Space … and Beyond

Wednesday, May 11, 2022

FINAL REMAINING "FRONTIERS," ACCORDING TO POPULAR USAGE

Announcements

- HW 12 due tonight 11:59pm EST
  - Last HW!
- Last lecture!

Previously: **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.

These are the "hardest" problems (in **NP**) to solve

NP-Completeness vs **NP**-Hardness

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.

"**NP**-Hard"

"**NP**-Complete" = in **NP** + "**NP**-Hard"

So a language can be **NP**-hard but not **NP**-complete!

Flashback: The Halting Problem

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

Thm: $HALT_{\mathsf{TM}}$ is undecidable

Proof, by contradiction:

- Assume $HALT_{\mathsf{TM}}$ has *decider* R; use it to create decider for A_{TM}:

- …

- But A_{TM} is undecidable and has no decider!

Flashback: The Halting Problem

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle |\ M \text{ is a TM and } M \text{ halts on input } w\}$

Thm: $HALT_{\mathsf{TM}}$ is undecidable

Proof, by contradiction:

- Assume $HALT_{\mathsf{TM}}$ has *decider* R; use it to create decider for A_{TM}:

$S =$ "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*. ← This means M loops on input w
3. If R accepts, simulate M on w until it halts. ← This step always halts
4. If M has accepted, *accept*; if M has rejected, *reject*."

Flashback: The Halting Problem

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

$HALT_{\mathsf{TM}}$ is **undecidable** …
so it's **definitely undecidable** in a limited amount of steps, i.e., it's not in **P** or **NP**

Thm: $HALT_{\mathsf{TM}}$ is undecidable

Proof, by contradiction:

- Assume $HALT_{\mathsf{TM}}$ has *decider* R; use it to create decider for A_{TM}:

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:
1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

- But A_{TM} is undecidable!
  - I.e., this decider that we just created cannot exist! So $HALT_{\mathsf{TM}}$ is undecidable

The Halting Problem is **NP**-Hard

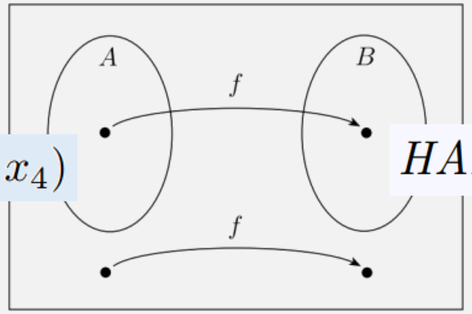
$$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Proof: **Reduce *3SAT* to the Halting Problem**

(Why does this prove that the Halting Problem is **NP**-hard?)

Because *3SAT* is **NP**-complete! (so every **NP** problem is poly time reducible to *3SAT*)

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



$$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

The Halting Problem is **NP**-Hard

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

<u>Computable function</u>, from $3SAT \rightarrow HALT_{\mathsf{TM}}$:

On input ϕ, a formula in 3cnf:

- Construct TM M

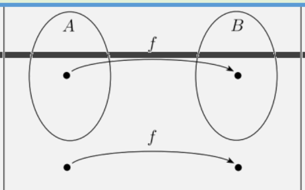
  M = on input ϕ
  - Try all assignments
    - If any satisfy ϕ, then **accept**
  - When all assignments have been tried, start over

  This loops when there is no satisfying assignment!

- Output $<M, \phi>$

$\Rightarrow$ If ϕ has a satisfying assignment, then M halts on ϕ
$\Leftarrow$ If ϕ has no satisfying assignment, then M loops on ϕ

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



$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

Review:

DEFINITION

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

1. B is in NP, and

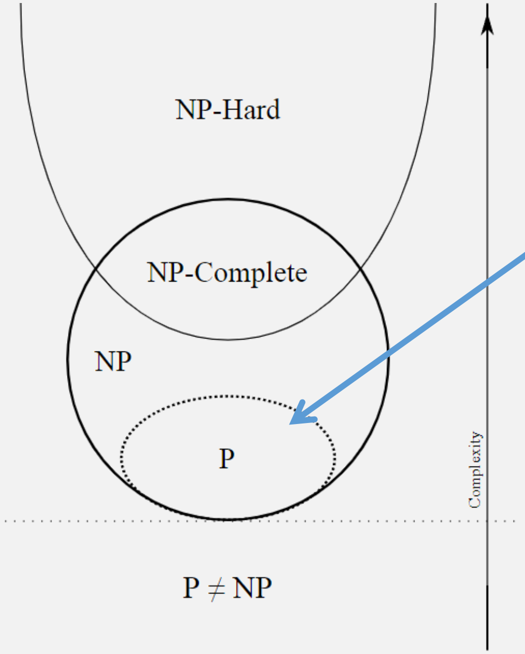
2. every A in NP is polynomial time reducible to B.

