tableau: **all four** must be TRUE
⇐ nonaccepting tableau: **one** must be FALSE

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

For a string w, **start config** is always $\#q_0w_1 \ldots w_n \ldots \#$

The variables in the start config, ANDed together

$$\phi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}$$

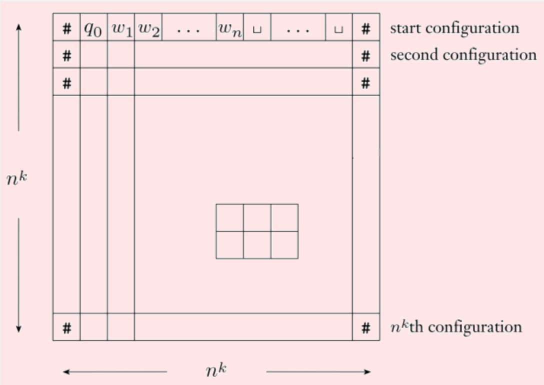
i.e., **tableau has valid start config**

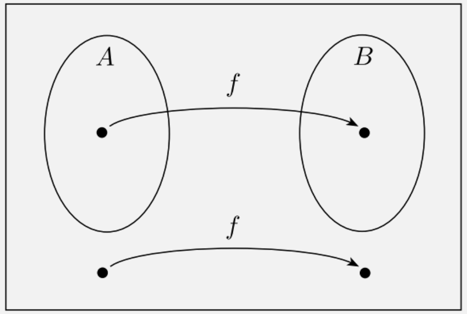
⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **Yes**, assign $x_{i,j,s}$ = TRUE if it's in the tableau,
- and assign other vars = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?

- Not necessarily





ϕ_{cell} ☑ $\wedge$ ϕ_{start} ☑ $\wedge$ ϕ_{move} $\wedge$ ϕ_{accept}

⇒ accepting tableau: **all four** must be TRUE
⇐ nonaccepting tableau: **one** must be FALSE

$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\text{accept}}}$$

The state q_{accept} must appear in some cell

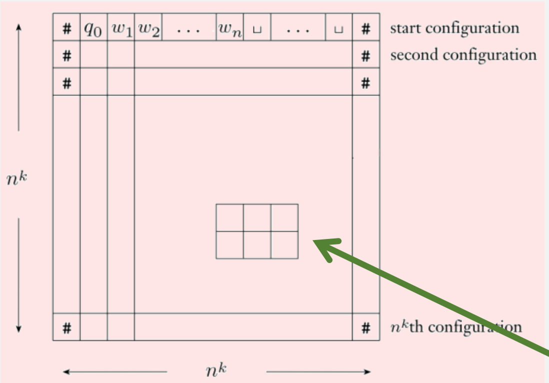
i.e., **tableau has valid accept config**

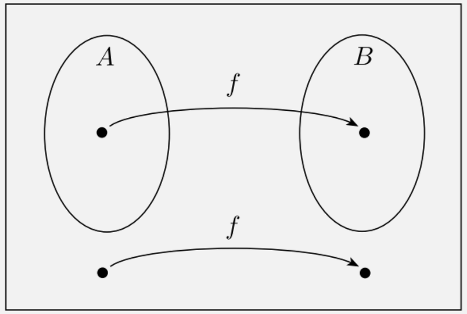
⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **Yes**, assign $x_{i,j,s}$ = TRUE if it's in the tableau,
- and assign other vars = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?

- **Yes**, because it wont have q_{accept}





⇒ accepting tableau: **all four** must be TRUE
⇐ nonaccepting tableau: **one** must be FALSE ☑

$$\phi_{\text{cell}} ☑ \wedge \phi_{\text{start}} ☑ \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}} ☑$$

- Ensures that every configuration is legal according to the previous configuration and the TM's δ transitions
- Only need to verify every 2×3 "window"
  - Why?
  - Because in one step, only the cell at the head can change
- E.g., if $\delta(q_1, \texttt{b}) = \{(q_2,\texttt{c},\text{L}), (q_2,\texttt{a},\text{R})\}$
  - Which are legal?

(a) 🙂

a	q_1	b
q_2	a	c

(b) 🙂

a	q_1	b
a	a	q_2

(c) ???

a	a	q_1
a	a	b

(d) 🙂

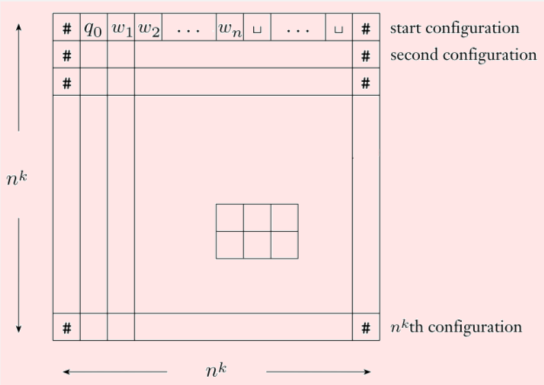
#	b	a
#	b	a

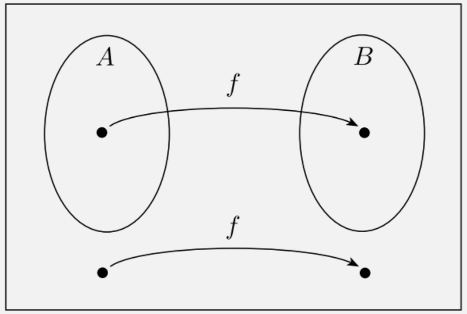
(e) 🙂

a	b	a
a	b	q_2

(f) 🙂

b	b	b
c	b	b





⇒ accepting tableau: **all four** must be TRUE
⇐ nonaccepting tableau: **one** must be FALSE ☑

ϕ_{cell} ☑ $\wedge$ ϕ_{start} ☑ $\wedge$ ϕ_{move} $\wedge$ ϕ_{accept} ☑

i.e., **all transitions are legal, according to δ fn**

$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} \left(\text{the } (i,j)\text{-window is legal}\right)$$

i,j = upper center cell

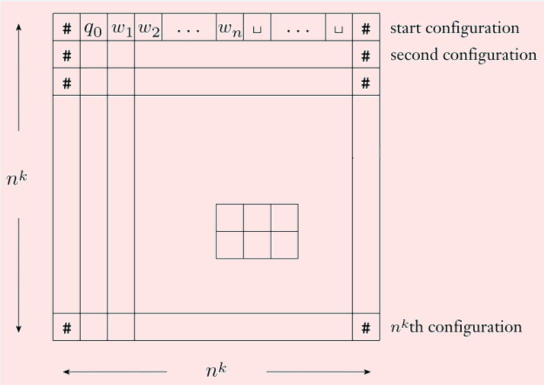
$$\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}\right)$$

