An Intractable Problem

Wednesday, December 8, 2021

“I can't find an efficient algorithm, but neither can all these furious people.”
Announcements

• HW 10 in
  • Due Tues 12/7 11:59pm EST

• HW 11 out
  • Due Tues 12/14 11:59pm EST

• Course evaluation at end of class today
Last Time: Nonexistent Algorithms

• It’s hard to prove that something doesn’t exist

• For algorithms/deciders, the best we can say is usually:
  • “There’s no known poly time algorithm that decides ... e.g., SAT”
Last Time: Proving a Nonexistent Algorithm

1. **Prove proper containment** of two complexity classes,
   • e.g., if $A \subset B$

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

3. **Conclude** that the language cannot be in the smaller class
   • e.g., then $L \notin A$, i.e., $L$ has no decider of some complexity!
**Last Time:** Hierarchy Theorems

**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.

**NL \nsubseteq PSPACE**

**PSPACE \nsubseteq EXPSPACE**

**Theorem**

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

**P \nsubseteq EXPSPACE**
Last Time: A Nonexistent Algorithm

1. **Prove proper containment** of two complexity classes,
   - e.g., $\text{NL} \subset \text{PSPACE}$

2. **Prove completeness** of a language in the larger class,
   - e.g., $\text{TQBF} \in \text{PSPACE}$ and $\text{TQBF}$ is $\text{PSPACE}$-hard

3. **Conclude** that the language cannot be in the smaller class
   - e.g., $\text{TQBF} \notin \text{NL}$,
   - i.e., $\text{TQBF}$ has no logspace NTM decider!

What about a nonexistent poly time algorithm?
Thm: $EQ_{REX^\uparrow}$ is Intractable! (not in $P$!)

$EQ_{REX^\uparrow} = \{(Q, R) \mid Q$ and $R$ are equivalent regular expressions with exponentiation$\}$
A Nonexistent Polynomial Time Algorithm

1. Prove proper containment of two complexity classes,
   - e.g, $P \subset \cdots$

2. Prove completeness of a language in the larger class,
   - e.g, $EQ_{\text{REX}^*} \in \cdots$ and $EQ_{\text{REX}^*}$ is $\cdots$-hard

3. Conclude that the language cannot be in the smaller class
   - e.g, $EQ_{\text{REX}^*} \notin P$,
   - i.e., $EQ_{\text{REX}^*}$ has no poly time decider!
A Nonexistent Polynomial Time Algorithm

1. Prove proper containment of two complexity classes,
   - e.g., $P \subseteq \text{EXPSPACE}$

2. Prove completeness of a language in the larger class,
   - e.g., $EQ_{\text{REX}^\uparrow} \in \text{EXPSPACE}$ and $EQ_{\text{REX}^\uparrow}$ is $\text{EXPSPACE}$-hard

3. Conclude that the language cannot be in the smaller class
   - e.g., $EQ_{\text{REX}^\uparrow} \notin P$,
   - i.e., $EQ_{\text{REX}^\uparrow}$ has no poly time decider!
\( P \subset \text{EXPSPACE} \)

- \( P \subseteq \text{PSPACE} \), because
  - \( \Rightarrow \) A poly time algorithm uses at most poly space
  - \( \Leftarrow \) But a poly space algorithm can take more than poly time
    - Because space can be reused

- And Space Hierarchy Theorem says: \( \text{PSPACE} \subset \text{EXPSPACE} \)

  **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.

- So \( P \subseteq \text{PSPACE} \subset \text{EXPSPACE} \)
A Nonexistent Polynomial Time Algorithm

1. Prove proper containment of two complexity classes,
   • e.g., $P \subset \text{EXPSPACE}$

2. Prove completeness of a language in the larger class,
   • e.g., $EQ_{\text{REX}} \in \text{EXPSPACE}$ and $EQ_{\text{REX}}$ is $\text{EXPSPACE}$-hard