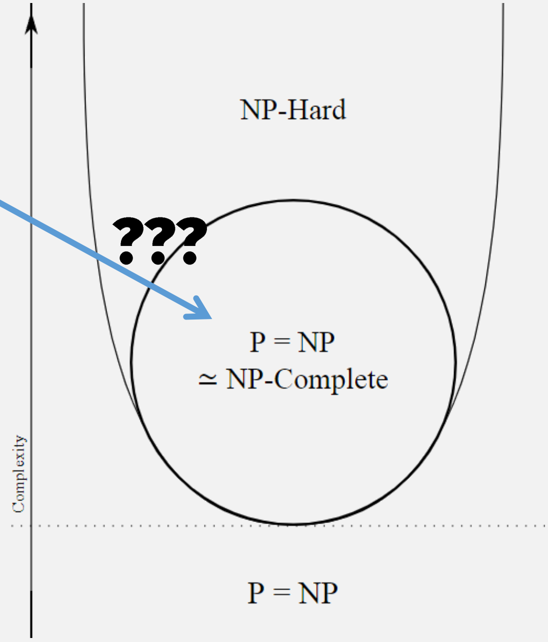
So **a language can satisfy condition #2** but **not condition #1**

But **can a language satisfy condition #1** but **not condition #2?**

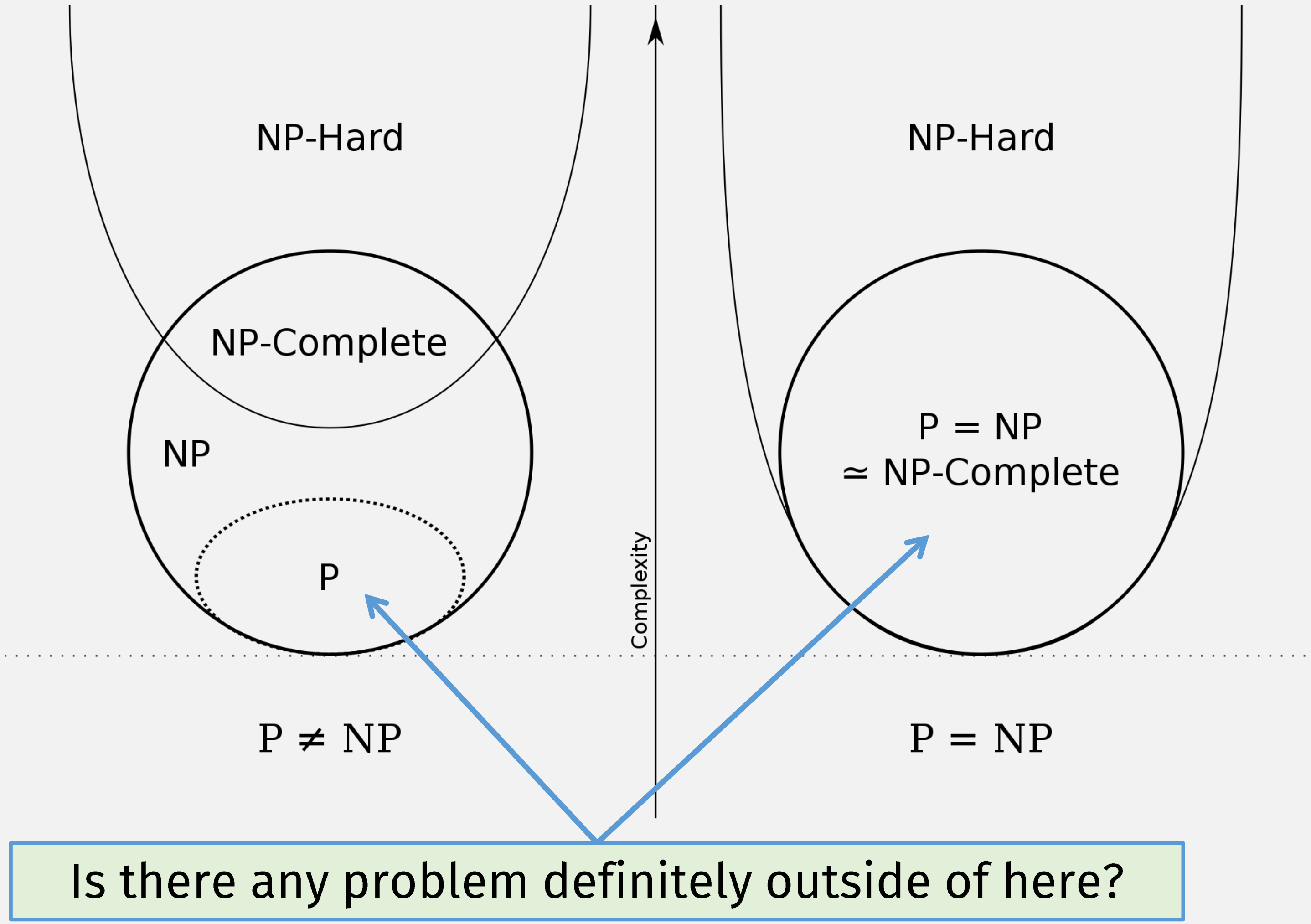
Yes, every language in P ...