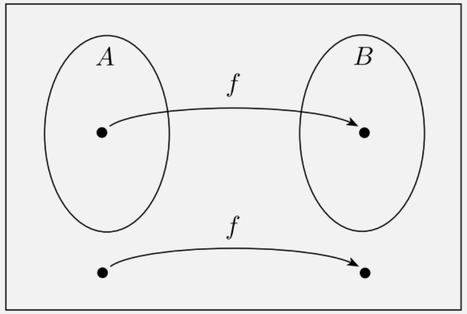
⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **Yes**, assign $x_{i,j,s}$ = TRUE if it's in the tableau,
- and assign other vars = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?

- Not necessarily





⇒ accepting tableau: **all four** must be TRUE ✓
⇐ nonaccepting tableau: **one** must be FALSE ✓

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} \left(\text{the } (i, j)\text{-window is legal}\right)$$

i,j = upper center cell

$$\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}\right)$$

⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **Yes**, assign $x_{i,j,s}$ = TRUE if it's in the tableau,
- and assign other vars = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?

- Not necessarily

To Show Poly Time Mapping Reducibility …

Language A is ***polynomial time mapping reducible***, or simply ***polynomial time reducible***, to language B, written $A \leq_P B$, if a polynomial time computable function $f: \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the ***polynomial time reduction*** of A to B.

To show poly time mapping reducibility:

☑ 1. create **computable fn**,

➡ 2. show that it **runs in poly time**,

☑ 3. then show **forward direction** of mapping red.,

4. and **reverse direction**
☑ (or **contrapositive** of **forward direction**)

Time complexity of the reduction

- Number of cells = $O(\boldsymbol{n^{2k}})$

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

$O(\boldsymbol{n^{2k}})$

"The following must be TRUE for every cell i,j"

"The variable for one s must be TRUE"

And only one variable for some s must be TRUE

Time complexity of the reduction

- Number of cells = $O(\boldsymbol{n^{2k}})$

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \ne t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

$O(\boldsymbol{n^{2k}})$

$$\begin{aligned}\phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}\end{aligned}$$

$O(\boldsymbol{n^k})$

The variables in the start config, ANDed together

Time complexity of the reduction

- Number of cells = $O(n^{2k})$

$$\phi_{\text{cell}} = \bigwedge_{1\le i,j\le n^k}\left[\left(\bigvee_{s\in C} x_{i,j,s}\right) \wedge \left(\bigwedge_{\substack{s,t\in C\\ s\neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})\right)\right]$$

$O(n^{2k})$

$$\phi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}$$

$O(n^k)$

$$\phi_{\text{accept}} = \bigvee_{1\le i,j\le n^k} x_{i,j,q_{\text{accept}}}$$

The state q_{accept} must appear in some cell

$O(n^{2k})$

Time complexity of the reduction

- Number of cells = $O(\boldsymbol{n^{2k}})$

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

$O(\boldsymbol{n^{2k}})$

$$\begin{aligned} \phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#} \end{aligned}$$

$O(\boldsymbol{n^{k}})$

$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\text{accept}}}$$

$O(\boldsymbol{n^{2k}})$

$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} \big(\text{the } (i,j)\text{-window is legal}\big)$$

$O(\boldsymbol{n^{2k}})$

Time complexity of the reduction

Total:
$O(\boldsymbol{n^{2k}})$

- Number of cells = $O(\boldsymbol{n^{2k}})$

$$\phi_{\text{cell}} = \bigwedge_{1\le i,j\le n^k} \left[\left(\bigvee_{s\in C} x_{i,j,s}\right) \wedge \left(\bigwedge_{\substack{s,t\in C\\ s\neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})\right)\right]$$

$O(\boldsymbol{n^{2k}})$

$$\begin{aligned}\phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}\end{aligned}$$

$O(\boldsymbol{n^{k}})$

$$\phi_{\text{accept}} = \bigvee_{1\le i,j\le n^k} x_{i,j,q_{\text{accept}}}$$

$O(\boldsymbol{n^{2k}})$

$$\phi_{\text{move}} = \bigwedge_{1\le i< n^k,\ 1< j< n^k} \left(\text{the } (i,j)\text{-window is legal}\right)$$

$O(\boldsymbol{n^{2k}})$

To Show Poly Time Mapping Reducibility …

Language A is ***polynomial time mapping reducible***, or simply ***polynomial time reducible***, to language B, written $A \leq_P B$, if a polynomial time computable function $f: \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the ***polynomial time reduction*** of A to B.

To show poly time mapping reducibility:

☑ 1. create **computable fn**,
☑ 2. show that it **runs in poly time**,
☑ 3. then show **forward direction** of mapping red.,
☑ 4. and **reverse direction** (or **contrapositive** of **forward direction**)

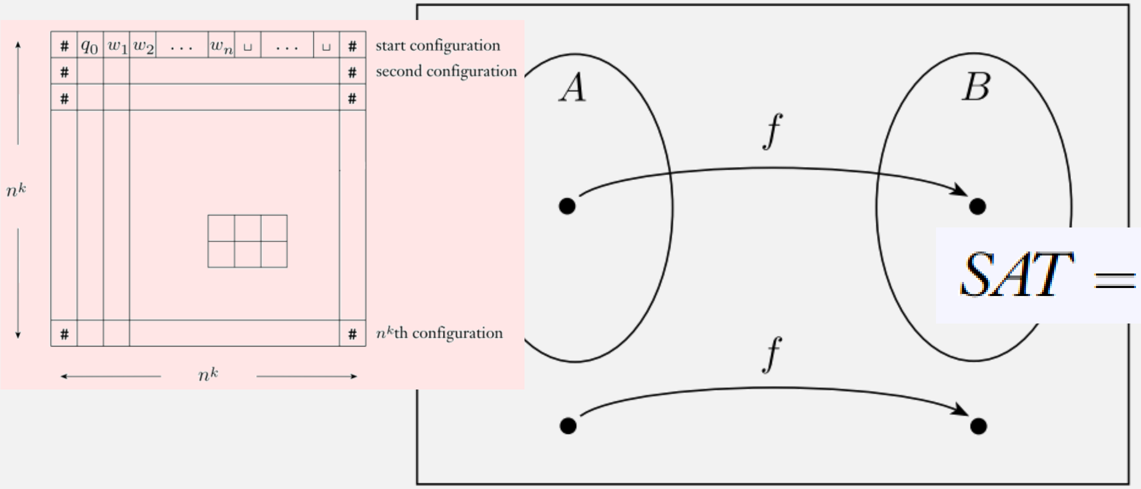
QED: *SAT* is NP-complete

DEFINITION

A language B is ***NP-complete*** if it satisfies two conditions:

