Intro to Complexity Theory

Computability theory (1930s - 1950s):

Is A decidable?

Complexity theory (1960s - present):

Is A decidable with restricted resources?

(time/memory/…)

Example: Let $A = \{\mathrm{a}^k\mathrm{b}^k \mid k \geq 0\}$.

Q: How many steps are needed to decide A?

Depends on the input.

We give an <u>upper bound</u> for all inputs of length n.

Called "worst-case complexity".

steps to decide $A = \{\mathrm{a}^k\mathrm{b}^k \mid k \geq 0\}$

Theorem: A 1-tape TM M can decide A where, on inputs of length n, M uses <u>at most cn^2 steps, for some fixed constant c</u>.

Terminology: M uses $\underline{O(n^2)}$ steps.

Proof: $M =$ "On input w

1. Scan input to check if $w \in \mathrm{a}^*\mathrm{b}^*$, *reject* if not.
2. Repeat until all crossed off.
   Scan tape, crossing off one a and one b.
   Reject if only a's or only b's remain.
3. Accept if all crossed off. "

Analysis:

$O(n)$ steps

$+O(n)$ iterations

$\times\, O(n)$ steps

$O(n) + O(n^2)$ steps

$= O(n^2)$ steps

Check-in 12.1

How much improvement is possible in the bound for this theorem about 1-tape TMs deciding A?

(a) $O(n^2)$ is best possible.

(b) $O(n \log n)$ is possible.

(c) $O(n)$ is possible.

Check-in 12.1

Deciding $A = \{a^k b^k | \, k \geq 0\}$ faster

Theorem: A 1-tape TM M can decide A by using $O(n \log n)$ steps.

Proof:

$M =$ "On input w

1. Scan tape to check if $w \in a^* b^*$. *Reject* if not.
2. Repeat until all crossed off.
   Scan tape, crossing off every other a and b.
   Reject if even/odd parities disagree.
3. Accept if all crossed off. "

Analysis:

$O(n)$ steps

$+O(\log n)$ iterations

$\times\, O(n)$ steps

$O(n) + O(n \log n)$ steps

$= O(n \log n)$ steps

	Parities
a's	
b's	

Further improvement? Not possible.

Theorem: A 1-tape TM M cannot decide A by using $o(n \log n)$ steps.

You are not responsible for knowing the proof.

Deciding $A = \{\mathrm{a}^k\mathrm{b}^k \mid k \geq 0\}$ even faster

Theorem: A multi-tape TM M can decide A using $O(n)$ steps.

$M =$ "On input w
1. Scan input to check if $w \in \mathrm{a}^*\mathrm{b}^*$, *reject* if not.
2. Copy a's to second tape.
3. Match b's with a's on second tape.
4. *Accept* if match, else *reject*. "

Analysis:

$O(n)$ steps

$+O(n)$ steps

$+O(n)$ steps

$= O(n)$ steps

Model Dependence

Number of steps to decide $A = \{\mathrm{a}^k\mathrm{b}^k \mid k \geq 0\}$ depends on the model.

- **1-tape TM:** $O(n \log n)$
- **Multi-tape TM:** $O(n)$

Computability theory: model independence (Church-Turing Thesis)
Therefore model choice doesn't matter. Mathematically nice.

Complexity Theory: model dependence
But dependence is low (polynomial) for reasonable deterministic models.
We will focus on questions that do not depend on the model choice.

So… we will continue to use the 1-tape TM as the basic model for complexity.

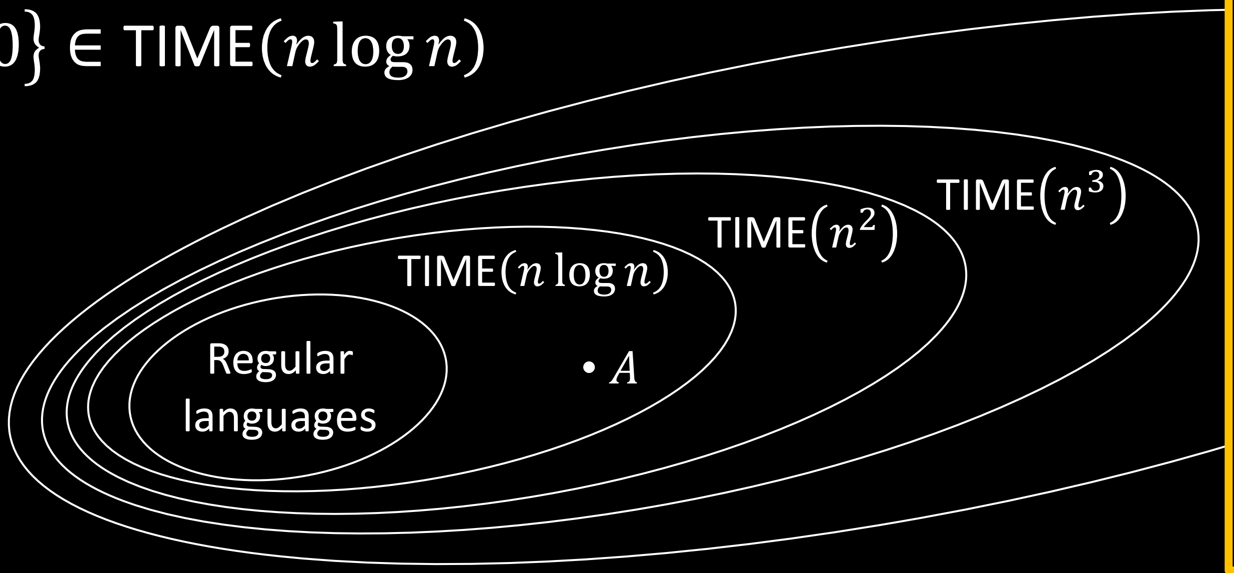
TIME Complexity Classes

Defn: Let $t\colon \mathbb{N} \to \mathbb{N}$. Say TM M runs in time $t(n)$ if M always halts within $t(n)$ steps on all inputs of length n.

Defn: $\text{TIME}\big(t(n)\big) = \{B \mid$ some deterministic 1-tape TM M decides B and M runs in time $O\big(t(n)\big)\}$

Example:

$A = \{\mathrm{a}^k\mathrm{b}^k \mid k \geq 0\} \in \text{TIME}(n \log n)$



Check-in 12.2

Let $B = \{ww^{\mathcal{R}} \mid w \in \{\mathrm{a},\mathrm{b}\}^*\}$.

What is the smallest function t such that $B \in \text{TIME}\big(t(n)\big)$?

(a) $O(n)$

(b) $O(n \log n)$

(c) $O(n^2)$

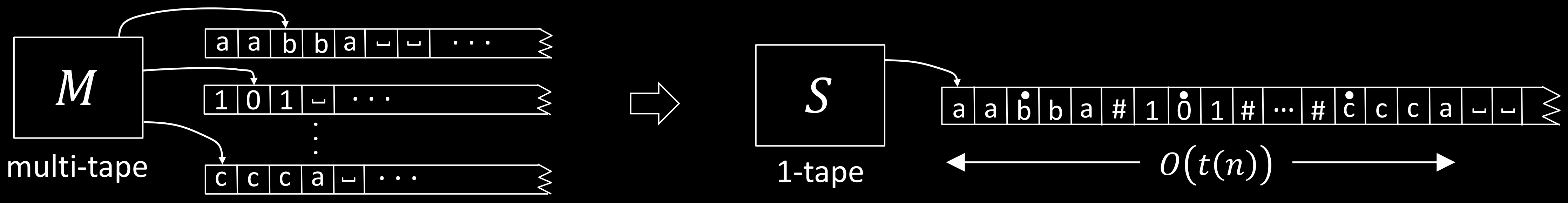
(d) $O(n^3)$

Multi-tape vs 1-tape time

Theorem: Let $t(n) \geq n$.

If a multi-tape TM decides B in time $t(n)$, then $B \in \text{TIME}\big(t^2(n)\big)$.

Proof: Analyze conversion of multi-tape to 1-tape TMs.



To simulate 1 step of M's computation, S uses $O\big(t(n)\big)$ steps.

So total simulation time is $O\big(t(n) \times t(n)\big) = O\big(t^2(n)\big)$.

Similar results can be shown for other reasonable deterministic models.

Relationships among models

Informal Defn: Two models of computation are <u>polynomially related</u> if each can simulate the other with a polynomial overhead:
So $t(n)$ time $\rightarrow t^k(n)$ time on the other model, for some k.

All reasonable deterministic models are polynomially related.

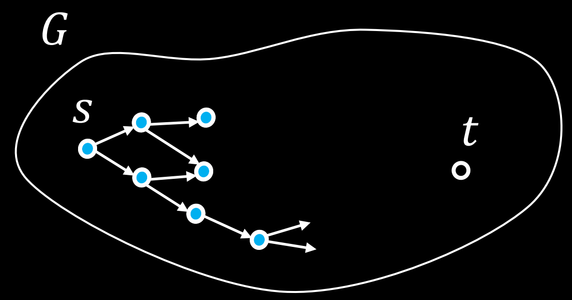
- 1-tape TMs
- multi-tape TMs
- multi-dimensional TMs
- random access machine (RAM)
- cellular automata

The Class P

Defn: $\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$

$= $ polynomial time decidable languages

- Invariant for all reasonable deterministic models
- Corresponds roughly to realistically solvable problems



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

Theorem: $PATH \in \mathrm{P}$

Proof: $M =$ "On input $\langle G, s, t\rangle$

1. Mark s
2. Repeat until nothing new is marked:
   For each marked node x:
   Scan G to mark all y where (x, y) is an edge
3. *Accept* if t is marked. *Reject* if not.

