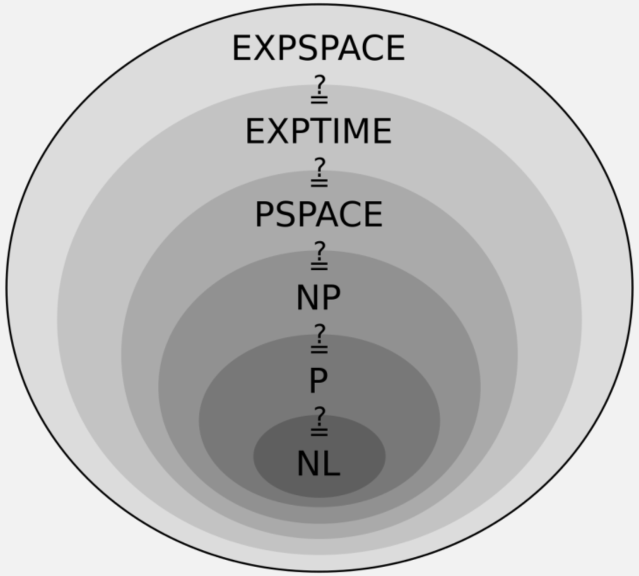
UMB CS622

Hierarchy Theorems

Monday, December 6, 2021

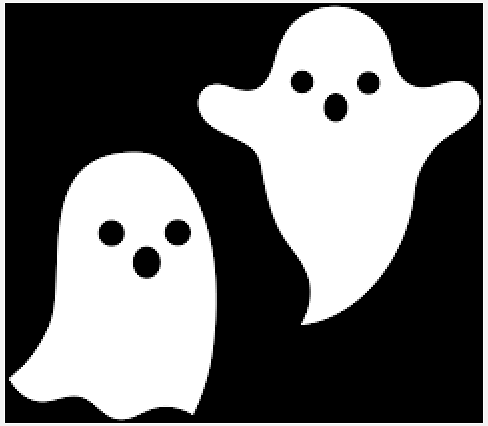


Announcements

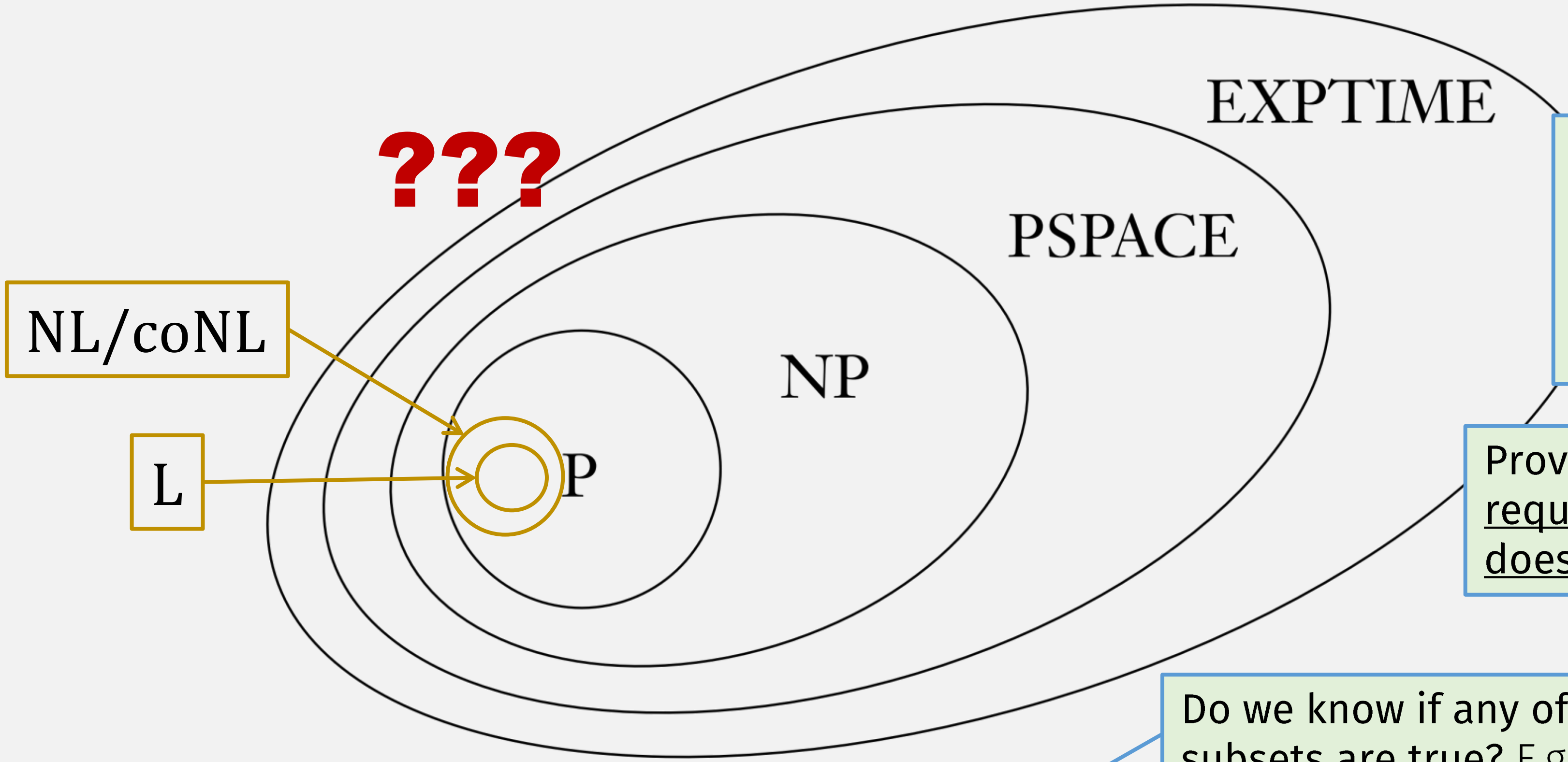
- ~~HW 9~~
  - ~~Due Tues 11/30 11:59pm EST~~
- HW 10
  - Due Tues 12/7 11:59pm EST
- HW 11
  - Out Wed 12/8
  - Due Tues 12/14 11:59pm EST