☑ **1.** B is in NP, and

☑ **2.** every A in NP is polynomial time reducible to B.



$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

Now it will be much easier to prove that other languages are NP-complete!

THEOREM

known

unknown

Key Thm: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

To use this theorem, C must be in **NP**

DEFINITION

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

Proof:

- Need to show: C is **NP**-complete:
  - it's in **NP** (given), and
  - every lang A in **NP** reduces to C in poly time (must show)
- For every language A in **NP**, reduce $A \rightarrow C$ by:
  - First reduce $A \rightarrow B$ in poly time
    - Can do this because B is **NP**-Complete
  - Then reduce $B \rightarrow C$ in poly time
    - This is given
- Total run time: Poly time + poly time = poly time

If you're not Stephen Cook or Leonid Levin, **use this theorem to prove a language is NP-complete**

THEOREM

Using: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

3 steps to prove a language C is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

To show poly time mapping reducibility:
1. create **computable fn**,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction**
   (or **contrapositive** of **forward direction**)

THEOREM

Using: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

3 steps to prove a language C is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

Example:

Let $C = 3SAT$, to prove $3SAT$ is **NP**-Complete:

1. Show $3SAT$ is in **NP**

Flashback: $3SAT$ is in **NP**

$3SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$

Let $\boldsymbol{n}$ = the number of variables in the formula

Verifier:

On input <ϕ, c>, where c is a possible assignment of variables in ϕ to values:
- Accept if c satisfies ϕ

Running Time: $O(\boldsymbol{n})$

Non-deterministic Decider:

On input <ϕ>, where ϕ is a boolean formula:
- Non-deterministically try all possible assignments in parallel
- Accept if any satisfy ϕ

Running Time: Checking each assignment takes time $O(\boldsymbol{n})$

THEOREM

Using: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

3 steps to prove a language is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

Example:

Let C = *3SAT*, to prove *3SAT* is **NP**-Complete:

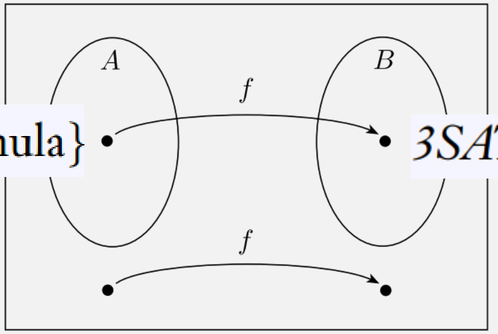
☑ 1. Show *3SAT* is in **NP**

☑ 2. Choose B, the **NP**-complete problem to reduce from: *SAT*

3. Show a poly time mapping reduction from *SAT* to *3SAT*

Flashback: *SAT* is Poly Time Reducible to *3SAT*

$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$



$3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$

Need: poly time computable fn converting a Boolean formula ϕ to 3CNF:

1. Convert ϕ to CNF (an AND of OR clauses)
   a) Use DeMorgan's Law to push negations onto literals

      $\neg(P \vee Q) \iff (\neg P) \wedge (\neg Q)$ $\quad$ $\neg(P \wedge Q) \iff (\neg P) \vee (\neg Q)$ $\quad$ $O(\boldsymbol{n})$

   b) Distribute ORs to get ANDs outside of parens

      $(P \vee (Q \wedge R)) \Leftrightarrow ((P \vee Q) \wedge (P \vee R))$ $\quad$ $O(\boldsymbol{n})$

2. Convert to 3CNF by adding new variables

   $(a_1 \vee a_2 \vee a_3 \vee a_4) \Leftrightarrow (a_1 \vee a_2 \vee z) \wedge (\overline{z} \vee a_3 \vee a_4)$ $\quad$ $O(\boldsymbol{n})$

Remaining step: show iff relation holds …

… easy for formula conversion: each step is already a known "law"

THEOREM

Using: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

3 steps to prove a language is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

Example:

Let C = *3SAT*, to prove *3SAT* is **NP**-Complete:

☑ 1. Show *3SAT* is in **NP**

☑ 2. Choose B, the **NP**-complete problem to reduce from: *SAT*

☑ 3. Show a poly time mapping reduction from *SAT* to *3SAT*

Each NP-complete problem we prove makes it easier to prove the next one!

THEOREM

Using: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

3 steps to prove a language is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

Example:

Let C = ~~*3SAT*~~ ***CLIQUE***, to prove ~~*3SAT*~~ ***CLIQUE*** is **NP**-Complete:

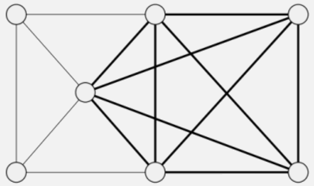
? 1. Show ~~*3SAT*~~ ***CLIQUE*** is in **NP**

? 2. Choose B, the **NP**-complete problem to reduce from: ~~*SAT*~~ ***3SAT***

? 3. Show a poly time mapping reduction from B to C

Flashback:

CLIQUE is in NP



$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

PROOF IDEA The clique is the certificate.

Let n = # nodes in G

c is at most n

PROOF The following is a verifier V for *CLIQUE*.