... unless P = NP





NP-Completeness vs **NP**-Hardness



Space …

Flashback: Dynamic Programming Example

- Chomsky Grammar G:
  - S → AB | BC
  - A → BA | a
  - B → CC | b
  - C → AB | a
- Example string: **baaba**
- Store every partial string and their generating variables in a table

We are gaining time …

… by spending more space!

Substring end char

Substring start char

	b	**a**	**a**	**b**	**a**
b	vars for “b”	vars for “ba”	vars for “baa”	…	
a		vars for “a”	vars for “aa”	vars for “aab”	
a			…		
b					
a					

Space Complexity, Formally

TMs have a **space complexity**

DEFINITION

Let M be a deterministic Turing machine that halts on all inputs. The ***space complexity*** of M is the function $f : \mathcal{N} \longrightarrow \mathcal{N}$, where $f(n)$ is the maximum number of tape cells that M scans on any input of length n. If the space complexity of M is $f(n)$, we also say that M runs in space $f(n)$.

If M is a nondeterministic Turing machine wherein all branches halt on all inputs, we define its space complexity $f(n)$ to be the maximum number of tape cells that M scans on any branch of its computation for any input of length n.

decider

Space Complexity Classes

TMs have a **space complexity**

Languages are in a **space complexity class**

DEFINITION

Let $f\colon \mathcal{N} \longrightarrow \mathcal{R}^+$ be a function. The ***space complexity classes***, **SPACE**($f(n)$) and **NSPACE**($f(n)$), are defined as follows.

$$\text{SPACE}(f(n)) = \{L \mid L \text{ is a } \boxed{\text{language}} \text{ decided by an } O(f(n)) \text{ space deterministic Turing machine}\}.$$

$$\text{NSPACE}(f(n)) = \{L \mid L \text{ is a } \boxed{\text{language}} \text{ decided by an } O(f(n)) \text{ space nondeterministic Turing machine}\}.$$

Compare:

Let $t\colon \mathcal{N} \longrightarrow \mathcal{R}^+$ be a function. Define the ***time complexity class***, **TIME**($t(n)$), to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

$$\textbf{NTIME}(t(n)) = \{L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine}\}.$$

Example: *SAT* Space Usage

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

$2^{O(m)}$ exponential time machine

$M_1 =$ "On input $\langle\phi\rangle$, where ϕ is a Boolean formula:

1. For each truth assignment to the variables $x_1, \ldots, x_m$ of ϕ:
2. Evaluate ϕ on that truth assignment.
3. If ϕ ever evaluated to 1, *accept*; if not, *reject*."

Each loop iteration requires $O(m)$ space

But the space is re-used on each loop!
(nothing is stored from the last loop)

So the entire machine only needs $O(m)$ space!

SAT is in $O(m)$ space complexity class!

Space is **"more powerful"** than time.