Flashback: Is SAT Intractable? (Not in $\mathbf{P}$?)

- There's no known poly time algorithm that decides SAT
- But it's hard to prove that an algorithm doesn't exist



Last Time: Space vs Time: Conjecture



So far, only if we "skip" steps:

- **NL ⊂ PSPACE**
- **PSPACE ⊂ EXPSPACE**
- **P ⊂ EXPTIME**

Proving is difficult because it requires showing that an algorithm doesn't exist (e.g., poly time)!

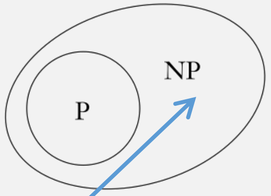
Do we know if any of these subsets are true? E.g., **P ⊂ NP**

We think? **L ⊂ NL = coNL ⊂ P ⊂ NP ⊂ PSPACE = NPSPACE ⊂ EXPTIME**

We know: **L ⊆ NL = coNL ⊆ P ⊆ NP ⊆ PSPACE = NPSPACE ⊆ EXPTIME**

How to Prove an Algorithm "Doesn't Exist"

1. <u>Prove containment</u> of two language complexity classes,
   - e.g, if $\mathbf{P} \subset \mathbf{NP}$



2. <u>Prove completeness</u> of a language in the larger class,
   - e.g, and if $SAT \in \mathbf{NP}$
   - and SAT is $\mathbf{NP}$-hard

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.

3. <u>Conclude</u> that the language cannot be in the smaller class
   - e.g, then $SAT \notin \mathbf{P}$
   - i.e., SAT has no poly time algorithm
   - (see also HW 9, problem # 2, part 2 for related problem)

THEOREM

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

- Prove that if $\mathbf{P} \neq \mathbf{NP}$, then 3NODES cannot be $\mathbf{NP}$-complete.

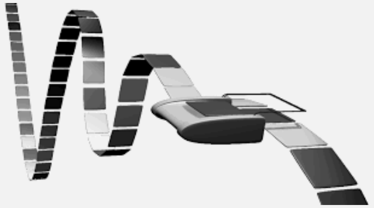
Theorems

$$\text{PSPACE} \subsetneq \text{EXPSPACE}$$

$$\text{P} \subsetneq \text{EXPTIME}$$

Could help prove that some language doesn't have a poly time algorithm

How Much Is a Tape Cell Worth?



