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 $\text{SAT}$ Intractable? (Not in $\text{P}$?)

• There’s no known poly time algorithm that decides $\text{SAT}$

• But it’s hard to prove that an algorithm doesn’t exist
Last Time: Space vs Time: **Conjecture**

We think:

\[ L \subset NL = \text{coNL} \subset P \subset NP \subset \text{PSPACE} = \text{NPSPACE} \subset \text{EXPTIME} \]

We know:

\[ L \subset NL = \text{coNL} \subset P \subset NP \subset \text{PSPACE} = \text{NPSPACE} \subset \text{EXPTIME} \]

So far, only if we “skip” steps:
- \( NL \subset \text{PSPACE} \)
- \( \text{PSPACE} \subset \text{EXPSPACE} \)
- \( P \subset \text{EXPTIME} \)

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., \( P \subset NP \)
How to Prove an Algorithm “Doesn’t Exist”

1. Prove containment of two language complexity classes,
   • e.g. if \( P \subset NP \)

2. Prove completeness of a language in the larger class,
   • e.g. and if \( SAT \in NP \)
   • and \( SAT \) is \( NP \)-hard

3. Conclude that the language cannot be in the smaller class
   • e.g. then \( SAT \notin P \)
   • i.e., \( SAT \) has no poly time algorithm
   • (see also HW 9, problem # 2, part 2 for related problem)
     • Prove that if \( P \neq NP \), then 3NODES cannot be \( NP \)-complete.
Theorems

\[ \text{PSPACE} \subseteq \text{EXPSPACE} \]

\[ \text{P} \subsetneq \text{EXPTIME} \]

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 $f : \mathcal{N} \to \mathcal{N}$, 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 $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$. Say that $f(n) = O(g(n))$ if positive integers $c$ and $n_0$ exist such that for every integer $n \geq n_0$,

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

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.
**Flashback: Small-$\circ$ Notation**

Let $f$ and $g$ be functions $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$. Say that $f(n) = o(g(n))$ 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 $n \geq n_0$.

**Analogy**
- **Big-$O$**: $\leq$
- **Small-$o$**: $<$

Let $f$ and $g$ be functions $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$. Say that $f(n) = O(g(n))$ 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 \( f : \mathbb{N} \rightarrow \mathbb{N} \), 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 $f : \Sigma^* \rightarrow \Sigma^*$ 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 $f: \mathbb{N} \rightarrow \mathbb{N}$, 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: $f(n)$

Function #2 (a TM)

Input $n$: unary

Output $f(n)$: binary

Space usage: $O(f(n))$
Space Constructible Function Example

Let \( f(n) = n^2 \)

<table>
<thead>
<tr>
<th>Input ( n ) (base 10)</th>
<th>Input ( n ) (unary)</th>
<th>Output ( n^2 ) (base 10)</th>
<th>Output ( n^2 ) (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Space Constructible Function Example

Let $f(n) = n^2$

<table>
<thead>
<tr>
<th>Input $n$ (base 10)</th>
<th>Input $n$ (unary)</th>
<th>Output $n^2$ (base 10)</th>
<th>Output $n^2$ (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>4</td>
<td>100</td>
</tr>
</tbody>
</table>
Space Constructible Function Example

Let $f(n) = n^2$

<table>
<thead>
<tr>
<th>Input $n$ (base 10)</th>
<th>Input $n$ (unary)</th>
<th>Output $n^2$ (base 10)</th>
<th>Output $n^2$ (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>111</td>
<td>9</td>
<td>1001</td>
</tr>
</tbody>
</table>
Space Constructible Function Example

Let $f(n) = n^2$

<table>
<thead>
<tr>
<th>Input $n$ (base 10)</th>
<th>Input $n$ (unary)</th>
<th>Output $n^2$ (base 10)</th>
<th>Output $n^2$ (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>111</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>11111111111111111</td>
<td>256</td>
<td>1000000000 (2^8)</td>
</tr>
</tbody>
</table>
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) \( n \times n \):
  - Quadratic (grade school) algorithm
  - \( \log^2(n) \) space

Total space: \( O(\log^2(n)) \)
Space allowed: \( O(n^2) \)
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: $O(n^k)$
**Space Hierarchy Theorem**

**THEOREM**

*Space hierarchy theorem*  
For any space constructible function $f: \mathcal{N} \rightarrow \mathcal{N}$, 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 $f : \mathbb{N} \rightarrow \mathbb{N}$, 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 a decider $D$ that runs in $O(f(n))$ space.
- Make sure $D$ rejects something from every $o(f(n))$ language...
- ... using diagonalization!
### Flashback: Diagonalization with TMs

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

<table>
<thead>
<tr>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
</tr>
<tr>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
</tr>
<tr>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**What should happen here?**

**It must both accept and reject!**

**TM $D$ can’t exist!**

**Try to construct “opposite” TM**

**All TMs**

**opposites**

**All TM Encodings**
Diagonalization with $o(f(n))$ TMs?

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

<table>
<thead>
<tr>
<th></th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>$\langle D \rangle$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$M_2$</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$M_3$</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$M_4$</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
</tr>
<tr>
<td>$D$</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td></td>
</tr>
</tbody>
</table>

Try to construct “opposite” TM

TM $D$ can exist!

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

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

All TMs

Doesn’t matter!
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!