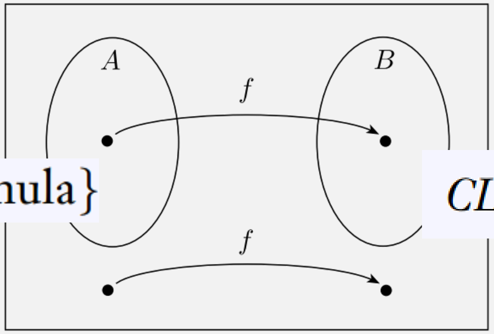
V = "On input $\langle\langle G, k\rangle, c\rangle$:

1. Test whether c is a subgraph with k nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If both pass, *accept*; otherwise, *reject*."

For each node in c, check whether it's in G: $O(n^2)$

For each pair of nodes in c, check whether there's an edge in G: $O(n^2)$

Flashback: 3SAT is polynomial time reducible to CLIQUE.



$3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$

$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

Need: poly time computable fn converting a 3cnf-formula …

Example:

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

- … to a graph containing a clique:
  - Each clause maps to a group of 3 nodes
  - Connect all nodes except:
    - Contradictory nodes
    - Nodes in the same group

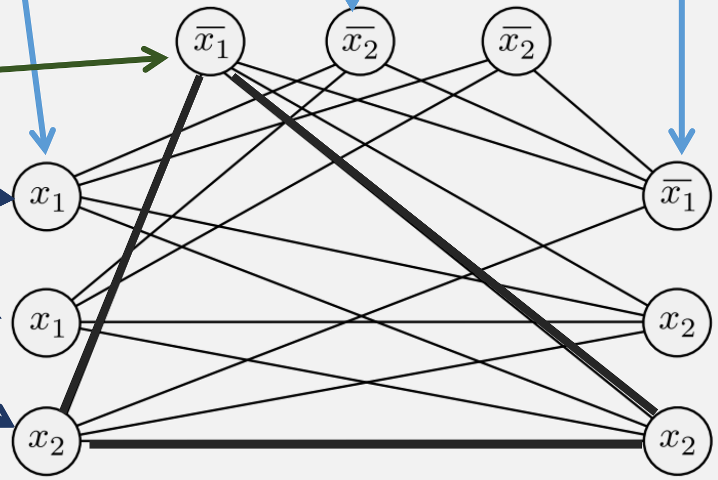
Don't forget iff

⇒ If $\phi \in 3SAT$

- Then each clause has a TRUE literal
  - Those are nodes in the clique!
  - E.g., $x_1 = 0, x_2 = 1$

⇐ If $\phi \notin 3SAT$

- For any assignment, some clause must have a contradiction with another clause
- Then in the graph, some clause's group of nodes won't be connected to another group, preventing the clique



Runs in poly time:
- # literals = # nodes $O(n)$
- # edges poly in # nodes $O(n^2)$

THEOREM

Using: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

3 steps to prove a language is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

Example:

Let C = ~~*3SAT*~~ ***CLIQUE***, to prove ~~*3SAT*~~ ***CLIQUE*** is **NP**-Complete:

☑ 1. Show ~~*3SAT*~~ ***CLIQUE*** is in **NP**

☑ 2. Choose B, the **NP**-complete problem to reduce from: ~~SAT~~ ***3SAT***

☑ 3. Show a poly time mapping reduction from B to C

NP-Complete problems, so far

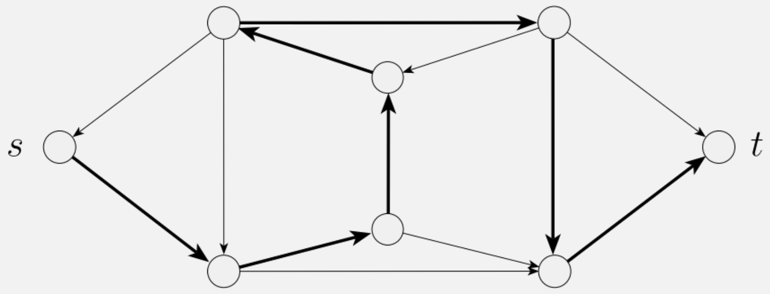
- $SAT = \{\langle\phi\rangle | \ \phi \text{ is a satisfiable Boolean formula}\}$ (Cook-Levin Theorem)
- $3SAT = \{\langle\phi\rangle | \ \phi \text{ is a satisfiable 3cnf-formula}\}$ (reduced *SAT* to *3SAT*)
- $CLIQUE = \{\langle G, k\rangle | \ G \text{ is an undirected graph with a } k\text{-clique}\}$ (reduced *3SAT* to *CLIQUE*)

Each NP-complete problem we prove makes it easier to prove the next one!

Flashback: The *HAMPATH* Problem

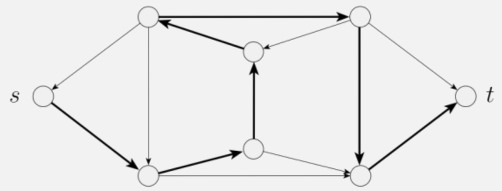
$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

- A Hamiltonian path goes through every node in the graph



- The **Search** problem:
  - Exponential time (brute force) algorithm:
    - Check all possible paths and see if any connect s and t using all nodes $O(\boldsymbol{n}^{\boldsymbol{n}})$
  - Polynomial time algorithm:
    - We don't know if there is one!!!
- The **Verification** problem:
  - Still $O(\boldsymbol{n}^{\boldsymbol{2}})$!
  - *HAMPATH* is polynomially verifiable, but not polynomially decidable
  - i.e., It's in in **NP** but not known to be in **P**

Theorem: *HAMPATH* is NP-complete



$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

THEOREM

Using: If B is NP-complete and $B \leq_{\mathrm{P}} C$ for C in NP, then C is NP-complete.

3 steps to prove a language is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

Theorem: *HAMPATH* is NP-complete

$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$

To prove *HAMPATH* is **NP**-complete:

☑ 1. Show *HAMPATH* is in **NP** (in HW9)

? 2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

3. Show a poly time mapping reduction from *B* to *HAMPATH*

Theorem: *HAMPATH* is NP-complete

$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$

To prove *HAMPATH* is **NP**-complete:

☑ 1. Show *HAMPATH* is in **NP** (in HW9)

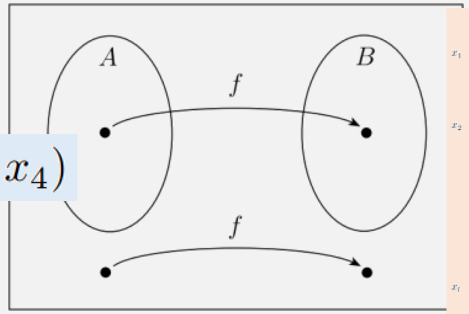
☑ 2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

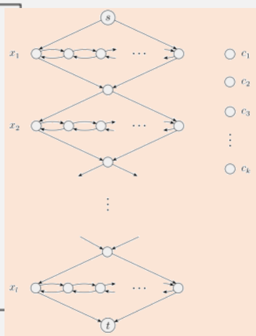
? 3. Show a poly time mapping reduction from *3SAT* to *HAMPATH*

To show poly time mapping reducibility:
1. create **computable fn**,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction** (or **contrapositive** of **forward direction**)

???