- Does giving a TM "more space" make it "more powerful"?
  - I.e., does it increase the # of problems it can solve?
- What if we only give a TM 1 more tape cell?
  - (Might not help in some cases?)
- Can we formalize "more space" and "more powerful"?

Space Hierarchy Theorem

THEOREM

Space hierarchy theorem For any space constructible function $f: \mathcal{N} \longrightarrow \mathcal{N}$, a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

Flashback: Big-O Notation

Let f and g be functions $f, g\colon \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $\boldsymbol{f(n) = O(g(n))}$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$,

$$f(n) \leq c\, g(n).$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

"only care about **large** n"

Flashback: Small-o Notation

Let f and g be functions $f, g\colon \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $\boldsymbol{f(n) = o(g(n))}$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

In other words, $f(n) = o(g(n))$ means that for any real number $c > 0$, a number n_0 exists, where $f(n) < c\, g(n)$ for all $n \geq n_0$.

<u>Analogy</u>

- Big-O : ≤
- Small-o : <

Let f and g be functions $f, g\colon \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $\boldsymbol{f(n) = O(g(n))}$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$,

$$f(n) \leq c\, g(n).$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

Space Hierarchy Theorem

???

THEOREM

Space hierarchy theorem For any space constructible function $f : \mathcal{N} \longrightarrow \mathcal{N}$, a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

Flashback: Computable Functions

- **A TM that** (instead of accept/reject) **"outputs" final tape contents**

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a ***computable function*** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Space Constructible Functions

DEFINITION

A function $f: \mathcal{N} \longrightarrow \mathcal{N}$, where $f(n)$ is at least $O(\log n)$, is called ***space constructible*** if the function that maps the string 1^n to the binary representation of $f(n)$ is computable in space $O(f(n))$.

Function #1: $f(n)$

(Computable) **Function #2** (a TM)

Input n: unary

Output $f(n)$: binary

Space usage: $O(f(n))$

Unary representation

Space Constructible Function Example

Let $f(n) = n^2$

Input n (base 10)	**Input n** (unary)	**Output n^2** (base 10)	**Output n^2** (binary)
1	**1**	**1**	**1**

Space Constructible Function Example

Let $f(n) = n^2$

Input n (base 10)	**Input n** (unary)	**Output n^2** (base 10)	**Output n^2** (binary)
1	1	1	1
2	**11**	**4**	**100**

Space Constructible Function Example

Let $f(n) = n^2$

Input n (base 10)	**Input n** (unary)	**Output n^2** (base 10)	**Output n^2** (binary)
1	1	1	1
2	11	4	100
3	**111**	**9**	**1001**

Space Constructible Function Example

Let $f(n) = n^2$

Input n (base 10)	**Input n** (unary)	**Output n^2** (base 10)	**Output n^2** (binary)
1	1	1	1
2	11	4	100
3	111	9	1001
	…		
16	**1111111111111111**	**256**	**100000000** (2^8)

Space Constructible Function Example

Let $f(n) = n^2$

On input 1^n (n in unary notation):

- Convert to binary by …
  - Counting the # of 1s
  - (counters require) $\log(n)$ space
- Multiply (binary nums) $n * n$:
  - Quadratic (grade school) algorithm
  - $\log^2(n)$ space

Total space: $O(\log^2(n))$

Space allowed: $O(n^2)$

Don't count input space $O(n)$

Otherwise, cant compute $\log n$ in $\log n$ space

Space Constructible Function Example

Let $f(n) = n^{k}$

On input 1^{n} (n in unary notation):
- Convert to binary by …
  - Counting the # of 1s
  - (counters require) $\log(n)$ space
- Repeat k times: multiply by n:
  - Quadratic (grade school) algorithm
  - $\log^{k}(n)$ space

Total space: $O(\log^{k}(n))$

Space allowed: $O(n^{k})$

Don't count input space $O(n)$

Otherwise, cant compute $\log n$ in $\log n$ space

Space Hierarchy Theorem

THEOREM

Space hierarchy theorem For any space constructible function $f: \mathcal{N} \longrightarrow \mathcal{N}$, a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

Space Hierarchy Theorem: Proof Plan

THEOREM

Space hierarchy theorem For any space constructible function $f : \mathcal{N} \longrightarrow \mathcal{N}$, a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

- Let A be a language with decider D that runs in $O(f(n))$ space
- Make sure D rejects something from every $o(f(n))$ language ...
- ... using diagonalization!