$\leq n$ iterations

$\times \leq n$ iterations

$\times \; O(n^2)$ steps

$O(n^4)$ steps

To show polynomial time: Each stage should be clearly polynomial and the total number of steps polynomial.

$PATH$ and $HAMPATH$

Example: $HAMPATH = \{\langle G, s, t\rangle |$ G is a directed graph with a path from s to t and the path goes through every node of G $\}$

Called a Hamiltonian path

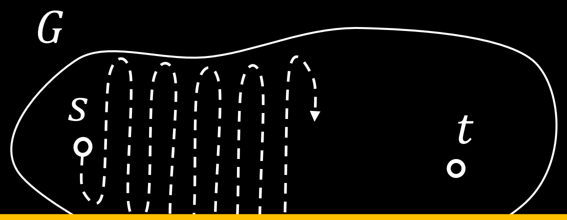
Recall Theorem: $PATH \in$ P

Question: $HAMPATH \in$ P ?

"On input $\langle G, s, t\rangle$

1. Let m be the number of nodes in G.
2. For each path of length m in G:
   test if m is a Hamiltonian path from s to t.
   Accept if yes.
3. *Reject* if all paths fail."

May be $m! > 2^m$ paths of length m
so algorithm is exponential time
not polynomial time.



Check-in 12.3

Is $HAMPATH \in$ P ?

(a) Definitely Yes. You have a polynomial-time algorithm.

(b) Probably Yes. It should be similar to showing $PATH \in$ P.

(c) Toss up.

(d) Probably No. Hard to beat the exponential algorithm.

(e) Definitely No. You can prove it!

Check-in 12.3

Nondeterministic Complexity

In a nondeterministic TM (NTM) decider, all branches halt on all inputs.

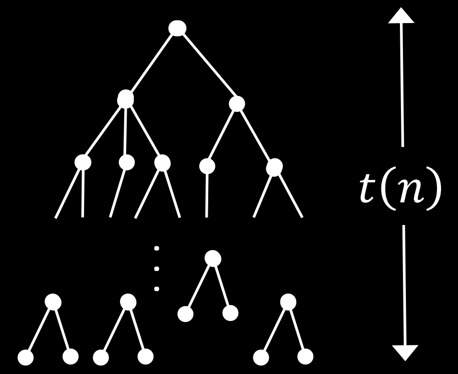
Defn: An NTM runs in time $t(n)$ if all branches halt within $t(n)$ steps on all inputs of length n.

Defn: $\text{NTIME}\big(t(n)\big) = \{B \mid$ some 1-tape NTM decides B and runs in time $O\big(t(n)\big)$ $\}$

Defn: $\textbf{NP} = \bigcup_k \text{NTIME}(n^k)$
$=$ nondeterministic polynomial time decidable languages

- Invariant for all reasonable nondeterministic models
- Corresponds roughly to easily verifiable problems

Computation tree for NTM on input w.



all branches halt within $t(n)$ steps

$HAMPATH \in$ NP

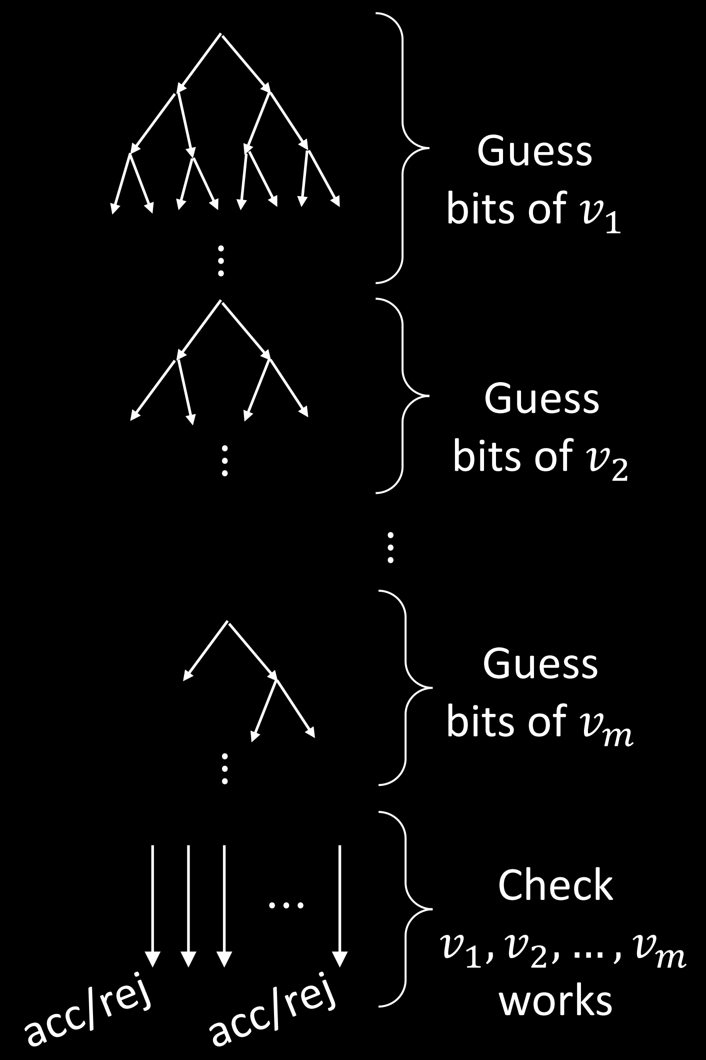
Theorem: $HAMPATH \in$ NP

Proof:

"On input $\langle G, s, t\rangle$ (Say G has m nodes.)

1. Nondeterministically write a sequence $v_1, v_2, \ldots, v_m$ of m nodes.
2. *Accept* if $v_1 = s$
   $v_m = t$
   each (v_i, v_{i+1}) is an edge
   and no v_i repeats.
3. *Reject* if any condition fails."

Computation of M on $\langle G, s, t\rangle$



$COMPOSITES \in$ NP

Defn: $COMPOSITES = \{x \mid x$ is not prime and x is written in binary$\}$
$= \{x \mid x = yz$ for integers $y, z > 1,\ x$ in binary$\}$

Theorem: $COMPOSITES \in$ NP

Proof: "On input x

1. Nondeterministically write y where $1 < y < x$.
2. *Accept* if y divides x with remainder 0.
   Reject if not."

Note: Using base 10 instead of base 2 wouldn't matter because can convert in polynomial time.

Bad encoding: write number k in unary: $1^k = \overbrace{111\cdots1}^{k}$, exponentially longer.

Theorem (2002): $COMPOSITES \in$ P

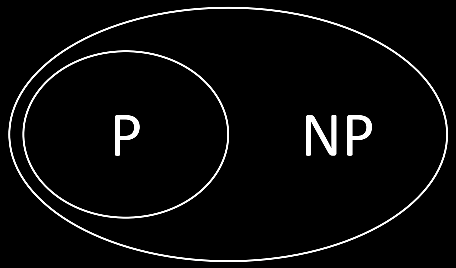
We won't cover this proof.

Intuition for P and NP

NP = All languages where can <u>verify</u> membership quickly
P = All languages where can <u>test</u> membership quickly

Examples of quickly verifying membership:
- $HAMPATH$: Give the Hamiltonian path.
- $COMPOSITES$: Give the factor.

The <u>Hamiltonian path</u> and the <u>factor</u> are called ***short certificates*** of membership.



Check-in 14.1

Let $\overline{HAMPATH}$ be the complement of $HAMPATH$.

So $\langle G, s, t\rangle \in \overline{HAMPATH}$ if G does <u>not</u> have a Hamiltonian path from s to t.

Is $\overline{HAMPATH} \in$ NP?

(a) Yes, we can invert the accept/reject output of the NTM for $HAMPATH$.

(b) No, we cannot give a short certificate for a graph not to have a Hamiltonian path.

(c) I don't know.

Check-in 14.1

Recall A_{CFG}

Recall: $A_{\text{CFG}} = \{\langle G, w\rangle \mid G$ is a CFG and $w \in L(G)\}$

Theorem: A_{CFG} is decidable

Proof: $D_{\text{A-CFG}} =$ "On input $\langle G, w\rangle$

1. Convert G into Chomsky Normal Form.
2. Try all derivations of length $2|w| - 1$.
3. *Accept* if any generate w. *Reject* if not.

Chomsky Normal Form (CNF):

A → BC

B → b

Let's always assume G is in CNF.

Theorem: $A_{\text{CFG}} \in$ NP

Proof: "On input $\langle G, w\rangle$

1. Nondeterministically pick some derivation of length $2|w| - 1$.
2. *Accept* if it generates w. *Reject* if not.

Attempt to show $A_{\mathrm{CFG}} \in \mathrm{P}$

Theorem: $A_{\mathrm{CFG}} \in \mathrm{P}$

Proof attempt:

Recursive algorithm C tests if G generates w, starting at any specified variable R.