3. Conclude that the language cannot be in the smaller class
   • e.g., $EQ_{\text{REX}} \notin P$,
   • i.e., $EQ_{\text{REX}}$ has no poly time decider!
Flashback: Regular Expressions

A regular expression is if $R$ is

1. $a$ for some $a$ in the alphabet $\Sigma$,
2. $\varepsilon$,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions,
5. $(R_1 \circ R_2)$, where $R_1$ and $R_2$ are regular expressions, or
6. $(R_1^*)$, where $R_1$ is a regular expression.
Flashback: **RegExpr → NFA**

**Regular expression** $R$ is a **regular expression** if $R$ is:

1. $a$ for some $a$ in the alphabet $\Sigma$,
2. $\varepsilon$,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions,
5. $(R_1 \circ R_2)$, where $R_1$ and $R_2$ are regular expressions,
6. $(R_1^*)$, where $R_1$ is a regular expression.
RegExpr→NFA is in PSPACE

• From HW10, Problem # 2
$EQ_{\text{NFA}}$ is in $\text{PSPACE}$

- Prove not $EQ_{\text{NFA}}$ is in $\text{PSPACE}$
  - From HW10, Problem # 3

- And prove $\text{PSPACE}$ closed under complement
  - From HW10, Problem # 1
$\overline{EQ_{NFA}}$ is in $\text{NPSPACE} (= \text{PSPACE})$
Flashback: Nondeterministic Space Usage

\[ ALL_{\text{NFA}} = \{ \langle A \rangle \mid A \text{ is an NFA and } L(A) = \Sigma^* \} \]

Nondeterministic decider for \( ALL_{\text{NFA}} \)

\( N = \) “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 \( M \):
   1. Nondeterministically select an input symbol and change the positions of the markers on \( M \)’s states to simulate reading that symbol.
3. 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.”

Additionally, need a counter to count to \( 2^q \):

- requires \( \log{(2^q)} = q \) extra space

Machine tracks “current” states of NFA:

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

Each loop uses only \( O(q) \) space!

So the whole machine runs in (nondeterministic) linear \( O(q) \) space!
\( \overline{EQ_{NFA}} \) is in \( \text{NPSPACE} (= \text{PSPACE}) \)

\[ N = \text{"On input } \langle N_1, N_2 \rangle, \text{ where } N_1 \text{ and } N_2 \text{ are NFAs:}\]

1. Place a marker on each of the start states of \( N_1 \) and \( N_2 \).
2. Repeat \( 2^{q_1 + q_2} \) times, where \( q_1 \) and \( q_2 \) are the numbers of states in \( N_1 \) and \( N_2 \):
3. Nondeterministically select an input symbol and change the positions of the markers on the states of \( N_1 \) and \( N_2 \) to simulate reading that symbol.
4. If at any point a marker was placed on an accept state of one of the finite automata and not on any accept state of the other finite automaton, \textit{accept}. Otherwise, \textit{reject}.”

Machine runs in:
- nondeterministic \( O(q) \) space
- deterministic \( O(q^2) \) space

Track 2 sets of "current" states
$EQ_{REX}$ is in **PSPACE**

From HW10, Problem # 4

1. Convert regular expressions to NFAs (**PSPACE**)

2. Check if NFAs are equivalent (**PSPACE**)

$EQ_{REX} = \{(Q, R) | Q$ and $R$ are equivalent regular expressions$\}$
Regular Expressions + Exponentiation

Let \( \uparrow \) be the *exponentiation operation*.

- If \( R \) is a regular expression, then

\[
R^k = R \uparrow k = \underbrace{R \circ R \circ \cdots \circ R}_k
\]

- I.e., exponentiation = concatenation \( k \) times

- So regular expressions with exponentiation ...
  - ... still equivalent to regular langs!
Thm: $EQ_{\text{REX}^\uparrow}$ is Intractable! (not in $\mathbf{P}$!)