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$		$\vdots$			$\ddots$

Flashback: Diagonalization with TMs

Diagonal: Result of Giving a TM its own Encoding as Input

All TM Encodings

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$		$\vdots$			$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		?	
$\vdots$		$\vdots$					

opposites

All TMs

Try to construct "opposite" TM

What should happen here?

TM D can't exist!

It must both accept and reject!

Diagonalization with $o(f(n))$ TMs?

<u>Diagonal</u>: Result of Giving a TM its own Encoding as Input

All TM Encodings

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	<u>*accept*</u>	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	<u>*accept*</u>	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	<u>*reject*</u>	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	<u>*reject*</u>		*accept*	
$\vdots$			$\vdots$		$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		<u>*reject*</u>	
$\vdots$			$\vdots$				

Opposites, if M is $o(f(n))$

All TMs

Doesn't matter!

Try to construct "opposite" TM

TM D <u>can</u> exist!

But only for $o(f(n))$ TMs!

Space Hierarchy Theorem: Diagonalization

- Let A be a language with decider D that runs in $O(f(n))$ space
- Make sure D rejects something from every $o(f(n))$ language ...
- ... using diagonalization!

- If M is an $o(f(n))$ space TM ...

  ... make D differ from M on one input:

  ... <M> itself!
- Specifically D runs M with <M> and checks space usage is $o(f(n))$
- If M accepts <M> then D rejects <M>
  - and vice versa
- Then D cannot use $o(f(n))$ space!

<u>3 potential issues:</u>

1. **M uses more than $o(f(n))$ space**
   - D rejects M if it ever uses more than $f(n)$ space
2. **M uses more than $o(f(n))$ space for small n**
   - Accept all inputs with arbitrary padding <M>10*
3. **M might go into loop**
   - $f(n)$ space TM cannot run for more than $2^{f(n)}$ steps
   - So D runs M for only $2^{f(n)}$ steps

Space Hierarchy Theorem: Proof

THEOREM

Space hierarchy theorem For any space constructible function $f: \mathcal{N} \longrightarrow \mathcal{N}$, a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

PROOF The following $O(f(n))$ space algorithm D decides a language A that is not decidable in $o(f(n))$ space.

D = "On input w: ← $\langle M \rangle 10^*$

1. Let n be the length of w.
2. Compute $f(n)$ using space constructibility and mark off this much tape. If later stages ever attempt to use more, *reject*.
3. If w is not of the form $\langle M \rangle 10^*$ for some TM M, *reject*.
4. Simulate M on w while counting the number of steps used in the simulation. If the count ever exceeds $2^{f(n)}$, *reject*.
5. If M accepts, *reject*. If M rejects, *accept*."

Use only $f(n)$ space

Make sure input is long enough

Run for only $2^{f(n)}$ steps

Space Hierarchy Theorem: Corollary # 1

For any two functions $f_1, f_2 \colon \mathcal{N} \longrightarrow \mathcal{N}$, where $f_1(n)$ is $o(f_2(n))$ and f_2 is space constructible, $\text{SPACE}(f_1(n)) \subsetneq \text{SPACE}(f_2(n))$.

PROOF

$\subset$ that we want

- f_2 is space constructible, so by the Space Hierarchy Thm …

Space hierarchy theorem For any space constructible function $f \colon \mathcal{N} \longrightarrow \mathcal{N}$, a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

- … some lang A is decidable in $O(f_2(n))$ space but not $o(f_2(n))$
- So $A \in \text{SPACE}(f_2(n))$ but $A \notin \text{SPACE}(f_1(n))$
  - Because $f_1(n) = o(f_2(n))$
- Thus, $\text{SPACE}(f_1(n)) \neq \text{SPACE}(f_2(n))$
- So $\text{SPACE}(f_1(n)) \subset \text{SPACE}(f_2(n))$

Space Hierarchy Theorem: Corollary # 2

For any two real numbers $0 \leq \epsilon_1 < \epsilon_2$, $\text{SPACE}(n^{\epsilon_1}) \subsetneq \text{SPACE}(n^{\epsilon_2})$.