C = "On input $\langle G, w, \mathrm{R}\rangle$

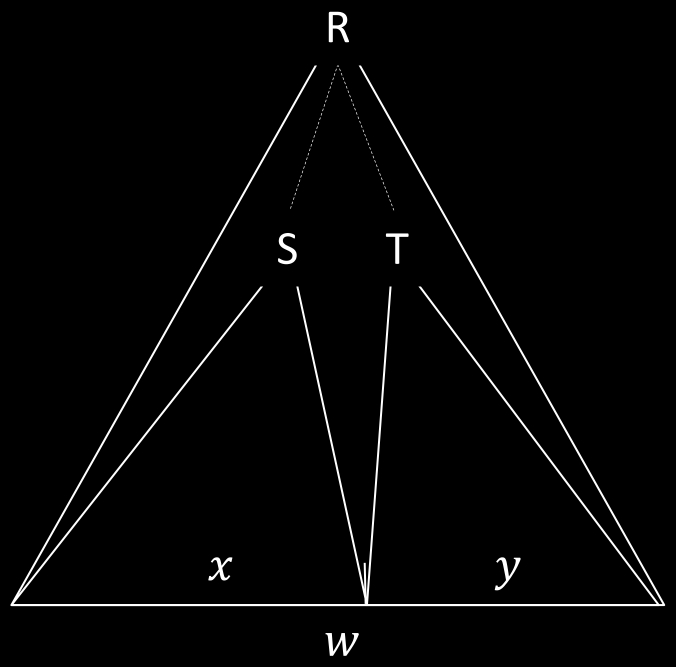
1. For each way to divide $w = xy$ and for each rule $\mathrm{R} \to \mathrm{ST}$
2. Use C to test $\langle G, x, \mathrm{S}\rangle$ and $\langle G, y, \mathrm{T}\rangle$
3. *Accept* if both accept
4. *Reject* if none of the above accepted."

Then decide A_{CFG} by starting from G's start variable.

C is a correct algorithm, but it takes non-polynomial time.
(Each recursion makes $O(n)$ calls and depth is roughly $\log n$.)

Fix: Use recursion + memory called *Dynamic Programming* (DP)

Observation: String w of length n has $O(n^2)$ substrings $w_i \cdots w_j$ therefore there are only $O(n^2)$ possible sub-problems $\langle G, x, \mathrm{S}\rangle$ to solve.



DP shows $A_{\mathrm{CFG}} \in \mathrm{P}$

Theorem: $A_{\mathrm{CFG}} \in \mathrm{P}$

Proof : Use DP (Dynamic Programming) = recursion + memory.

"memoization"

$D =$ "On input $\langle G, w, \mathrm{R}\rangle$

0. If previously solved $\langle G, w, \mathrm{R}\rangle$ then answer same, else continue.
1. For each way to divide $w = xy$ and for each rule $\mathrm{R} \rightarrow \mathrm{ST}$
2. Use D to test $\langle G, x, \mathrm{S}\rangle$ and $\langle G, y, \mathrm{T}\rangle$
3. *Accept* if both accept
4. *Reject* if none of the above accepted."

Then decide A_{CFG} by starting from G's start variable.

same as before

Total number of calls is $O(n^2)$ so time used is polynomial.

Alternately, solve all smaller sub-problems first: "bottom up"

Check-in 14.2

Suppose B is a CFL.
Does that imply that $B \in \mathrm{P}$?

(a) Yes

(b) No.

Check-in 14.2

$A_{\mathrm{CFG}} \in$ P & Bottom-up DP

Theorem: $A_{\mathrm{CFG}} \in$ P

Proof : Use bottom-up DP.

$D =$ "On input $\langle G, w \rangle$

1. For each w_i and variable R
   Solve $\langle G, w_i, \mathrm{R} \rangle$ by checking if R $\rightarrow w_i$ is a rule.

   Solve for substrings of length 1

2. For $k = 2, \ldots, n$ and each substring u of w where $|u| = k$ and variable R
   Solve $\langle G, u, \mathrm{R} \rangle$ by checking for each R $\rightarrow$ ST and each division $u = xy$
   if both $\langle G, x, \mathrm{S} \rangle$ and $\langle G, y, \mathrm{T} \rangle$ were positive.

   Solve for substrings of length k by using previous answers for substrings of length $< k$.

3. *Accept* if $\langle G, w, \mathrm{S} \rangle$ is positive where S is the original start variable.
4. *Reject* if not."

Total number of calls is $O(n^2)$ so time used is polynomial.

Often, bottom-up DP is shown as filling out a table.

Satisfiability Problem

Defn: A ***Boolean formula*** ϕ has Boolean variables (TRUE/FALSE values) and Boolean operations AND ($\wedge$), OR ($\vee$), and NOT ($\neg$).

Defn: ϕ is ***satisfiable*** if ϕ evaluates to TRUE for some assignment to its variables.
Sometimes we use 1 for True and 0 for False.

Example: Let $\phi = (x \vee y) \wedge (\overline{x} \vee \overline{y})$ (Notation: $\overline{x}$ means $\neg x$)
Then ϕ is satisfiable (x=1, y=0)

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

Theorem (Cook, Levin 1971): $SAT \in$ P $\rightarrow$ P = NP
Proof method: polynomial time (mapping) reducibility

Check-in 14.3

Is $SAT \in$ NP?

(a) Yes.

(b) No.

(c) I don't know.

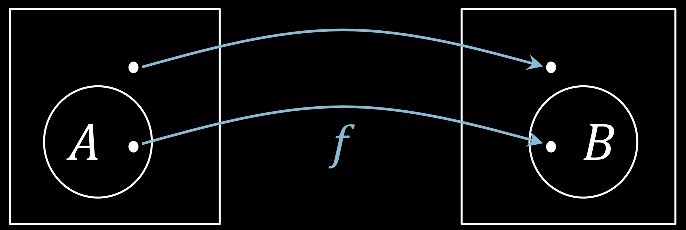
(d) No one knows.

Check-in 14.3

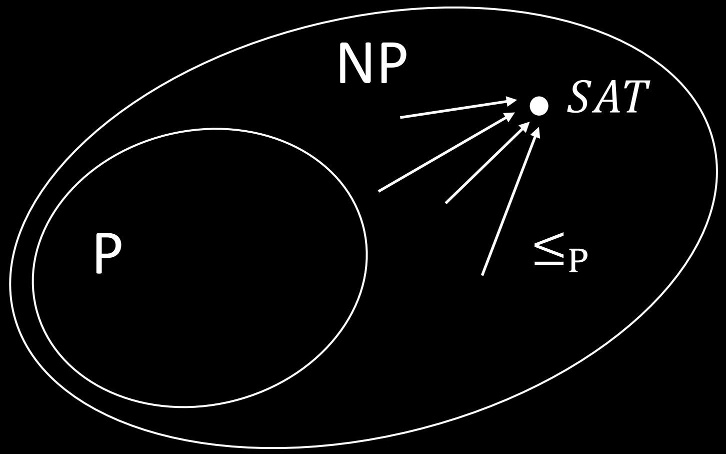
Polynomial Time Reducibility

Defn: A is polynomial time reducible to B $(A \leq_{\mathrm{P}} B)$ if $A \leq_{\mathrm{m}} B$ by a reduction function that is computable in polynomial time.

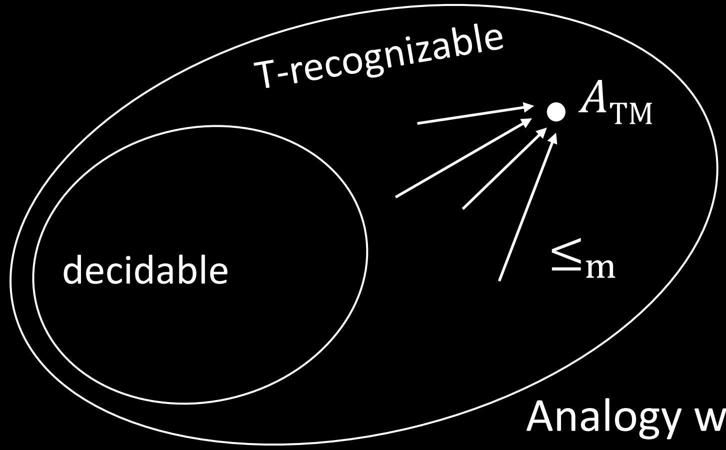
Theorem: If $A \leq_{\mathrm{P}} B$ and $B \in$ P then $A \in$ P.



f is computable in polynomial time



Idea to show $SAT \in$ P $\rightarrow$ P = NP



Analogy with A_{TM}

$\leq_{\mathrm{P}}$ Example: $3SAT$ and $CLIQUE$

Defn: A Boolean formula ϕ is in Conjunctive Normal Form (CNF) if it has the form $\phi = \underbrace{(x \vee \overline{y} \vee z)}_{\text{clause}} \wedge \underbrace{(\overline{x} \vee \overline{s} \vee z \vee u)}_{\text{clause}} \wedge \cdots \wedge (\overline{z} \vee \overline{u})$

literals

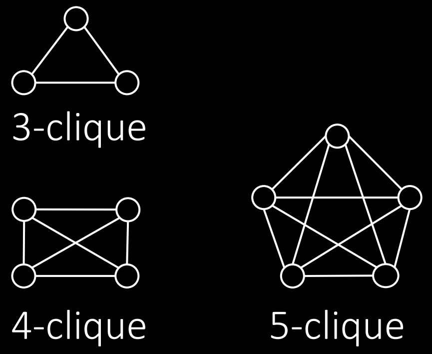
Literal: a variable or a negated variable
Clause: an OR ($\vee$) of literals.
CNF: an AND ($\wedge$) of clauses.
3CNF: a CNF with exactly 3 literals in each clause.