$EQ_{\text{REX}^\uparrow} = \{ (Q, R) \mid Q$ and $R$ are equivalent regular expressions with exponentiation $\}$

Theorem

$EQ_{\text{REX}^\uparrow}$ is EXPSPACE-complete.
**EXPSPACE-Completeness**

**Definition**

A language $B$ is *EXPSPACE-complete* if

1. $B \in \text{EXPSPACE}$, and
2. every $A$ in EXPSPACE is polynomial time reducible to $B$.

**Theorem**

$EQ_{\text{REX}}^\uparrow$ is EXPSPACE-complete.
$E_{Q_{REX}^\uparrow}$ is in \textbf{EXPSPACE}

$E_{Q_{REX}^\uparrow} = \{ (Q, R) | Q \text{ and } R \text{ are equivalent regular expressions with exponentiation} \}$

Similar to $E_{Q_{REX}}$ 

\textbf{Problem #4} 

\begin{itemize}
  \item \begin{itemize}
    \item $E = \text{“On input } (R_1, R_2)\text{, where } R_1 \text{ and } R_2 \text{ are regular expressions with exponentiation:
    
      1. Convert } R_1 \text{ and } R_2 \text{ to equivalent regular expressions } B_1 \text{ and } B_2
      \text{ that use repetition instead of exponentiation.}
      
      2. Convert } B_1 \text{ and } B_2 \text{ to equivalent NFAs } N_1 \text{ and } N_2, \text{ using the conversion procedure given in the proof of Lemma 1.55.}
      
      3. Use the deterministic version of algorithm } N \text{ to determine whether } N_1 \text{ and } N_2 \text{ are equivalent.”
    \end{itemize}
  \end{itemize}

\textbf{From HW10}
**EXPSPACE-Completeness**

**Definition**
A language $B$ is *EXPSPACE-complete* if

1. $B \in \text{EXPSPACE}$, and
2. every $A$ in EXPSPACE is polynomial time reducible to $B$.

**Theorem**
$E_{Q_{REX^\uparrow}}$ is EXPSPACE-complete.
$EQ_{\text{REX}\uparrow}$ is EXPSPACE-Hard

$EQ_{\text{REX}\uparrow} = \{\langle Q, R \rangle \mid Q$ and $R$ are equivalent regular expressions with exponentiation\}$
Flashback: Undecidability By Checking TM Configs

\[ ALL_{CFG} = \{ \langle G \rangle | G \text{ is a CFG and } L(G) = \Sigma^* \} \]

Proof, by contradiction

- Assume \( ALL_{CFG} \) has a decider \( R \). Use it to create decider for \( A_{TM} \):

On input \(<M, w>\):

- Construct a PDA \( P \) that rejects sequences of \( M \) configs that accept \( w \)
- Convert \( P \) to a CFG \( G \)
- Give \( G \) to \( R \):
  - If \( R \) accepts, then \( M \) has no accepting config sequences for \( w \), so reject
  - If \( R \) rejects, then \( M \) has an accepting config sequence for \( w \), so accept

Any machine that can validate TM config sequences could be used to prove undecidability?
Flashback: Reducing every NP language to SAT

We know NP languages have a poly time NTM $M$! So reduce $M$ accepting config sequences to a satisfiable formula!
Reducing every \textbf{EXPSPACE} lang to $EQ_{\text{REX}^\uparrow}$

Some \textbf{EXPSPACE} lang $= \{w \mid w \text{ is ???}\}$

Need to reduce some $w$ to 2 equivalent regular expressions???