$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$



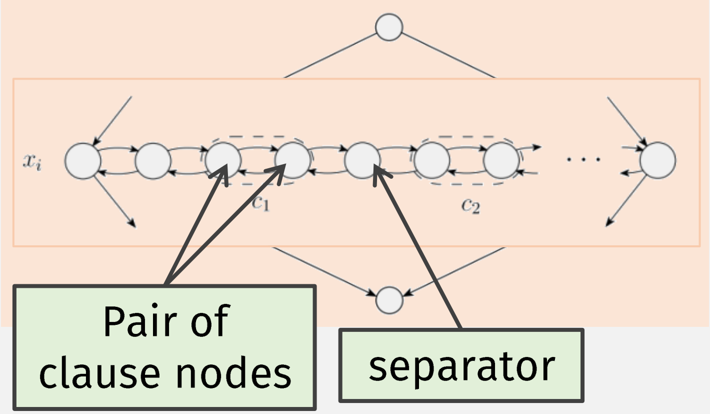


Computable Fn: Formula (blue) → Graph (orange)

Example input: $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$

k = # clauses

- Clause → (extra) single nodes, Total = k
- Variable → diamond-shaped graph "gadget"
  - Clause → 2 "connector" nodes + separator
  - Total = $3k+1$ "connector" nodes per "gadget"

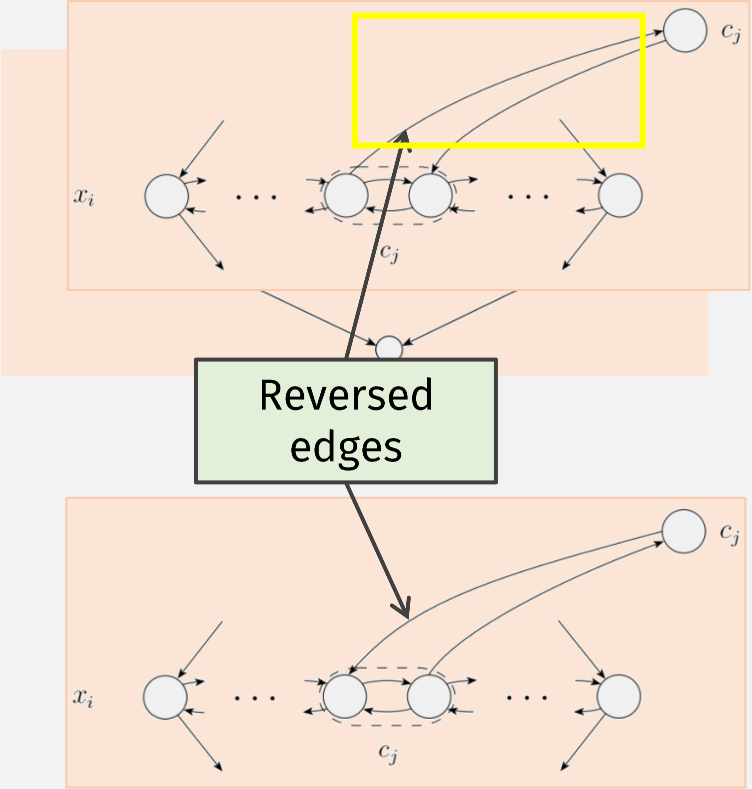


Computable Fn: Formula (blue) → Graph (orange)

Example input: $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$

k = # clauses

- Clause → (extra) single nodes, Total = k
- Variable → diamond-shaped graph "gadget"
  - Clause → 2 "connector" nodes + separator
  - Total = $3k+1$ "connector" nodes per "gadget"
- Lit x_i in clause c_j → c_j node edges in gadget x_i
- Lit $\overline{x_i}$ in clause c_j → c_j edges in gadget x_i (rev)



Theorem: *HAMPATH* is NP-complete

$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$

To prove *HAMPATH* is **NP**-complete:

☑ 1. Show *HAMPATH* is in **NP**

☑ 2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

? 3. Show a poly time mapping reduction from *3SAT* to *HAMPATH*

To show poly time <u>mapping reducibility</u>:

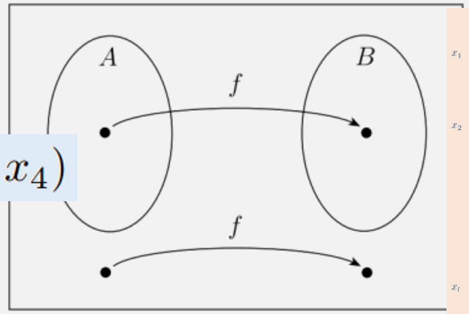
☑ 1. create **computable fn**,

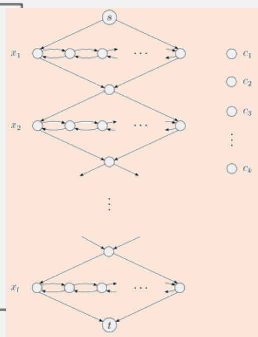
➡ 2. show that it **runs in poly time**,

3. then show **forward direction** of mapping red.,

4. and **reverse direction**
   (or **contrapositive** of **forward direction**)

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$





Polynomial Time?