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

Defn: A k-clique in a graph is a subset of k nodes all directly connected by edges.

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

Will show: $3SAT \leq_{\mathrm{P}} CLIQUE$



$3SAT \leq_{\mathrm{P}} CLIQUE$

Theorem: $3SAT \leq_{\mathrm{P}} CLIQUE$

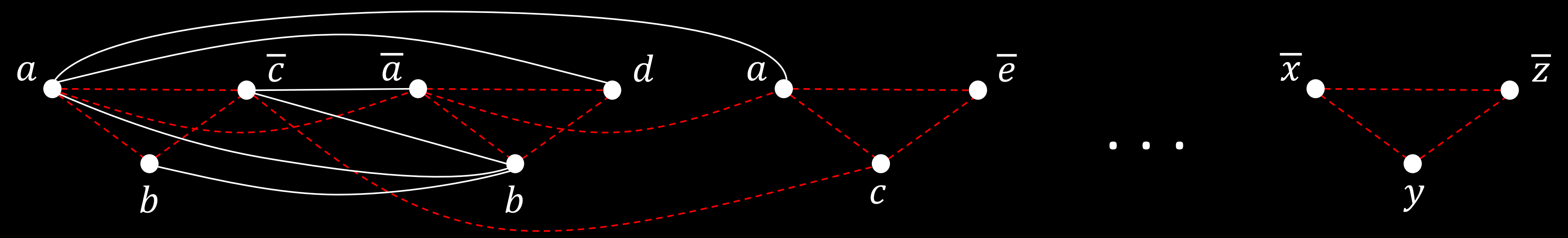
Proof: Give polynomial-time reduction f that maps ϕ to G, k where ϕ is satisfiable iff G has a k-clique.

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

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

$f \downarrow$

$\boxed{G}$ =



$\boxed{k}$ = # clauses

Forbidden edges:
1) within a clause
2) inconsistent labels (a and $\overline{a}$)

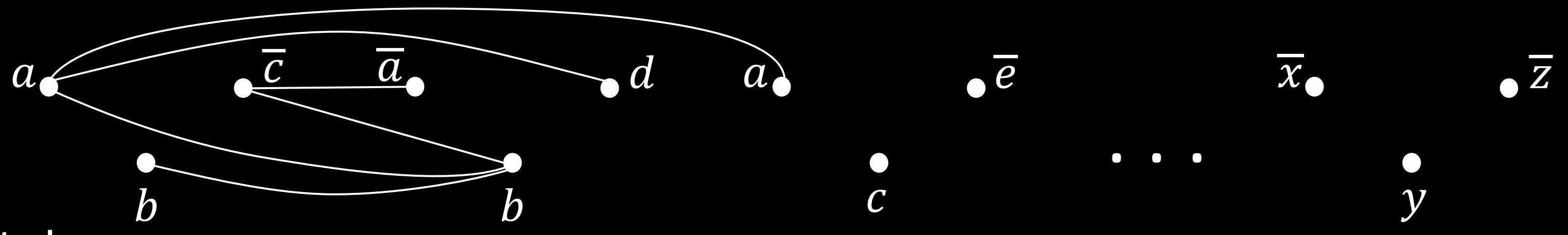
G has all non-forbidden edges

$3SAT \leq_P CLIQUE$ conclusion

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

$\downarrow f$

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



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

($\rightarrow$) 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.

($\leftarrow$) Take any k-clique in G. It must have 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.

Corollary: $CLIQUE \in \mathrm{P} \rightarrow 3SAT \in \mathrm{P}$

Check-in 15.1

Does this proof require 3 literals per clause?

(a) Yes, to prove the claim.

(b) Yes, to show it is in poly time.

(c) No, it works for any size clauses.

Check-in 15.1

NP-completeness

Defn: B is NP-complete if
1) $B \in \mathrm{NP}$
2) For all $A \in \mathrm{NP}$, $A \leq_{\mathrm{P}} B$

If B is NP-complete and $B \in \mathrm{P}$ then $\mathrm{P} = \mathrm{NP}$.

Cook-Levin Theorem: SAT is NP-complete
Proof: Next lecture; assume true

NP

next lecture $\leq_{\mathrm{P}} SAT$

today $\leq_{\mathrm{P}} HAMPATH$

To show some language C is NP-complete, show $3SAT \leq_P C$.

or some other previously shown NP-complete language

Check-in 15.2

What language that we've previously seen is most analogous to SAT?

(a) A_{TM}
(b) E_{TM}
(c) $\{0^k 1^k \mid k \geq 0\}$

Check-in 15.2

$HAMPATH$ is NP-complete

Theorem: $HAMPATH$ is NP-complete

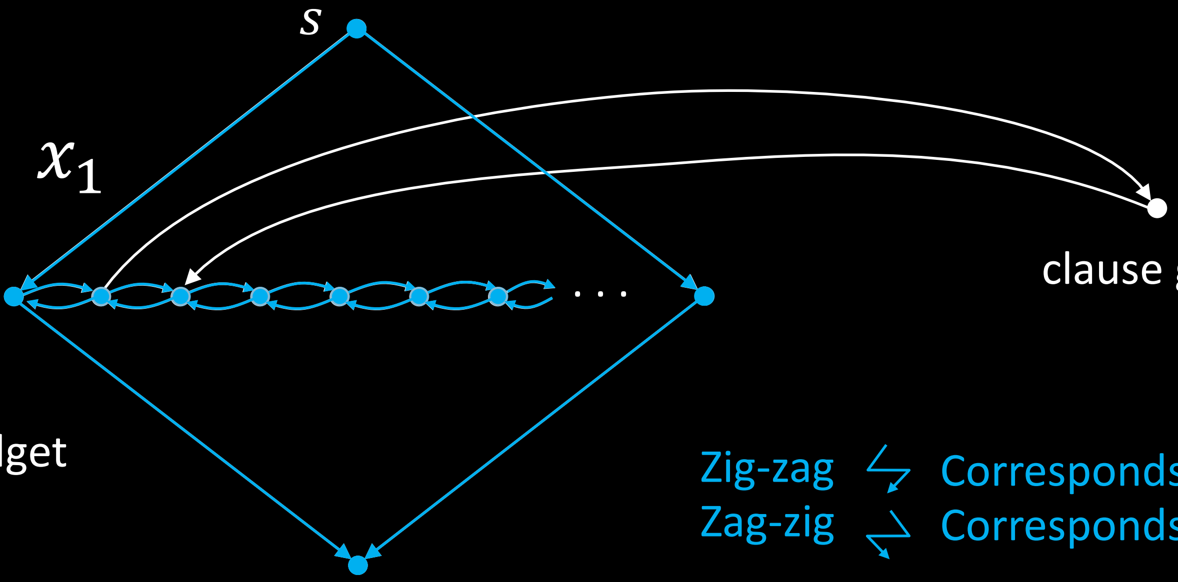
Proof: Show $3SAT \leq_P HAMPATH$ (assumes $3SAT$ is NP-complete)