We know the language has an exp space decider!
Reducing every **EXPSPACE** lang to $EQ_{\text{REX}^\uparrow}$

$R_2$ equals $R_{\text{bad-start}} \cup R_{\text{bad-window}} \cup R_{\text{bad-reject}}$

Some **EXPSPACE** lang = \{w | w is ???\}

$R_1 = \Delta^*$, \quad \Delta = \Gamma \cup Q \cup \{\#\}$

$R_2 = \text{non-rejecting } M \text{ config seqs for } w$

$\Rightarrow$ if $M$ accepts $w$, there are no rejecting $M$ config seqs for $w$ so $R_1 = R_2$

$\Leftarrow$ if $M$ rejects $w$, there are rejecting $M$ config seqs for $w$ so $R_1 \neq R_2$

We know the language has an exp space decider!
Rejecting Config Sequences

A rejecting sequence of $M$ configs on $w$:
- **Starts** in start state $q_0$ with $w$ on the tape
- Each step must be valid according to $\delta$
- Ends in config with state $q_{\text{reject}}$

- $R_2$ generates config seqs that **don't satisfy** (at least 1 of) these
  $$R_{\text{bad-start}} \cup R_{\text{bad-window}} \cup R_{\text{bad-reject}}$$

**Important:**
- $R_2$ must be polynomial in length to have poly time reduction!
\[ R_{\text{bad-start}} = S_0 \cup S_1 \cup \cdots \cup S_n \cup S_b \cup S_# \]

\( R_{\text{bad-start}} \) = all strings not beginning with start config of \( M \) with \( w \)

- \( w = w_1, \ldots, w_n \) (length \( n \))
- \( S_0 = \Delta_{-q_0} \Delta^* \) = all strings that don’t start with \( q_0 \)
- \( S_i = \Delta^i \Delta_{-wi} \Delta^* \) = all strings whose \( i+1 \)th char isn’t \( w_i \)
  - These are all poly length (can be generated in poly time)
- \( S_b = \) all strings that don’t have a blank in pos \( n + 2 \) to \( 2^{n^k} \)
  - Could be exponential in length ...
  - ... unless we use exponentiation!

\[ S_b = \Delta^{n+1} (\Delta \cup \varepsilon)^{2(n^k) - n - 2} \Delta_{-\cup} \Delta^* \]

\( \Delta = \Gamma \cup Q \cup \{\#\} \)

\( \Delta_x = \) all chars in \( \Delta \) except for \( x \)

Exponential exponent ... takes \( \log(2^{n^k}) \) space = \( n^k \) space
Bad Reject

\[ R_{\text{bad-reject}} = \Delta^*-q_{\text{reject}} \]
Bad Window

- \text{bad}(abc, def) means window \(abc \rightarrow def\) not valid according to \(\delta\)
$R_2$ Total Length (Time)

- $R_{\text{bad-start}} = S_0 \cup S_1 \cup \cdots \cup S_n \cup S_b \cup S_\#$

- $O(n^k)$

- $R_{\text{bad-reject}} = \Delta^*_{\text{reject}}$

- $O(1)$

- $R_{\text{bad-window}} = \bigcup_{\text{bad}(abc,def)} \Delta^* abc \Delta^{(2(n^k) - 2)} \text{def} \Delta^*$

- $O(n^k)$

Exponential exponent ... takes \( \log(2^{n^k}) \) space = \( n^k \) space ... Can be generated in poly time

Total Time: $O(n^k)$
EXPSPACE-Completeness

**DEFINITION**

A language $B$ is *EXPSPACE-complete* if

1. $B \in \text{EXPSPACE}$, and
2. every $A$ in EXPSPACE is polynomial time reducible to $B$.

**THEOREM**

$EQ_{\text{REX}^\uparrow}$ is EXPSPACE-complete.
A Nonexistent Polynomial Time Algorithm

1. **Prove proper containment of two complexity classes,**
   - e.g., $P \subseteq \text{EXPSPACE}$

2. **Prove completeness of a language in the larger class,**
   - e.g., $EQ_{\text{REX}^\uparrow} \in \text{EXPSPACE}$ and $EQ_{\text{REX}^\uparrow}$ is EXPSPACE-hard

3. **Conclude** that the language cannot be in the smaller class
   - e.g., $EQ_{\text{REX}^\uparrow} \notin P$,
   - i.e., $EQ_{\text{REX}^\uparrow}$ has no poly time decider!
No Quiz 12/8

Fill out course evaluation