Proof

- From previous corollary …

For any two functions $f_1, f_2 \colon \mathcal{N} \longrightarrow \mathcal{N}$, where $f_1(n)$ is $o(f_2(n))$ and f_2 is space constructible, $\text{SPACE}(f_1(n)) \subsetneq \text{SPACE}(f_2(n))$.

- Earlier we showed that n^k is space constructible
- So for any two natural numbers $k_1 < k_2$:
  - $\text{SPACE}(n^{k1}) \subset \text{SPACE}(n^{k2})$
  - Because $n^{k1} = o(n^{k2})$
- Similarly, for two rationals $c_1 < c_2$: $\text{SPACE}(n^{c1}) \subset \text{SPACE}(n^{c2})$
- Two rationals exist between any two reals $\varepsilon_1 < c_1 < c_2 < \varepsilon_2$:
  - So $\text{SPACE}(n^{\varepsilon 1}) \subset \text{SPACE}(n^{\varepsilon 2})$

Space Hierarchy Theorem: Corollary # 3

$$\text{PSPACE} \subsetneq \text{EXPSPACE}$$

Proof

- **PSPACE** = SPACE(n^k)
- **EXPSPACE** = SPACE(2^n^k)
- $n^k = o(2^{n^k})$
- By Space Hierarchy Theorem …

Space hierarchy theorem For any space constructible function $f : \mathcal{N} \longrightarrow \mathcal{N}$, a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

- A language A is decidable in $O(2^{n^k})$ space but not $o(2^{n^k})$
- So $A \in$ **EXPSPACE** but $A \notin$ **PSPACE**
- So **EXPSPACE ≠ PSPACE**

Space Hierarchy Theorem: <u>Corollary</u> # 4

$$\text{NL} \subsetneq \text{PSPACE}$$

How does this help show that some lang <u>doesn't</u> have an algorithm with some complexity?

<u>Proof</u>

- **NL** = NSPACE(log n)
- By Savitch's Theorem …

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

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

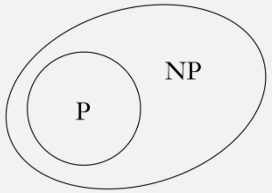
- **NL** = NSPACE(log n) ⊆ SPACE($\log^2 n$)
- By Space Hierarchy Theorem …

Space hierarchy theorem For any space constructible function $f: \mathcal{N} \longrightarrow \mathcal{N}$, a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

- SPACE($\log^2 n$) ⊂ SPACE(n) ⊂ SPACE(n^k) = **PSPACE**

How to Prove an Algorithm "Doesn't Exist"

1. Prove containment of two language complexity classes,
   - e.g, if $\mathbf{P} \subset \mathbf{NP}$



2. Prove completeness of a language in the larger class,
   - e.g, and if $SAT \in \mathbf{NP}$
   - and SAT is $\mathbf{NP}$-hard

3. Conclude that the language cannot be in the smaller class
   - e.g, then $SAT \notin \mathbf{P}$
   - i.e., SAT has no poly time algorithm