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

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

$$\downarrow f$$

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



clause gadget

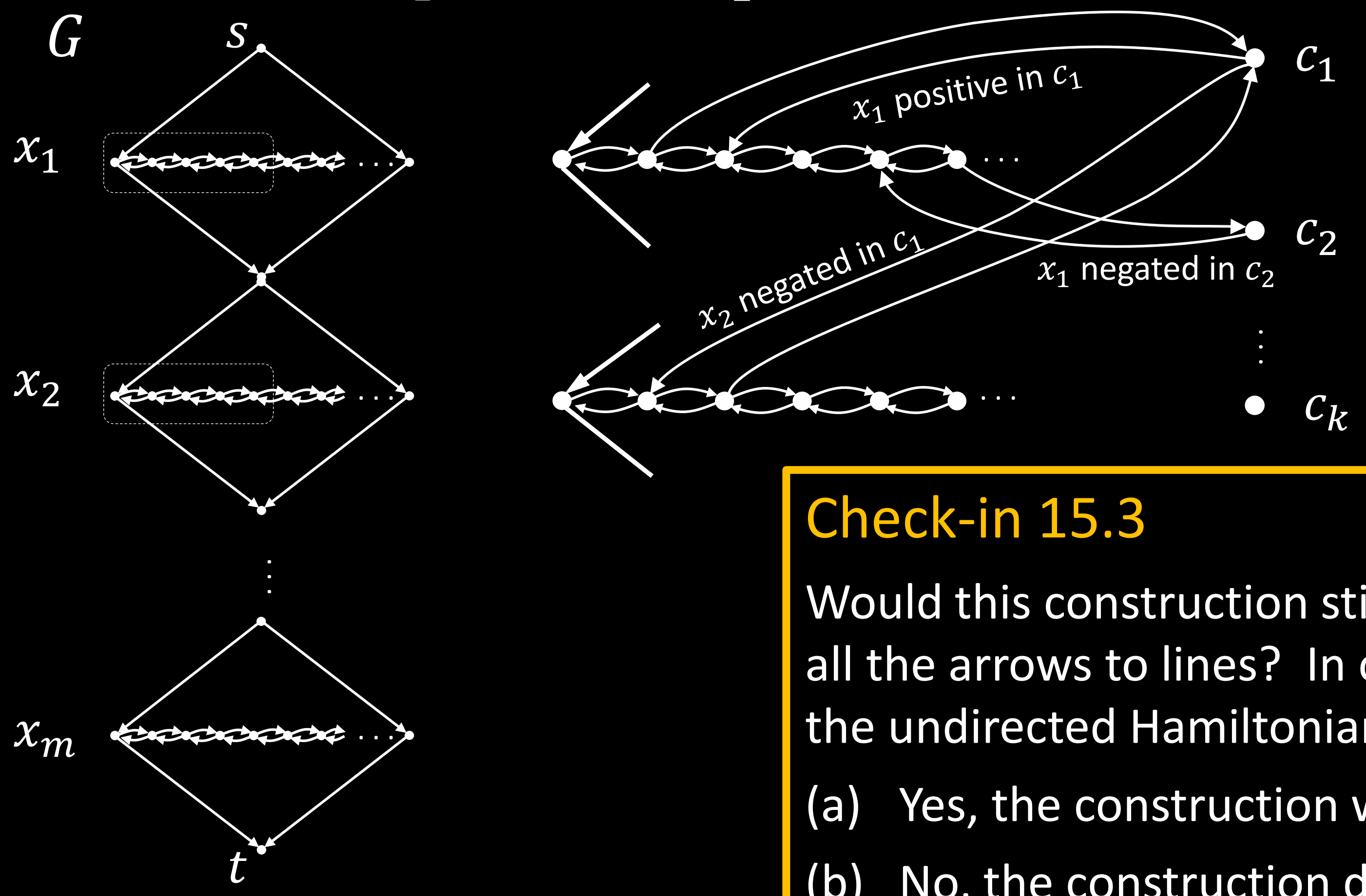
variable gadget

Zig-zag Corresponds to setting x_1 TRUE

Zag-zig Corresponds to setting x_1 FALSE

Construction of G

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



m variables
k clauses

The reduction f is computable in polynomial time.

Check-in 15.3

Would this construction still work if we made G undirected by changing all the arrows to lines? In other words, would this construction show that the undirected Hamiltonian path problem is NP-complete?

(a) Yes, the construction would still work.

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

Check-in 15.3

Cook-Levin Theorem (idea)

Theorem: SAT is NP-complete

Proof: 1) $SAT \in NP$ (done)

2) Show that for each $A \in NP$ we have $A \leq_P SAT$:

Let $A \in NP$ be decided by NTM M in time n^k.

Give a polynomial-time reduction f mapping A to SAT.

$f\colon \Sigma^* \to$ formulas

$f(w) = \langle \phi_{M,w} \rangle$

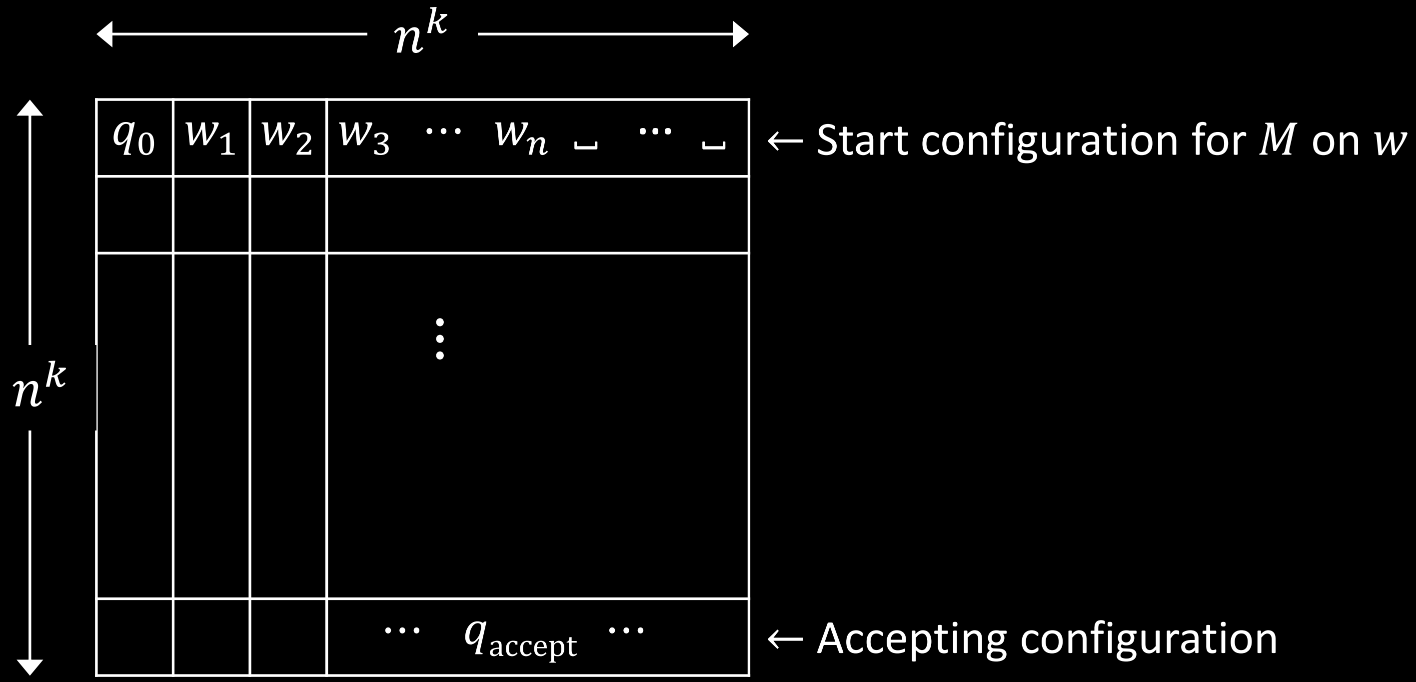
$w \in A$ iff $\phi_{M,w}$ is satisfiable

Idea: $\phi_{M,w}$ simulates M on w. Design $\phi_{M,w}$ to "say" M accepts w.

Satisfying assignment to $\phi_{M,w}$ is a computation history for M on w.

Tableau for M on w

Defn: An (accepting) tableau for NTM M on w is an $n^k \times n^k$ table representing an computation history for M on w on an accepting branch of the nondeterministic computation.

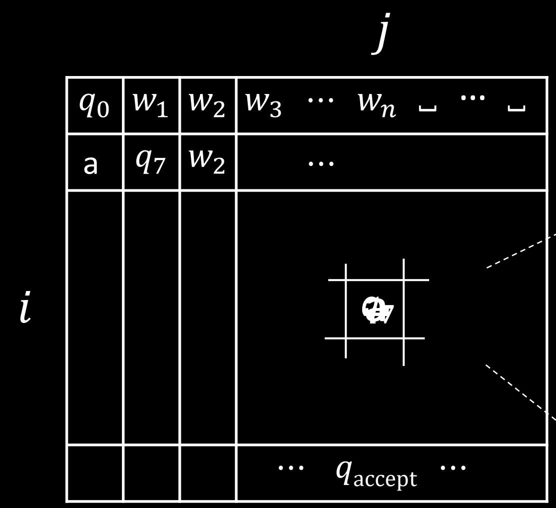


Construct $\phi_{M,w}$ to "say" M accepts w.

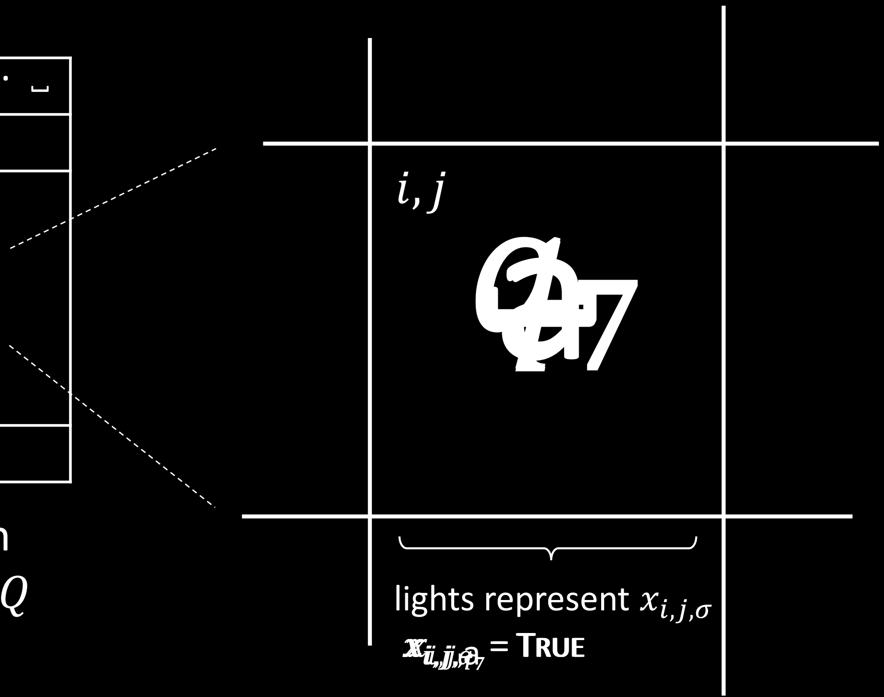
$\phi_{M,w}$ "says" a tableau for M on w exists.

$\phi_{M,w} = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$

Constructing $\phi_{M,w}$: ϕ_{cell}



Cell i, j can contain any symbol in $\Gamma \cup Q$



The variables of $\phi_{M,w}$ are $x_{i,j,\sigma}$ for $1 \leq i, j \leq n^k$ and $\sigma \in \Gamma \cup Q$.

$x_{i,j,\sigma}$ = TRUE means cell i, j contains σ.

Check-in 16.2

How many variables does $\phi_{M,w}$ have? Recall that $n = |w|$.