Example: Nondeterministic Space Usage

$$ALL_{\mathsf{NFA}} = \{\langle A \rangle |\ A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Nondeterministic decider for $\overline{ALL_{\mathsf{NFA}}}$

(accepts NFAs that reject something)

N = "On input $\langle M \rangle$, where M is an NFA:

1. Place a marker on the start state of the NFA.
2. Repeat 2^q times, where q is the number of states of M:
   Nondeterministically select an input symbol and change the positions of the markers on M's states to simulate reading that symbol.
4. *Accept* if stages 2 and 3 reveal some string that M rejects; that is, if at some point none of the markers lie on accept states of M. Otherwise, *reject*."

Machine tracks "current" state(s) of NFA

q states = 2^q possible combinations (so exponential time)

But each loop uses only $O(\boldsymbol{q})$ space!

Additionally, need a **counter** to count to 2^q: this requires $\log(2^q) = \boldsymbol{q}$ extra space

So the whole machine runs in (nondeterministic) linear $O(\boldsymbol{q})$ space!

Facts About Time vs Space (for Deciders)

TIME → SPACE

- If a decider runs in time $\boxed{t(\boldsymbol{n})}$, then its maximum space usage is …
- … $\boxed{t(\boldsymbol{n})}$
- … because it can add at most 1 tape cell per step

What about deterministic vs non-deterministic?

SPACE → TIME

- If a decider runs in space $\boxed{f(\boldsymbol{n})}$, then its maximum time usage is …
- … $(|\Gamma| + |Q|)^{f(\boldsymbol{n})} = \boxed{2^{df(\boldsymbol{n})}}$
- … because that's the number of possible configurations
- (and a decider cannot repeat a configuration)

Flashback: Deterministic vs Non-Det. Time

- If a non-deterministic TM runs in: $t(n)$ time
- Then an equivalent deterministic TM runs in: $2^{O(t(n))}$
  - **Exponentially** slower

What about space?

Deterministic vs Non-Det. Space

THEOREM

Savitch's theorem For any function $f: \mathcal{N} \longrightarrow \mathcal{R}^+$, where $f(n) \geq n$,

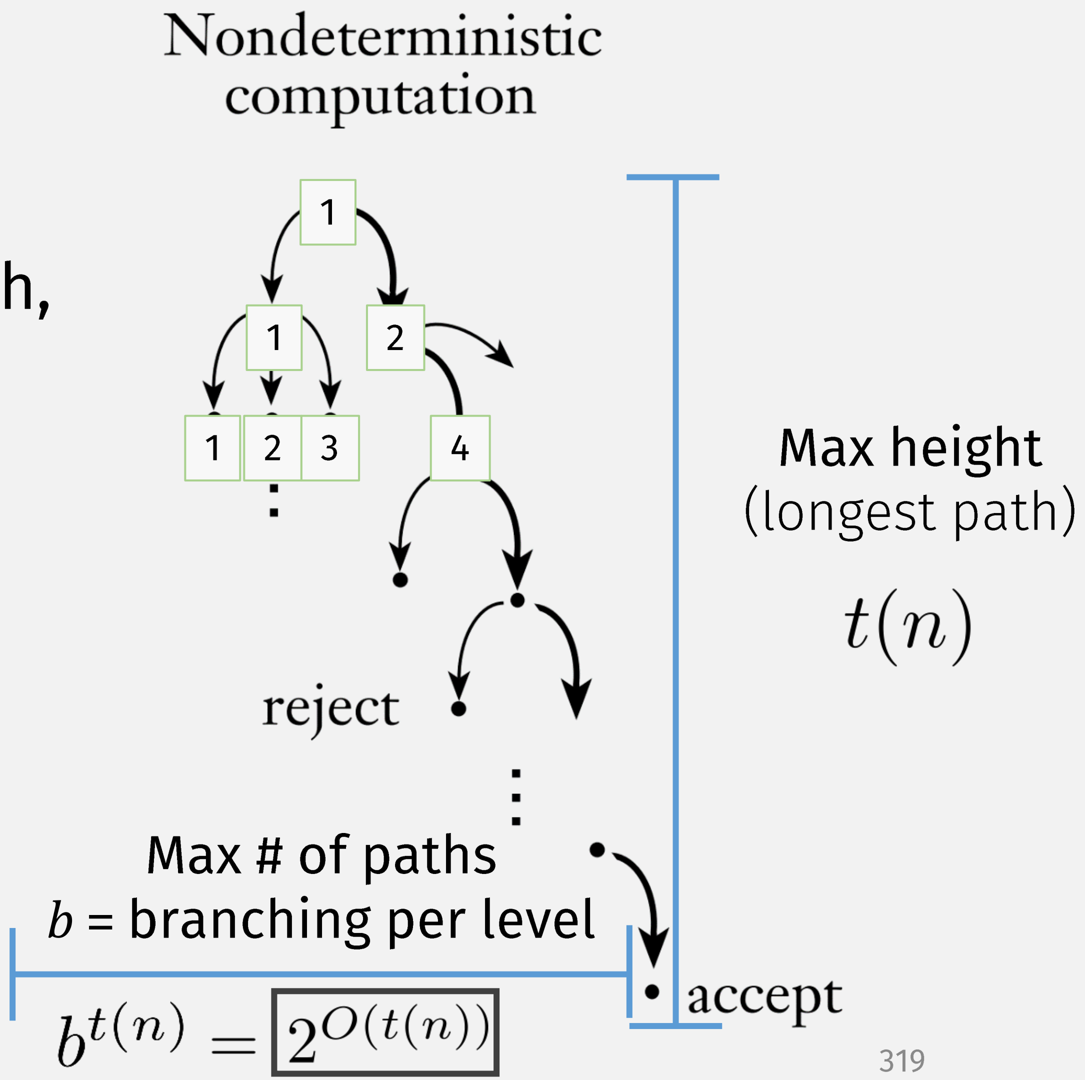
$$\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n)).$$