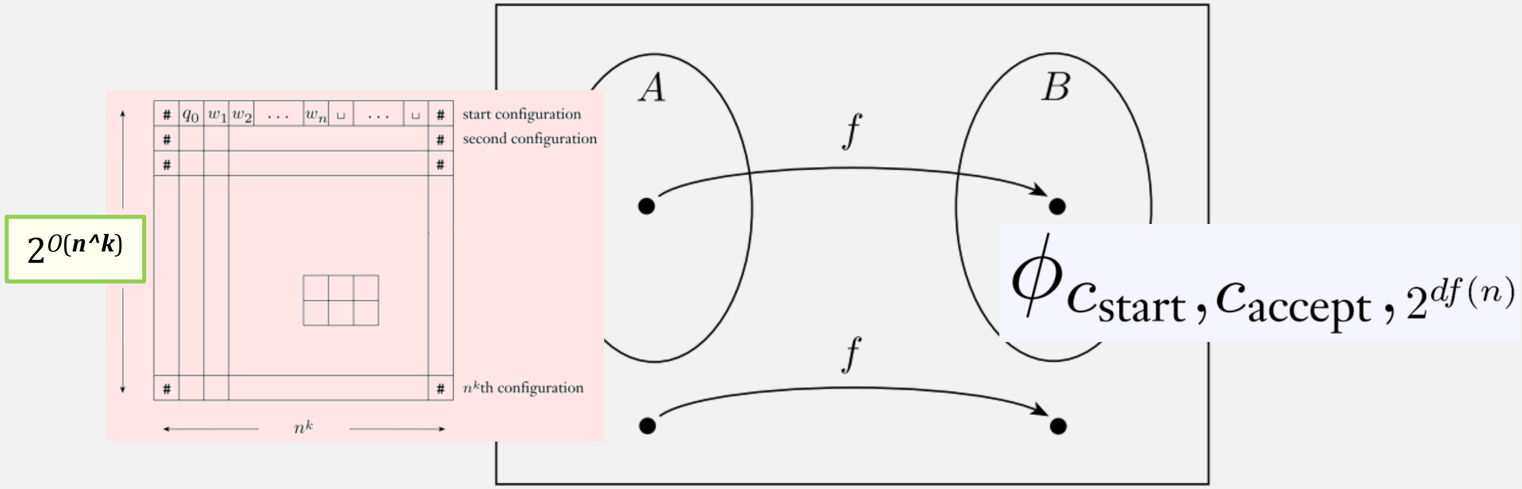
Flashback: **PSPACE**-Completeness

DEFINITION

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

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

If B merely satisfies condition 2, we say that it is ***PSPACE-hard***.



THEOREM

TQBF is PSPACE-complete.

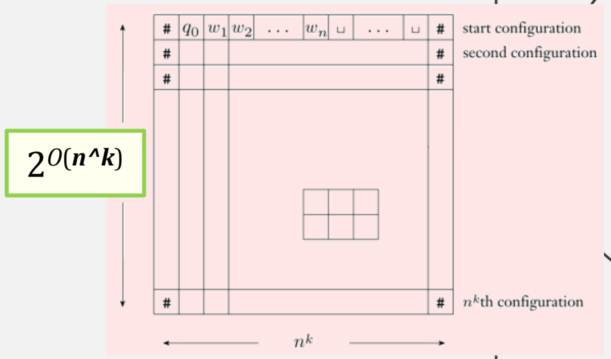
PSPACE-Completeness w.r.t. $\leq_L$

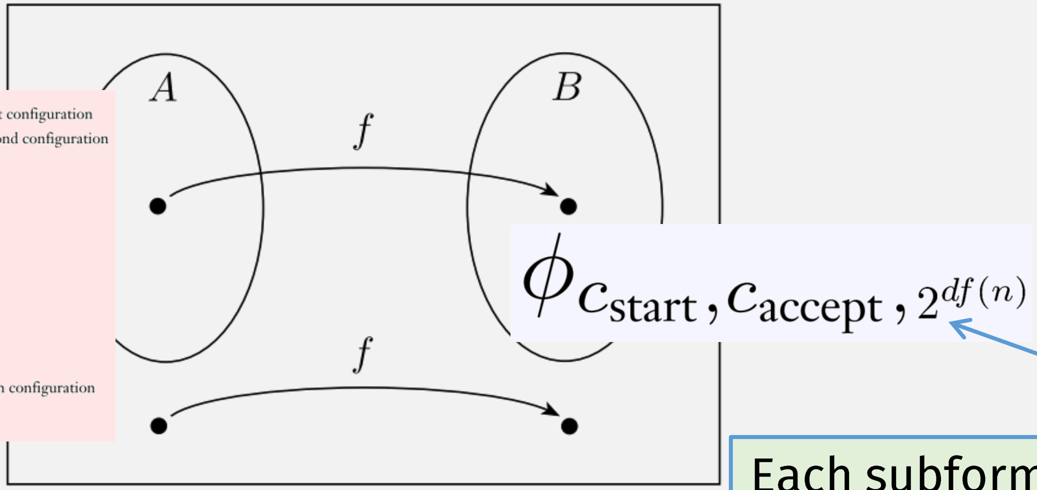
DEFINITION

A language B is ***PSPACE-complete*** if it satisfies two conditions: *with respect to log space reducibility*

1. B is in PSPACE, and
2. every A in PSPACE is ~~polynomial time~~ log space reducible to B.

If B merely satisfies condition 2, we say that it is ***PSPACE-hard***.