(a) $O(n)$

(b) $O(n^2)$

(c) $O(n^k)$

(d) $O(n^{2k})$

Check-in 16.2

Constructing $\phi_{M,w}$: ϕ_{start} and ϕ_{accept}

	1	2	3	⋯⋯ n^k	
1	q_0	w_1	w_2	w_3 ⋯ w_n ␣ ⋯ ␣	← Start configuration
	a	q_7	w_2	⋯	
n^k				⋯ q_{accept} ⋯	← Accepting configuration

$\phi_{M,w}$ "says" a tableau for M on w exists.

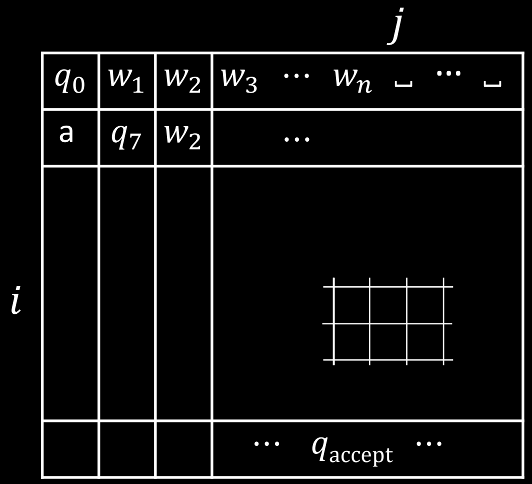
$\phi_{M,w} = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$

ϕ_{cell} done ✓

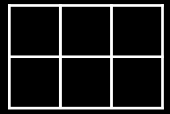
$\phi_{\text{start}} =$

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

Constructing $\phi_{M,w}$: ϕ_{move}



2×3 neighborhood



$\phi_{M,w}$ "says" a tableau for M on w exists.

$\phi_{M,w} = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$

✓ ✓ ✓

Legal neighborhoods: consistent with M's transition function

potential examples:

a	q_7	b
q_3	a	c

a	b	c
a	b	c

a	b	c
a	b	q_5

a	b	c
d	b	c

Illegal neighborhoods: not consistent with M's transition function

examples:

a	b	c
a	d	c

a	b	c
a	q_2	c

a	q_7	c
a	b	c

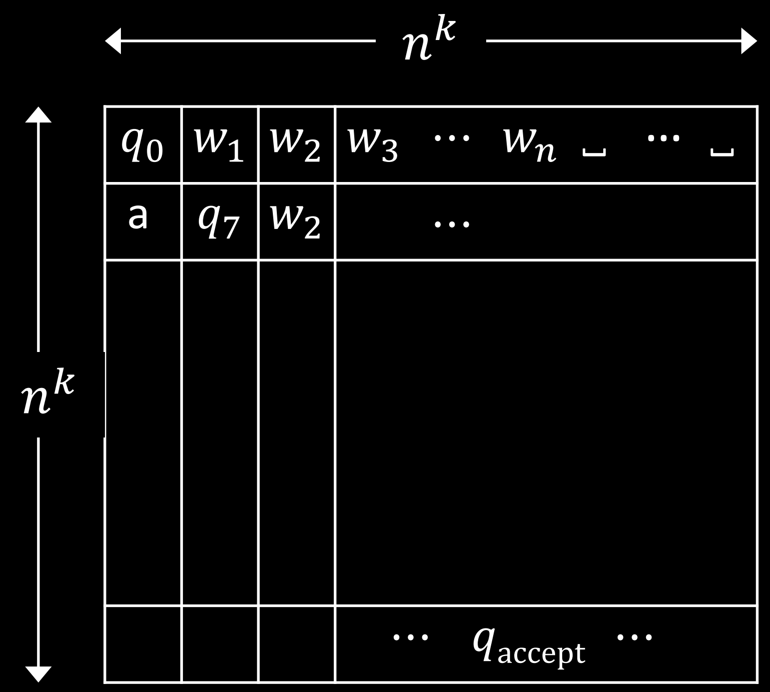
a	q_7	c
q_3	d	q_4

Claim: If every 2×3 neighborhood is legal then tableau corresponds to a computation history.

$$\phi_{\text{move}} = \bigwedge_{1<i,j<n^k} \left(\bigvee_{\text{Legal } \begin{smallmatrix} r & s & t \\ v & y & z \end{smallmatrix}} \left(x_{i,j-1,r} \wedge x_{i,j,s} \wedge x_{i,j+1,t} \wedge x_{i+1,j-1,v} \wedge x_{i+1,j,y} \wedge x_{i+1,j+1,z} \right) \right)$$

Says that the neighborhood at i, j is legal

Conclusion: SAT is NP-complete



Summary:
For $A \in$ NP, decided by NTM M,
we gave a reduction f from A to SAT:

$f \colon \Sigma^* \rightarrow$ formulas

$f(w) = \langle \phi_{M,w} \rangle$

$w \in A$ iff $\phi_{M,w}$ is satisfiable.

$$\phi_{M,w} = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

The size of $\phi_{M,w}$ is roughly the size of the tableau for M on w, so size is $O\left(n^k \times n^k\right) = O\left(n^{2k}\right)$.

Therefore f is computable in polynomial time.

$3SAT$ is NP-complete

a	b	a ∨ b = c	
1	1	1	$(a \wedge b) \to c$
0	1	1	$(\overline{a} \wedge b) \to c$
1	0	1	$(a \wedge \overline{b}) \to c$
0	0	0	$(\overline{a} \wedge \overline{b}) \to \overline{c}$

Theorem: $3SAT$ is NP-complete

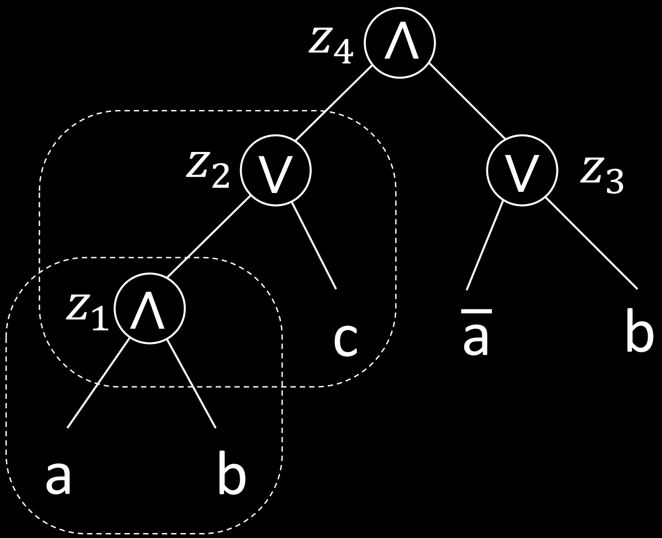
Proof: Show $SAT \leq_P 3SAT$

Give reduction f converting formula ϕ to 3CNF formula ϕ', <u>preserving satisfiability</u>.

(Note: ϕ and ϕ' are not logically equivalent)

Example: Say $\phi = \big((a \wedge b) \vee c\big) \wedge (\overline{a} \vee b)$

Tree structure for ϕ:



Logical equivalence: $(A \to B)$ and $\big(\overline{A} \vee B\big)$ $\quad \overline{(A \wedge B)}$ and $\big(\overline{A} \vee \overline{B}\big)$

$$\phi' = \big((a \wedge b) \to z_1\big) \wedge \big((\overline{a} \wedge b) \to \overline{z_1}\big) \wedge \Big(\big(a \wedge \overline{b}\big) \to \overline{z_1}\Big) \wedge \Big(\big(\overline{a} \wedge \overline{b}\big) \to \overline{z_1}\Big)$$

$$\wedge \big((z_1 \wedge c) \to z_2\big) \wedge \big((\overline{z_1} \wedge c) \to z_2\big) \wedge \big((z_1 \wedge \overline{c}) \to z_2\big) \wedge \big((\overline{z_1} \wedge \overline{c}) \to \overline{z_2}\big)$$

⋮ repeat for each z_i

$\wedge \; (z_4)$

Check-in 16.3

If ϕ has k operations (∧ and ∨), how many clauses has ϕ'?

(a) $k+1$

(b) $4k+1$

(c) k^2

(d) $2k^2$

Check-in 16.3

SPACE Complexity

Defn: Let $f\colon \mathbb{N} \to \mathbb{N}$ where $f(n) \geq n$. Say TM M runs in space $f(n)$ if M always halts and uses at most $f(n)$ tape cells on all inputs of length n.

Check-in 17.1

We define space complexity for multi-tape TMs by taking the sum of the cells used on all tapes.

Do we get the same class PSPACE for multi-tape TMs?

(a) No.

(b) Yes, converting a multi-tape TM to single-tape only squares the amount of space used.

(c) Yes, converting a multi-tape TM to single-tape only increases the amount of space used by a constant factor.

Check-in 17.1

Relationships between Time and SPACE Complexity

Theorem: For $t(n) \geq n$

1) $\text{TIME}\big(t(n)\big) \subseteq \text{SPACE}\big(t(n)\big)$
2) $\text{SPACE}\big(t(n)\big) \subseteq \text{TIME}\Big(2^{O(t(n))}\Big)$

$$= \bigcup_c \text{TIME}\big(c^{t(n)}\big)$$

Proof:

1) A TM that runs in $t(n)$ steps cannot use more than $t(n)$ tape cells.
2) A TM that uses $t(n)$ tape cells cannot use more than $c^{t(n)}$ time without repeating a configuration and looping (for some c).