- If a non-deterministic TM runs in: $f(n)$ space
- Then an equivalent deterministic TM runs in: $f^2(n)$ space
  - ~~**Exponentially**~~ Only **Quadratically** slower!

Flashback: Nondet. TM → Deterministic TM

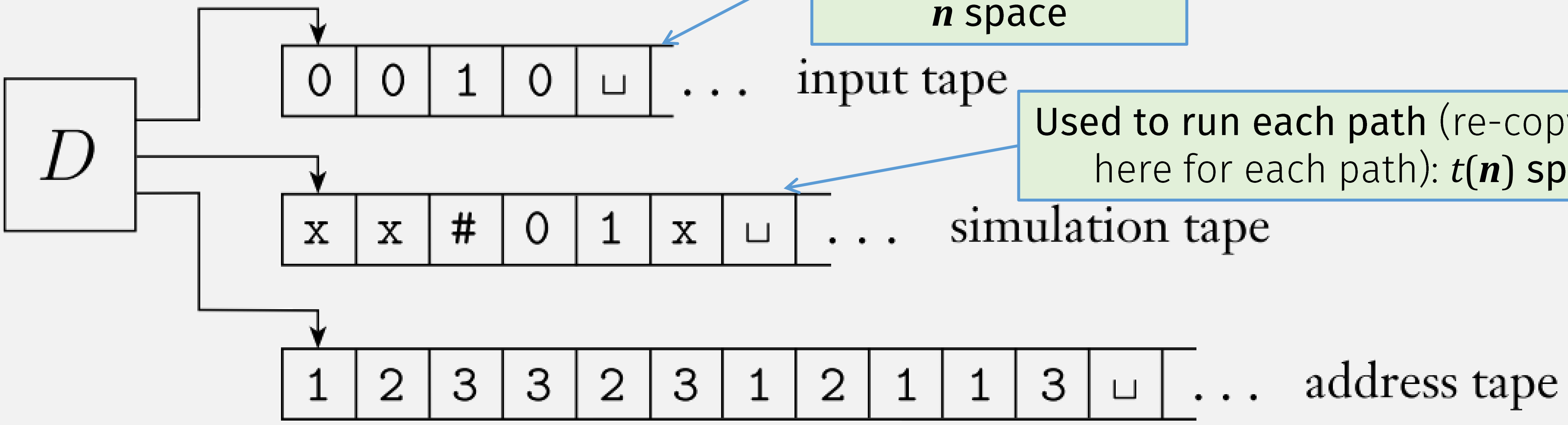
$t(n)$ time → $\boxed{2^{O(t(n))}}$ time

- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - 1-2
    - **1-1-1**



Flashback: Nondet → Deterministic TM: Space

3 tapes



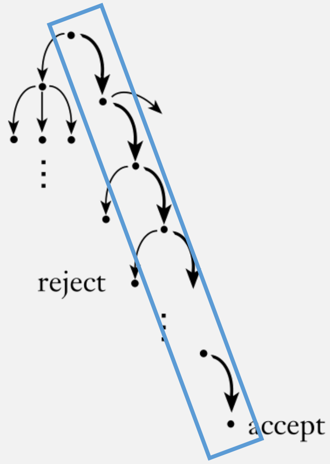
Always has input, never changes: $\boldsymbol{n}$ space