TOTAL: $O(k^2)$

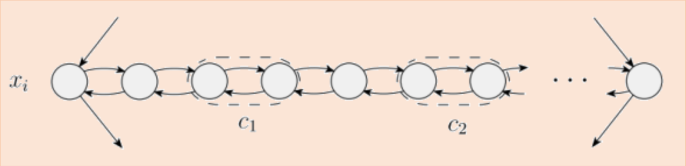
Example input: $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$

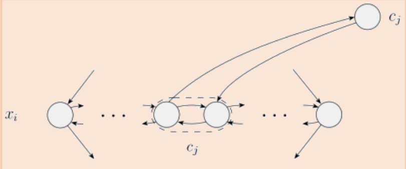
k = # clauses = at most $\mathbf{3}\boldsymbol{k}$ variables

- Clause → (extra) single nodes 

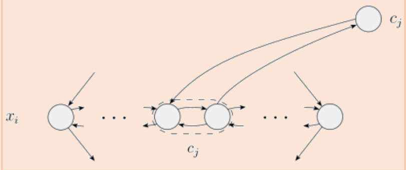
$O(\boldsymbol{k})$

- Variable → diamond-shaped graph "gadget" $O(\boldsymbol{k}^2)$
  - Clause → 2 "connector" nodes + separator
  - Total = $3k$+1 "connector" nodes per "gadget"



- Lit x_i in clause c_j → c_j node edges in gadget x_i 

$O(\boldsymbol{k})$

- Lit $\overline{x_i}$ in clause c_j → c_j edges in gadget x_i (rev) 

$O(\boldsymbol{k})$

Theorem: *HAMPATH* is NP-complete

$HAMPATH = \{\langle G, s, t\rangle \mid G$ is a directed graph with a Hamiltonian path from s to $t\}$

To prove *HAMPATH* is **NP**-complete:

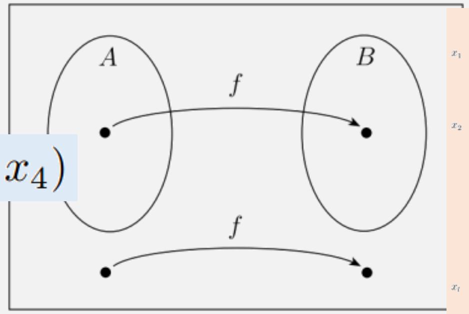
☑ 1. Show *HAMPATH* is in **NP**

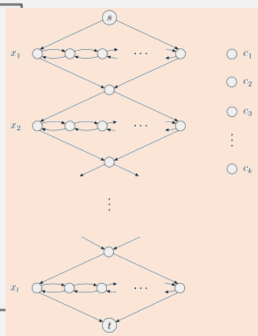
☑ 2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

? 3. Show a poly time mapping reduction from *3SAT* to *HAMPATH*

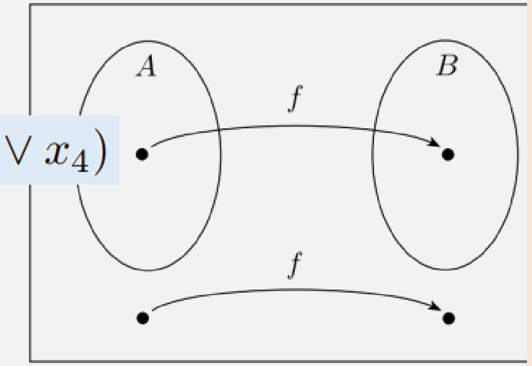
To show poly time mapping reducibility:
- ☑ 1. create **computable fn**,
- ☑ 2. show that it **runs in poly time**,
- → 3. then show **forward direction** of mapping red.,
- 4. and **reverse direction** (or **contrapositive** of **forward direction**)

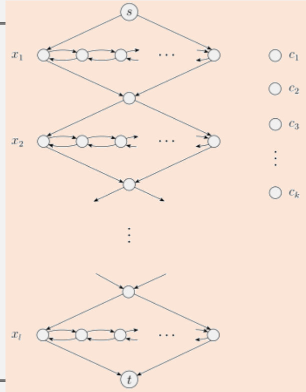
$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$





$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$



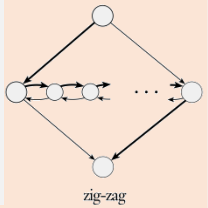


Want: Satisfiable 3cnf formula ⇔ graph with Hamiltonian path

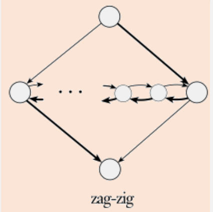
⇒ If there is satisfying assignment, then Hamiltonian path exists

These hit all nodes except extra c_js

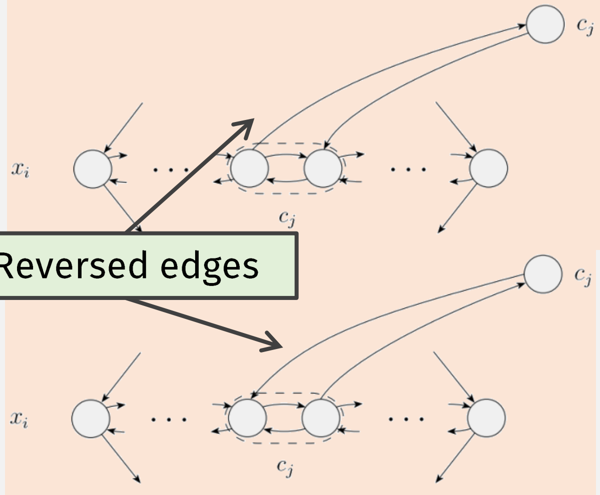
- x_i = TRUE → Hampath "zig-zags" gadget x_i



- x_i = FALSE → Hampath "zag-zigs" gadget x_i



- Lit x_i makes clause c_j TRUE → "detour" to c_j in gadget x_i
- Lit $\overline{x_i}$ makes clause c_j TRUE → "detour" to c_j in gadget x_i