THEOREM

TQBF is PSPACE-complete.
with respect to log space reducibility

Each subformula can be generated in log space

Space Hierarchy Theorem: Corollary # 4

$$\mathrm{NL} \subsetneq \mathrm{PSPACE}$$

- $TQBF \notin$ **NL**
- Because $TQBF$ is **PSPACE**-Complete (w.r.t log space reducibility)
- So if $TQBF \in$ **NL**
  - Then every **PSPACE** problem is in **NL**
  - and **NL** = **PSPACE**

An **NL** algorithm for $TQBF$ doesn't exist!

Now can we prove that a language doesn't have a poly time algorithm?

Time Constructible Functions

DEFINITION

A function $t: \mathcal{N} \longrightarrow \mathcal{N}$, where $t(n)$ is at least $O(n \log n)$, is called ***time constructible*** if the function that maps the string 1^n to the binary representation of $t(n)$ is computable in time $O(t(n))$.

Function #1: $t(n)$

(Computable) **Function #2** (a TM)

Input n: unary

Output $t(n)$: binary

Space usage: $O(t(n))$

Unary representation

Time Constructible Function Example

Let $t(n) = n^2$

On input $\mathbf{1}^{\boldsymbol{n}}$ ($\boldsymbol{n}$ in unary notation):

- Convert to binary by …
  - Counting the # of 1s
  - Each counter increment takes:
    - $\log(\boldsymbol{n})$ steps
  - Total: $O(\boldsymbol{n}\log(\boldsymbol{n}))$
- Multiply $n * n$:
  - Quadratic (grade school) algorithm
  - $O(\log^2(\boldsymbol{n}))$ steps

Total steps: $O(\boldsymbol{n}\log(\boldsymbol{n})) + O(\log^2(\boldsymbol{n})) = O(\boldsymbol{n}\log(\boldsymbol{n}))$

Steps allowed: $O(\boldsymbol{n}^2)$

Time Hierarchy Theorem

THEOREM

Time hierarchy theorem For any time constructible function $t\colon \mathcal{N} \longrightarrow \mathcal{N}$, a language A exists that is decidable in $O(t(n))$ time but not decidable in time $o(t(n)/\log t(n))$.

Time is "weaker"; Must increase # steps by at least log $t(n)$ to get extra "power"
(i.e., decide additional languages)

Time Hierarchy Theorem Proof

D takes $t(n)$ steps …

PROOF The following $O(t(n))$ time algorithm D decides a language A that is not decidable in $o(t(n)/\log t(n))$ time.

Overhead of the counter

D = "On input w:

Need to limit # of steps

1. Let n be the length of w.
2. Compute $t(n)$ using time constructibility and store the value $\lceil t(n)/\log t(n) \rceil$ in a binary counter. Decrement this counter before each step used to carry out stages 4 and 5. If the counter ever hits 0, *reject*.
3. If w is not of the form $\langle M \rangle 10^*$ for some TM M, *reject*.
4. Simulate M on w.
5. If M accepts, then *reject*. If M rejects, then *accept*."

… to simulate $t(n)/\log(t(n))$ steps of some M

A TM simulating another TM is not free!

(This style of diagonalization proof won't work to prove $\mathbf{P} \subset \mathbf{NP}$)

Time Hierarchy Corollary # 1

For any two functions $t_1, t_2 \colon \mathcal{N} \longrightarrow \mathcal{N}$, where $t_1(n)$ is $o(t_2(n)/\log t_2(n))$ and t_2 is time constructible, $\mathrm{TIME}(t_1(n)) \subsetneq \mathrm{TIME}(t_2(n))$.

Time Hierarchy Corollary # 2

For any two real numbers $1 \leq \epsilon_1 < \epsilon_2$, we have $\mathrm{TIME}(n^{\epsilon_1}) \subsetneq \mathrm{TIME}(n^{\epsilon_2})$.

Time Hierarchy Corollary # 3

$$P \subsetneq \text{EXPTIME}$$

So there exists some language that does not have a poly time algorithm!

(Next time, we see an example)

Check-in Quiz 12/6

On gradescope