Used to run each path (re-copy input here for each path): $t(\boldsymbol{n})$ **space**

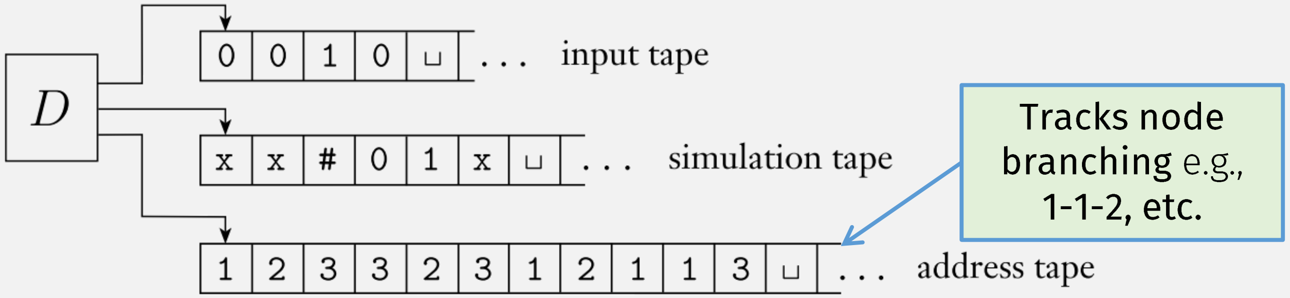
Tracks which node we are on, $2^{O(t(\boldsymbol{n}))}$ (exponential) space??

Nondet → Deterministic TM: Space

Let N be an NTM deciding language A in space $f(n)$



- This means a single path could use $f(n)$ space
- That path could take $2^{df(n)}$ steps
  - (That's the possible ways to fill the space)
  - Each step could be a non-deterministic branch that must be saved
- So naïvely tracking these branches requires $2^{df(n)}$ space!



- Instead, let's "divide and conquer" to reduce space!

"Divide and Conquer" TM Config Sequences

- Want to check whether:

TM configs → $c_{start} \xrightarrow{2^{O(f(n))} \text{ (possibly branching) steps}} c_{accept}$

Number of splits: $\log(2^{O(f(n))}) = O(f(n))$

Remembering the branch at every step costs exponential space

Each split must remember a "c_m" config $= O(f(n))$ space

- Instead, we check whether:

So long as we save the intermediate config

$c_{start} \xrightarrow{2^{O(f(n))}/2 \text{ steps}} c_m \xrightarrow{2^{O(f(n))}/2 \text{ steps}} c_{accept}$

Remembering these steps costs half the space …

… and we can <u>reuse</u> that space to check the second half

<u>Total</u>:
$O(f(n)) * O(f(n))$
$= \boldsymbol{O(f^2(n))}$ **space**
(Savitch's Thm)

- Keep dividing …

$c_{start} \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow c_{accept}$

Formally: A "Yielding" Algorithm

Start config | End config | # steps

Base case

CANYIELD = "On input c_1, c_2, and t:

1. If $t = 1$, then test directly whether $c_1 = c_2$ or whether c_1 yields c_2 in one step according to the rules of N. *Accept* if either test succeeds; *reject* if both fail.
2. If $t > 1$, then for each configuration c_m of N using space $f(n)$:
3.   Run CANYIELD$(c_1, c_m, \frac{t}{2})$.
4.   Run CANYIELD$(c_m, c_2, \frac{t}{2})$.
5.   If steps 3 and 4 both accept, then *accept*.
6. If haven't yet accepted, *reject*."

What's the middle config? Try them all
(it doesn't use any more space, per loop)

"divide and conquer"

Savitch's Theorem: Proof

- Let N be an NTM deciding language A in space $f(n)$
- Construct equivalent deterministic TM M using $O(f^2(n))$ space:

M = "On input w:
  1. Output the result of CANYIELD$(c_{\text{start}}, c_{\text{accept}}, 2^{df(n)})$."