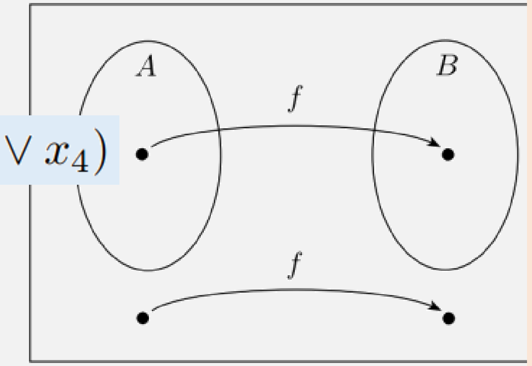


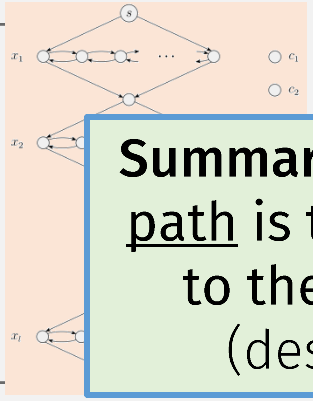
Now path goes through every node

Every clause must be TRUE so path hits all c_j nodes

- And edge directions align with TRUE/FALSE assignments

$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$

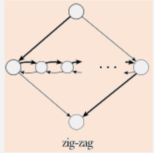


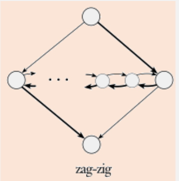


Summary: the <u>only possible Ham. path</u> is the one that corresponds to the satisfying assignment (described on prev slide)

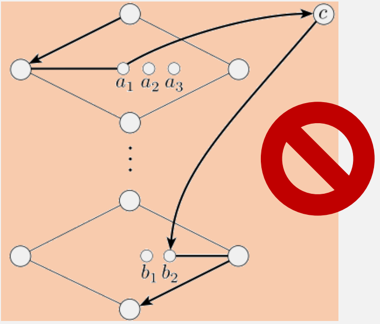
<u>Want</u>: Satisfiable 3cnf formula ⇔ graph with Hamiltonian path

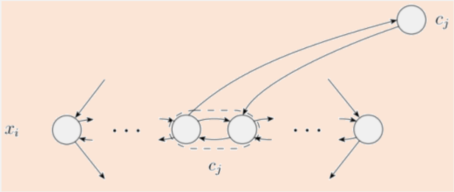
⇐ if output has Ham. path, then input had Satisfying assignment

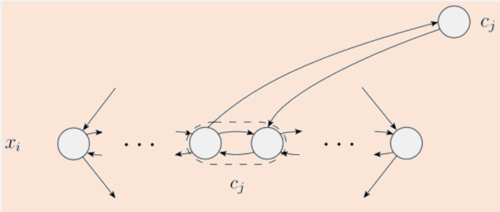
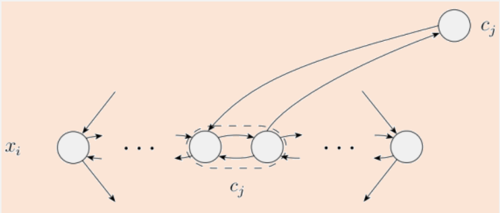




- A Hamiltonian path must choose to either zig-zag or zag-zig gadgets
- Ham path can only hit "detour" c_j nodes <u>by coming right back</u>
- Otherwise, it will miss some nodes





-  gadget x_i "detours" from left to right → x_i = TRUE
-  gadget x_i "detours" from right to left → x_i = FALSE

Theorem: *HAMPATH* is NP-complete

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

To prove *HAMPATH* is **NP**-complete:

☑ 1. Show *HAMPATH* is in **NP**

☑ 2. Choose *B*, the **NP**-complete problem to reduce from *3SAT*

☑ 3. Show a poly time mapping reduction from *3SAT* to *HAMPATH*

To show poly time mapping reducibility:

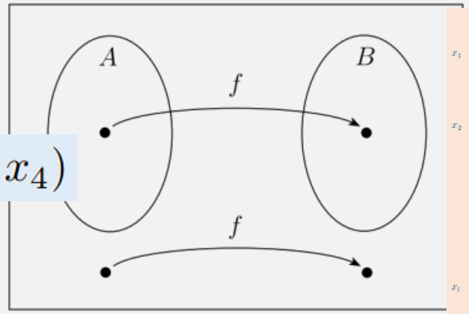
☑ 1. create **computable fn**,

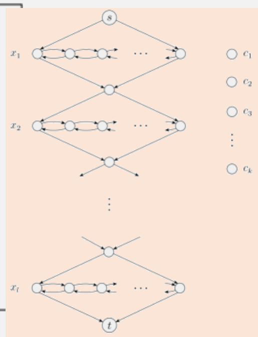
☑ 2. show that it **runs in poly time**,

☑ 3. then show **forward direction** of mapping red.,

☑ 4. and **reverse direction**
(or **contrapositive** of **forward direction**)

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$





Theorem: *UHAMPATH* is NP-complete

un

$$UHAMPATH = \{\langle G, s, t \rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

To prove *UHAMPATH* is **NP**-complete:

1. Show *UHAMPATH* is in **NP**
2. Choose the **NP**-complete problem to reduce from *HAMPATH*
3. Show a poly time mapping reduction from *???* to *UHAMPATH*

Theorem: *UHAMPATH* is NP-complete

$UHAMPATH = \{\langle G, s, t\rangle \mid G$ is a ~~directed~~ undirected graph with a Hamiltonian path from s to $t\}$

To prove *UHAMPATH* is **NP**-complete:

☑ 1. Show *UHAMPATH* is in **NP**

☑ 2. Choose the **NP**-complete problem to reduce from *HAMPATH*

➡ 3. Show a poly time mapping reduction from *HAMPATH* to *UHAMPATH*

Theorem: *UHAMPATH* is NP-complete

$UHAMPATH = \{\langle G, s, t \rangle | \ G \text{ is a } \text{un}\text{directed graph with a Hamiltonian path from } s \text{ to } t\}$

Need: Computable function from *HAMPATH* to *UHAMPATH*

Naïve Idea: Make all directed edges undirected?