Corollary: P ⊆ PSPACE

Theorem: NP ⊆ PSPACE [next slide]

NP ⊆ PSPACE

Theorem: NP ⊆ PSPACE

Proof:

1. $SAT \in$ PSPACE
2. If $A \leq_{\mathrm{P}} B$ and $B \in$ PSPACE then $A \in$ PSPACE

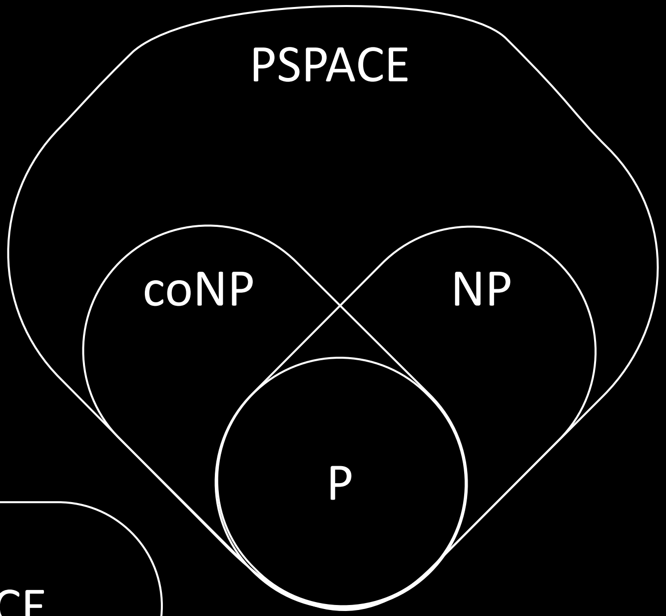
Defn: coNP $= \left\{ \overline{A} \,\middle|\, A \in \text{NP} \right\}$

$\overline{HAMPATH} \in$ coNP

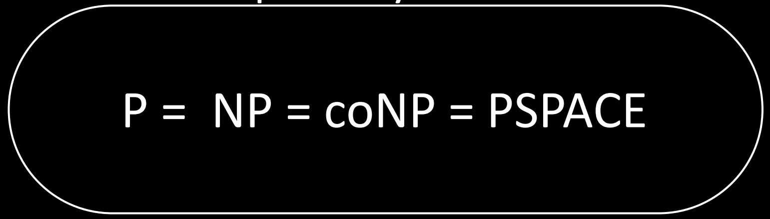
$TAUTOLOGY = \{\langle \phi \rangle |$ all assignments satisfy $\phi\} \in$ coNP

coNP ⊆ PSPACE (because PSPACE = coPSPACE)

P = PSPACE ? *Not known.*



Or possibly:



Example: $TQBF$

Defn: A quantified Boolean formula (QBF) is a Boolean formula with leading exists ($\exists x$) and for all ($\forall x$) quantifiers. All variables must lie within the scope of a quantifier.

A QBF is TRUE or FALSE.

Examples: $\phi_1 = \forall x\ \exists y\ [(x \vee y) \wedge (\overline{x} \vee \overline{y})]$

$\phi_2 = \exists y\ \forall x\ [(x \vee y) \wedge (\overline{x} \vee \overline{y})]$

Defn: $TQBF = \{\langle\phi\rangle \mid \phi$ is a QBF that is TRUE$\}$

Thus $\phi_1 \in TQBF$ and $\phi_2 \notin TQBF$.

Theorem: $TQBF \in$ PSPACE

Check-in 17.2

How is SAT a special case of $TQBF$?

(a) Remove all quantifiers.

(b) Add $\exists$ and $\forall$ quantifiers.

(c) Add only $\exists$ quantifiers.

(d) Add only $\forall$ quantifiers.

$TQBF \in$ PSPACE

Theorem: $TQBF \in$ PSPACE

Proof: "On input $\langle\phi\rangle$

1. If ϕ has no quantifiers, then ϕ has no variables
   so either $\phi =$ True or $\phi =$ False. Output accordingly.
2. If $\phi = \exists x\ \psi$ then evaluate ψ with $x =$ TRUE and $x =$ FALSE recursively.
   Accept if either accepts. *Reject* if not.
3. If $\phi = \forall x\ \psi$ then evaluate ψ with $x =$ TRUE and $x =$ FALSE recursively.
   Accept if both accept. *Reject* if not."

Space analysis:

Each recursive level uses constant space (to record the x value).
The recursion depth is the number of quantifiers, at most $n = |\langle\phi\rangle|$.

So $TQBF \in$ SPACE(n)

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A.

Defn: $LADDER_{\mathrm{DFA}} = \{\langle B, u, v \rangle |\ B$ is a DFA and $L(B)$ contains a ladder $y_1, y_2, \ldots, y_k$ where $y_1 = u$ and $y_k = v\}$.

Theorem: $LADDER_{\mathrm{DFA}} \in$ NPSPACE

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER_{\text{DFA}} \in$ NPSPACE

Theorem: $LADDER_{\text{DFA}} \in$ NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v.

Careful- (a) cannot store sequence, (b) must terminate.

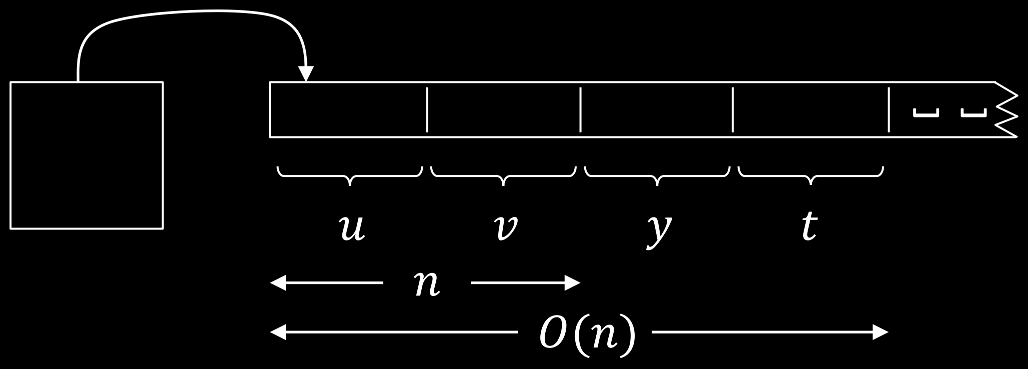
Proof: "On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y.
4. *Reject* if $y \notin L(B)$.
5. *Accept* if $y = v$.
6. *Reject* [exceeded t steps].

Space used is for storing y and t.

$LADDER_{\text{DFA}} \in \text{NSPACE}(n)$.

Theorem: $LADDER_{\text{DFA}} \in$ PSPACE (!)



$\leq t$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER_{\text{DFA}} \in$ PSPACE

Theorem: $LADDER_{\text{DFA}} \in$ SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.
Here's a recursive procedure to solve the bounded DFA ladder problem:

$BOUNDED\text{-}LADDER_{\text{DFA}} = \{\langle B, u, v, b\rangle \mid B$ a DFA and $u \xrightarrow{b} v$ by a ladder in $L(B)\}$

$B\text{-}L$ = "On input $\langle B, u, v, b\rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$
3. Recursively test $u \xrightarrow{b/2} w$ and $w \xrightarrow{b/2} v$ [division rounds up]
4. *Accept* both accept.
5. *Reject* [if all fail]."

Test $\langle B, u, v\rangle \in LADDER_{\text{DFA}}$ with $B\text{-}L$ procedure on input $\langle B, u, v, t\rangle$ for $t = |\Sigma|^m$

Space analysis:

Each recursive level uses space $O(n)$ (to record w).

Recursion depth is $\log t = O(m) = O(n)$.

Total space used is $O(n^2)$.

Check-in 17.3

Find an English word ladder connecting MUST and VOTE.

(a) Already did it.

(b) I will.

Check-in 17.3

PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

Proof: Convert NTM N to equivalent TM M, only squaring the space used.

For configurations c_i and c_j of N, write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

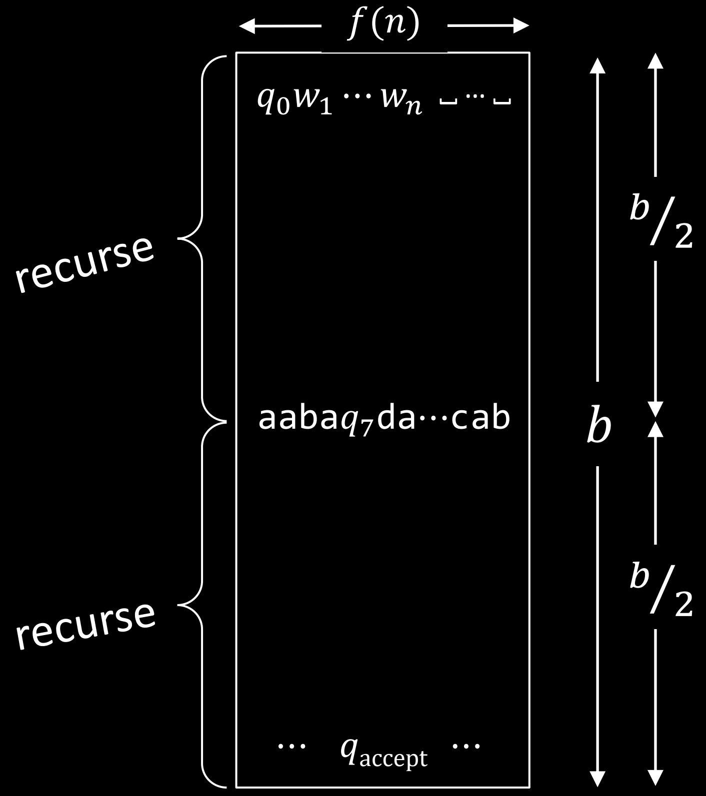
M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

1. If $b = 1$, check directly by using N's program and answer accordingly.
2. If $b > 1$, repeat for all configurations c_{mid} that use $f(n)$ space.
3. Recursively test $c_i \xrightarrow{b/2} c_{\text{mid}}$ and $c_{\text{mid}} \xrightarrow{b/2} c_j$
4. If both are true, *accept*. If not, continue.
5. *Reject* if haven't yet accepted."