Extra d constant depends on size of tape alphabet

  - c_{start} = start configuration of N
  - c_{accept} = new accepting config where all N's accepting configs go

PSPACE

DEFINITION

PSPACE is the class of languages that are decidable in polynomial space on a deterministic Turing machine. In other words,

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

NPSPACE

Analogous to **P** and **NP** for time complexity

DEFINITION

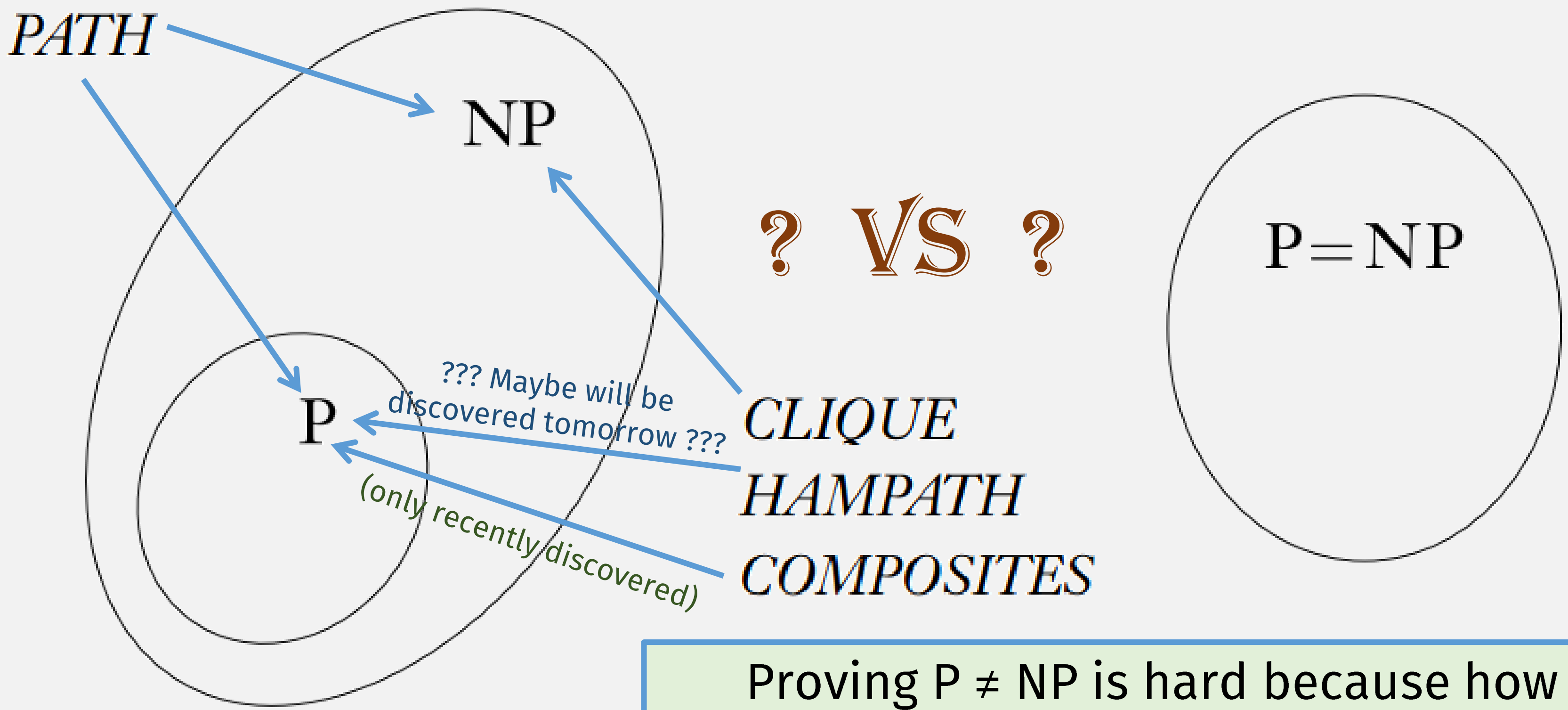
NPSPACE is the class of languages that are decidable in polynomial space on a nondeterministic Turing machine. In other words,

$$\text{NPSPACE} = \bigcup_k \text{NSPACE}(n^k).$$

But **P** ⊆ **PSPACE** and **NP** ⊆ **NPSPACE**

- Because each step can use at most one extra tape cell
- But space can be re-used

Flashback: Does P = NP?