3 potential issues:
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 $f: \mathbb{N} \rightarrow \mathbb{N}$, 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.

$$D = \text{“On input } w: \langle M \rangle 10^*$$

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.”
Space Hierarchy Theorem: Corollary # 1

For any two functions $f_1, f_2 : \mathbb{N} \rightarrow \mathbb{N}$, where $f_1(n)$ is $o(f_2(n))$ and $f_2$ is space constructible, $\text{SPACE}(f_1(n)) \subseteq \text{SPACE}(f_2(n))$.

**Proof**

• $f_2$ is space constructible, so by the Space Hierarchy Thm ...

**Space hierarchy theorem** For any space constructible function $f : \mathbb{N} \rightarrow \mathbb{N}$, 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 $f_1, f_2: \mathcal{N} \rightarrow \mathcal{N}$, 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 $k_1 < k_2$:
  - $\text{SPACE}(n^{k_1}) \subset \text{SPACE}(n^{k_2})$
  - Because $n^{k_1} = o(n^{k_2})$

- Similarly, for two rationals $c_1 < c_2$: $\text{SPACE}(n^{c_1}) \subset \text{SPACE}(n^{c_2})$

- Two rationals exist between any two reals $\epsilon_1 < c_1 < c_2 < \epsilon_2$:
  - So $\text{SPACE}(n^{\epsilon_1}) \subset \text{SPACE}(n^{\epsilon_2})$
Space Hierarchy Theorem: **Corollary # 3**

\[ \text{PSPACE} \subsetneq \text{EXPSPACE} \]

**Proof**

- **PSPACE** = SPACE\((n^k)\)
- **EXPSPACE** = SPACE\((2^k)\)
- \(n^k = o(2^k)\)
- By Space Hierarchy Theorem ...

**Space hierarchy theorem**  For any space constructible function \(f : \mathbb{N} \rightarrow \mathbb{N}\), 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 \(O(2^k)\) space but not \(o(2^k)\)
- So \(A \in \text{EXPSPACE}\) but \(A \notin \text{PSPACE}\)
- So \(\text{EXPSPACE} \neq \text{PSPACE}\)
Space Hierarchy Theorem: Corollary # 4

\[ \text{NL} \subsetneq \text{PSPACE} \]

**Proof**

- **NL** = NSPACE(log \(n\))
- By Savitch’s Theorem ...

\[ \text{Savitch’s theorem} \quad \text{For any function } f : N \rightarrow R^+, \text{ where } f(n) \geq n, \text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)). \]

- **NL** = NSPACE(log \(n\)) \(\subseteq\) SPACE(log² \(n\))
- By Space Hierarchy Theorem ...

\[ \text{Space hierarchy theorem} \quad \text{For any space constructible function } f : N \rightarrow N, \text{a language } A \text{ exists that is decidable in } O(f(n)) \text{ space but not in } o(f(n)) \text{ space.} \]

- SPACE(log² \(n\)) \(\subset\) SPACE(\(n\)) \(\subset\) SPACE(\(n^k\)) = PSPACE

How does this help show that some lang doesn’t have an algorithm with some complexity?
How to Prove an Algorithm “Doesn’t Exist”

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

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

3. **Conclude** that the language cannot be in the smaller class
   - e.g., then $SAT \notin P$
   - 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:

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. with respect to log space reducibility

Each subformula can be generated in log space
Space Hierarchy Theorem: Corollary # 4

\[ \text{NL} \subset \text{PSPACE} \]

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

An \text{NL} algorithm for \( TQBF \) doesn’t exist!

Now can we prove that a language doesn’t have a poly time algorithm?
A function $t: \mathbb{N} \rightarrow \mathbb{N}$, where $t(n)$ is at least $O(n \log n)$, is called \textit{time constructible} if the function that maps the string $1^n$ to the binary representation of $t(n)$ is computable in time $O(t(n))$. 

\textbf{Definition}

Function \#1: $t(n)$

(Computable) Function \#2 (a TM)

Input $n$: unary

Output $t(n)$: binary

Space usage: $O(t(n))$
Time Constructible Function Example

Let $t(n) = n^2$

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

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

Total steps: $O(n \log(n)) + O(\log^2(n)) = O(n \log(n))$
Steps allowed: $O(n^2)$
Time Hierarchy Theorem

**Theorem**

**Time hierarchy theorem**  For any time constructible function $t: \mathcal{N} \rightarrow \mathcal{N}$, 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**

**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.

$D =$ “On input $w$:

1. Let $n$ be the length of $w$.
2. Compute $t(n)$ using time constructibility and store the value $[t(n)/\log t(n)]$ 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.”

**Note:** A TM simulating another TM is not free! (This style of diagonalization proof won’t work to prove $P \subset NP$)
Time Hierarchy Corollary # 1

For any two functions $t_1, t_2: \mathbb{N} \rightarrow \mathbb{N}$, where $t_1(n)$ is $o(t_2(n)/\log t_2(n))$ and $t_2$ is time constructible, $\text{TIME}(t_1(n)) \subsetneq \text{TIME}(t_2(n))$. 
Time Hierarchy Corollary # 2

For any two real numbers $1 \leq \epsilon_1 < \epsilon_2$, we have $\text{TIME}(n^{\epsilon_1}) \not\subset \text{TIME}(n^{\epsilon_2})$. 
Time Hierarchy **Corollary # 3**

\[ P \not\subseteq \text{EXPTIME} \]

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