Test if N accepts w by testing $c_{\text{start}} \xrightarrow{t} c_{\text{accept}}$ where t = number of configurations = $|Q| \times f(n) \times d^{f(n)}$

Each recursion level stores 1 config = $O\big(f(n)\big)$ space.

Number of levels = $\log t = O\big(f(n)\big)$. Total $O\big(f^2(n)\big)$ space.



PSPACE-completeness

Defn: B is <u>PSPACE-complete</u> if
1) $B \in$ PSPACE
2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} B$

If B is PSPACE-complete and $B \in$ P then P $=$ PSPACE.

Check-in 18.1

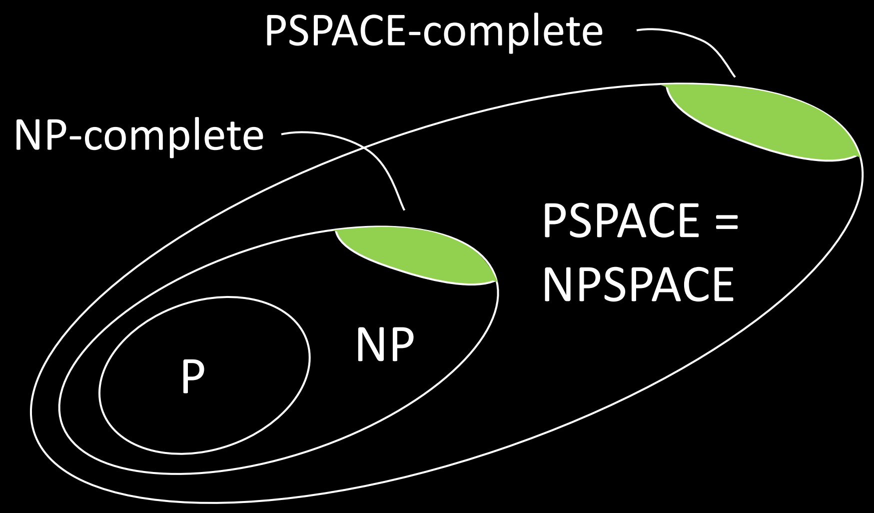
Knowing that $TQBF$ is PSPACE-complete, what can we conclude if $TQBF \in$ NP? Check all that apply.

(a) P = PSPACE

(b) NP = PSPACE

(c) P = NP

(d) NP = coNP



Think of complete problems as the “hardest” in their associated class.

$TQBF$ is PSPACE-complete

Recall: $TQBF = \{\langle\phi\rangle \mid \phi$ is a QBF that is TRUE$\}$

Examples: $\phi_1 = \forall x\ \exists y\ [(x \vee y) \wedge (\overline{x} \vee \overline{y})] \in TQBF$ [TRUE]
$\phi_2 = \exists y\ \forall x\ [(x \vee y) \wedge (\overline{x} \vee \overline{y})] \notin TQBF$ [FALSE]

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in$ PSPACE ✓

2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} TQBF$

Let $A \in$ PSPACE be decided by TM M in space n^k.

Give a polynomial-time reduction f mapping A to $TQBF$.

$f\colon \Sigma^* \to$ QBFs

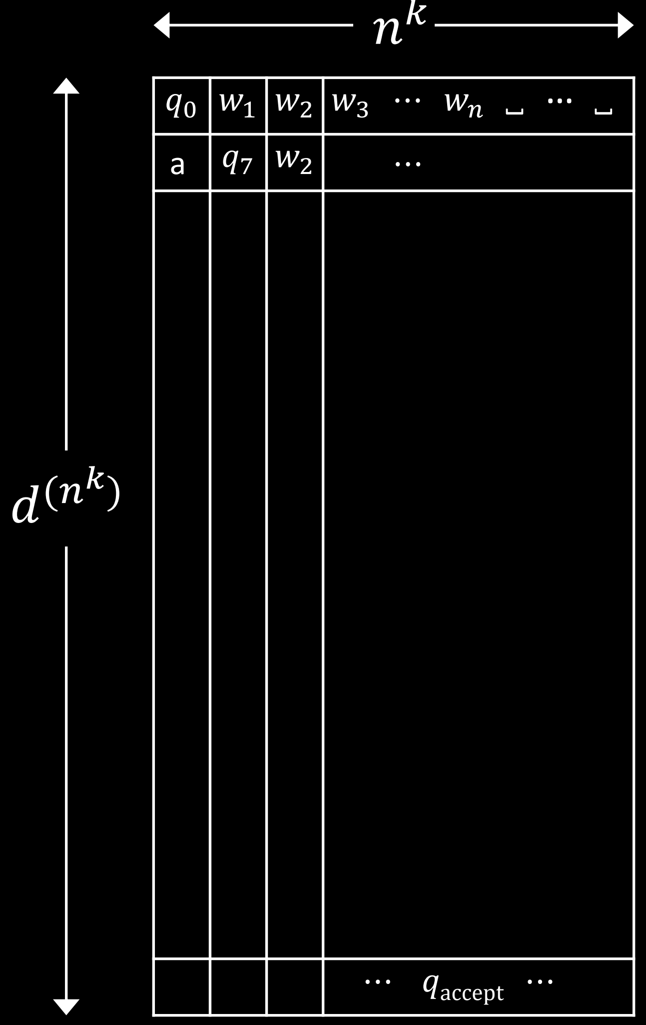
$f(w) = \langle\phi_{M,w}\rangle$

$w \in A$ iff $\phi_{M,w}$ is TRUE

Plan: Design $\phi_{M,w}$ to "say" M accepts w. $\phi_{M,w}$ simulates M on w.

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w



Recall: A tableau for M on w represents a computation history for M on w when M accepts w.
Rows of that tableau are configurations.

M runs in space n^k, its tableau has:

- n^k columns (max size of a configuration)
- $d^{(n^k)}$ rows (max number of steps)

Constructing $\phi_{M,w}$. Try Cook-Levin method.
Then $\phi_{M,w}$ will be as big as tableau.
But that is exponential: $n^k \times d^{(n^k)}$.
Too big! ☹

Constructing $\phi_{M,w}$: 2nd try

hide →

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which "says" $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\boxed{\substack{\exists x_1, x_2, \cdots, c_l \\ \text{as in Cook-Levin}}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \ \wedge \ \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

$\exists c_{\text{mid}}[\phi_{\ ,\ ,\ b/4} \ \wedge \ \phi_{\ ,\ ,\ b/4}]$ $\qquad$ $\exists c_{\text{mid}}[\phi_{\ ,\ ,\ b/4} \ \wedge \ \phi_{\ ,\ ,\ b/4}]$

⋮ $\qquad$ ⋮ $\exists c_{\text{mid}}[\phi_{\ ,\ ,\ b/8} \cdots]$

$\phi_{\ ,\ ,\ 1}$ defined as in Cook-Levin

Check-in 18.2

Why shouldn't we be surprised that this construction fails?

(a) We can't define a QBF by using recursion.

(b) It doesn't use ∀ anywhere.

(c) We know that $TQBF \notin$ P.

$$\phi_{M,w} = \phi_{c_{\text{start}}, c_{\text{accept}}, t}$$
$$t = d^{(n^k)}$$

Size analysis:
Each recursive level doubles number of QBFs.
Number of levels is $\log d^{(n^k)} = O(n^k)$.
→ Size is exponential. ☹

Check-in 18.2

Constructing $\phi_{M,w}$: 3rd try

$$\phi_{c_i,\,c_j,\,b} = \exists c_{\text{mid}} \Big[\underbrace{\phi_{c_i,\,c_{\text{mid}},\,b/2} \;\wedge\; \phi_{c_{\text{mid}},\,c_j,\,b/2}}_{\forall (c_g,\,c_h) \in \{(c_i,\,c_{\text{mid}}),\,(c_{\text{mid}},\,c_j)\}\,[\phi_{c_g,\,c_h,\,b/2}]} \Big]$$

$\forall (x \in S)\,[\,\psi\,]$
is equivalent to
$\forall x\,[(x \in S) \longrightarrow \psi]$

$\vdots$

$\phi_{\,,\,,1}$ defined as in Cook-Levin

$$\phi_{M,w} = \phi_{c_{\text{start}},\,c_{\text{accept}},\,t}$$
$$t = d^{(n^k)}$$

Size analysis:
Each recursive level adds $O(n^k)$ to the QBF.
Number of levels is $\log d^{(n^k)} = O(n^k)$.
→ Size is $O(n^k \times n^k) = O(n^{2k})$ ☺

Check-in 18.3

Would this construction still work if M were nondeterministic?

(a) Yes.

(b) No.