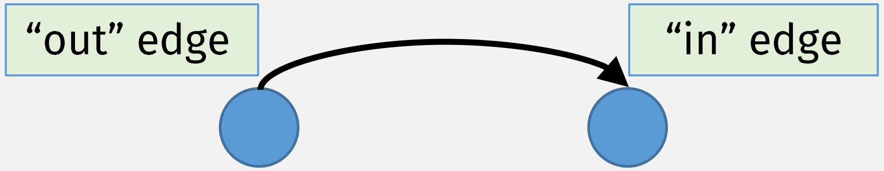
- Doesn't work!
- But we would create some paths that didn't exist before

Theorem: *UHAMPATH* is NP-complete

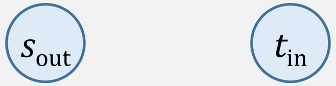
$UHAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a undirected graph with a Hamiltonian path from } s \text{ to } t\}$

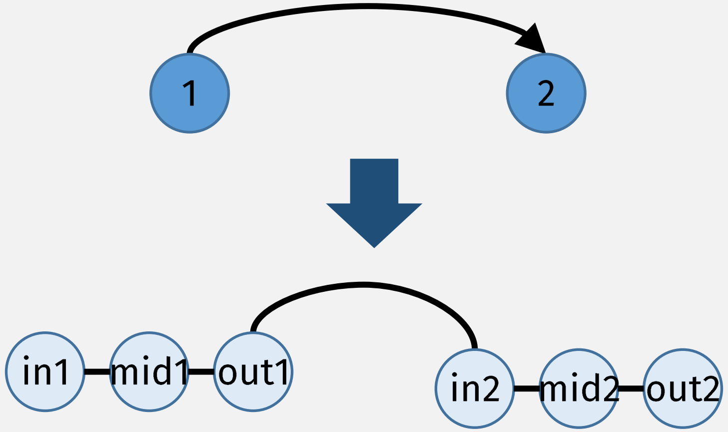
Need: Computable function from *HAMPATH* to *UHAMPATH*

Better Idea:



- Distinguish "in" vs "out" edges
- Nodes (directed) → 3 Nodes (undirected): in/mid/out
  - Connect in/mid/out with edges
  - Directed edge (u, v) → $(u_{\text{out}}, v_{\text{in}})$
- Except: s → s_{out}, t → t_{in} only



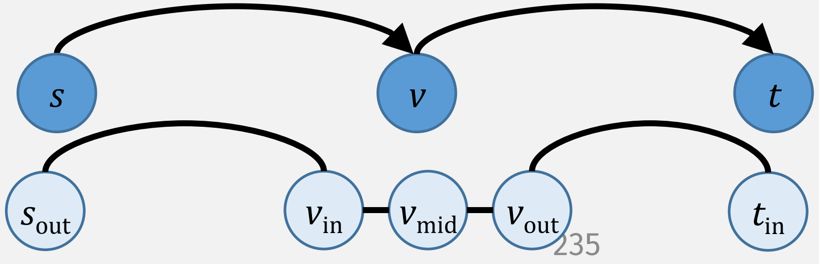


Theorem: *UHAMPATH* is NP-complete

$UHAMPATH = \{\langle G, s, t\rangle \mid G$ is a ~~directed~~ undirected graph with a Hamiltonian path from s to $t\}$

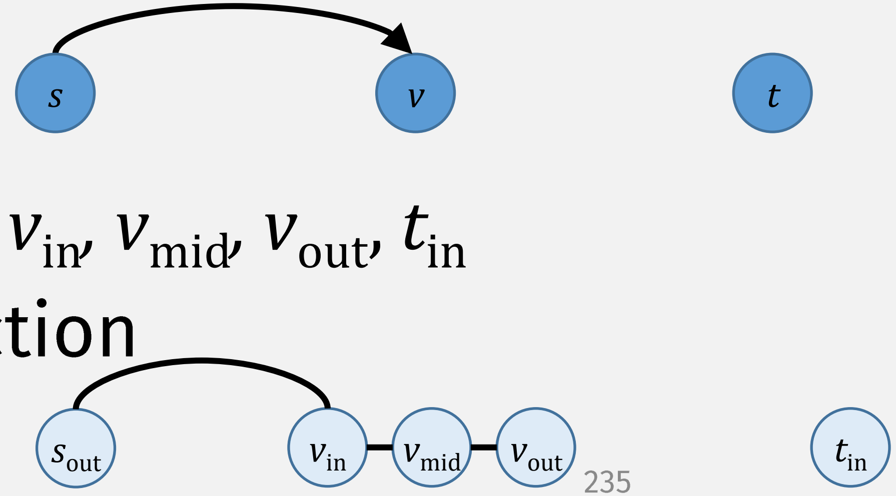
Need: Computable function from *HAMPATH* to *UHAMPATH*

$\Rightarrow$



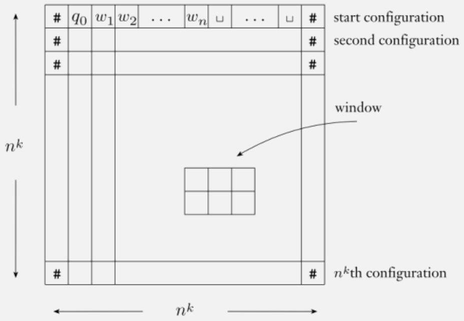
- If there was a directed path s, v, t ...
- ... then there is an undirected path $s_{out}, v_{in}, v_{mid}, v_{out}, t_{in}$

$\Leftarrow$



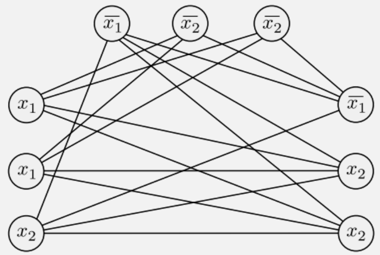
- If there was no directed path s, v, t ...
- ... then there is no undirected path $s_{out}, v_{in}, v_{mid}, v_{out}, t_{in}$
- Because there will be a missing connection

NP-Complete problems, so far

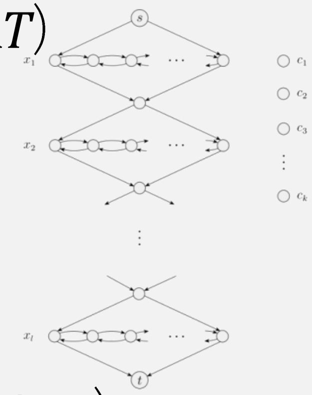


- $SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$ (Cook-Levin Theorem)

- $3SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$ (reduce from *SAT*)



- $CLIQUE = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$ (reduce from 3*SAT*)



- $HAMPATH = \{\langle G, s, t \rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$ (reduce from 3*SAT*)

- $UHAMPATH = \{\langle G, s, t \rangle \mid G \text{ is a undirected graph with a Hamiltonian path from } s \text{ to } t\}$ (reduce from *HAMPATH*)

Check-in Quiz 11/17

On gradescope