Proving P ≠ NP is hard because how do you prove an algorithm <u>doesn't</u> have a poly time algorithm?
(in general it's hard to prove that something <u>doesn't</u> exist)

PSPACE = NPSPACE ?

- **PSPACE:** langs decidable in poly space on <u>deterministic</u> TM

- **NPSPACE:** langs decidable in poly space on <u>nondeterministic</u> TM

<u>Theorem</u>: **PSPACE = NPSPACE** **!!!**

<u>Proof</u>: By Savitch's Theorem!

THEOREM

Savitch's theorem For any function $f: \mathcal{N} \longrightarrow \mathcal{R}^+$, where $f(n) \geq n$,

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)).$$

Space vs Time

- $\mathbf{P} \subseteq \mathbf{PSPACE}$ and $\mathbf{NP} \subseteq \mathbf{NPSPACE}$
  - Because each step can use at most one extra tape cell
  - And space can be re-used

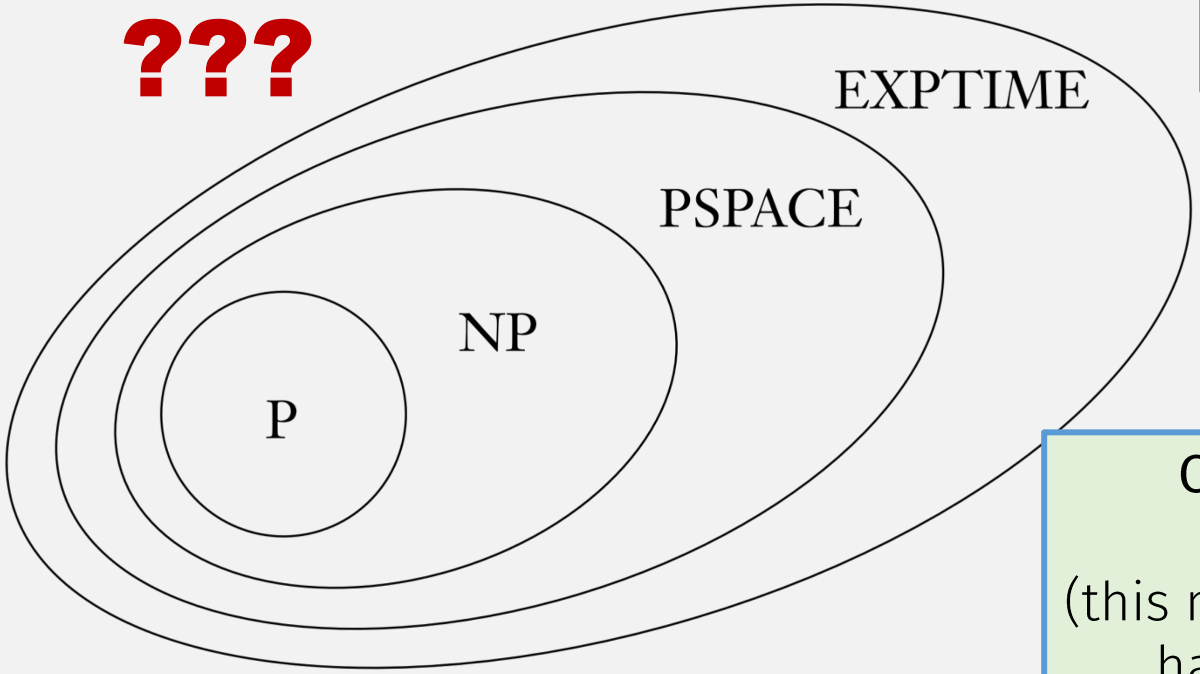
- $\mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$
  - Because an $f(n)$ space TM has $2^{O(f(n))}$ possible configurations
  - And a halting TM cannot repeat a configuration

- We already know $\mathbf{P} \subseteq \mathbf{NP}$ and $\mathbf{PSPACE} = \mathbf{NPSPACE}$ … so:

$$\boxed{\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE} \subseteq \mathbf{EXPTIME}}$$

Space vs Time: Conjecture

???



Researchers believe these are all completely contained within each other

But this is an open conjecture!

Only known result so far is:
$\mathbf{P \subset EXPTIME}$
(this means some problems provably have no poly time algorithm!)

$$\mathbf{P \subset NP \subset PSPACE = NPSPACE \subset EXPTIME}$$

Last Quiz 5/11

In gradescope