Lecture 23
CSE 431
Intro to Theory of
Computation

\[
\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n)) \subseteq \text{TIME}(2^{O(f(n))})
\]

\[\not\leq\]

\[L \subseteq \text{NL} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXP} \subseteq \text{NEXP}\]

\[\exists x, \forall x_2 \exists x_3 \ldots \exists x_n \phi(x_1 \ldots x_n)\]

Formula Game: played in formula \(\Phi\):

- \(E\) player wants to make \(\phi\) have value 1
- \(A\) player wants to make \(\phi\) have value 0.

Players take turn setting values of \(x\) vars:
- \(E\) player sets all \(\exists x_i\) variables
- \(A\) player sets all \(\forall x_i\) variables.

\(E\) player has a winning strategy \(\iff\) \(\Phi\) is true

\[\neg \Phi\] is true \(\iff\)

\[
\exists x_1 \forall x_2 \exists x_3 \ldots \exists x_n \phi(x_1 \ldots x_n)
\]

\[
\forall x_1 \exists x_2 \forall x_3 \ldots \forall x_n \neg \phi(x_1 \ldots x_n)
\]

\[
\exists x_1 \exists x_2 \forall x_3 \ldots \forall x_n \neg \phi(x_1 \ldots x_n)
\]

- \(E\) player for \(\neg \Phi\) has winning strategy
- \(A\) player for \(\Phi\) has winning strategy

Above is the basic idea for how to start showing that deciding guaranteed win in 2-player games are PSPACE-complete (\text{min} or \text{max} check, etc.).
Polynomial Hierarchy:

- sets of languages $\Sigma_k^0$, $\Pi_k^0$ for integer $k$

$\Sigma_k^0 = \{ A : A \leq_m^p \text{true formula in } \exists \forall \text{BF}_k \}$

where $\exists \forall \text{BF}_k$ = $k$ fully quantified Boolean formulas with

$\leq_k$ alternating blocks of quantifiers beginning with $\exists$

$\Pi_k^0 = \{ A : A \leq_m^p \text{true formula in } \forall \exists \text{BF}_k \}$

where $\forall \exists \text{BF}_k$ = $k$ fully quantified Boolean formulas with

$\leq_k$ alternating blocks of quantifiers beginning with $\forall$

$\Sigma_1^0 = NP$

$\Pi_1^0 = coNP$

Polynomial time hierarchy:

$PH = \bigcup_k \Sigma_k^0 = \bigcup_k \Pi_k^0$

There are other natural problems with practical importance
that are as hard as any problem in
classes like $\Sigma_2^0$

- eg. Minimum equivalent DNF (sum of products)
  $\langle F \rangle : F$ is a DNF formula that cannot be shortened
Then: If \( P = NP \) then \( PH = P \)

Proof: Idea: If \( P = NP \) then we can replace \( \exists \phi \forall \Psi \) by some \( \phi' \) of polynomial size, or \( \forall \phi \exists \Psi \) by some \( \phi' \) of polynomial size.

Note: Key difference between \( PH \) and \( PSPACE \) is that the # of attempts is bounded by a constant times being dependent on length of the input.

Logarithmic Space

Consider the following non-regular language

\[ A = \{ 0^n 1^m \mid n \geq 0 \} \]

The deciding \( A \):

On input \( x \)

Space:

Two counters:

- One length of input
- One \( \log(n) \) bits

Count # of 0's at start, begin shift 1.

Count # of 1's next.

If counts differ or there are more characters before 1, then reject

Else accept.

Let \( L = \text{SPACE}(\log n) \) \( \Rightarrow \) \( A \in L \)

\( NL = \text{NSPACE}(\log n) \)

\( L \subseteq NL \subseteq \text{TIME}(2^{O(\log n)}) = \mathbb{P} \)
Recall PATH = \( \exists \langle G, s, t \rangle : G \) is a directed graph with a path from \( s \) to \( t \).

**Proof Idea:** Guess and verify a path of length \( \leq \log n \) from \( s \) to \( t \) one vertex at a time.

- **Algorithm:**
  - Initialize: \( \text{count} \leftarrow 0 \), \( \text{curr} \leftarrow s \).
  - While \( \text{count} \leq \log n \) and \( \text{curr} \neq t \) do:
    - Guess next vertex (neighbor) \( v \) of \( \text{curr} \).
    - Check if \( (\text{curr}, v) \) is an edge.
    - If not then reject.
    - Else, \( \text{curr} \leftarrow v \).
    - If \( \text{curr} = t \) then accept; else reject.

**Question:** Is \( L = \text{NL} \)?

- \( L = \text{P?} \)
- \( \text{NL} = \text{P?} \)
- \( \text{NL} = \text{NP?} \)
- \( L = \text{NP?} \)

All of these are open. If \( P \neq \text{NP} \) then \( L \neq \text{NP} \), which may be easier to show.
To study these questions we need a finer notion of reducibility than \( \leq \) which allows polynomial slack.

For this we need a notion of log-space computable

function.

We modify our 2-tape space bounded notion of TM
to a 3-tape TM like this:

```
<table>
<thead>
<tr>
<th>input tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>(log n)</td>
</tr>
<tr>
<td>work tape</td>
</tr>
<tr>
<td>read-write</td>
</tr>
<tr>
<td>output tape</td>
</tr>
<tr>
<td>write only</td>
</tr>
</tbody>
</table>
```

**Def:** A function \( f \) is **logspace computable**

If it is computed by some logspace transducer

\( \Rightarrow f \) is polynomial-time computable (see below)

**Ex:** Consider the sort function on \( n \), \( O(\log n) \)-bit numbers

\[
\text{SORT}(x_1, \ldots, x_n) = \text{"x_1 \rightarrow x_n in sorted order"}
\]

\[\text{i.e., } x_{\sigma_1} \leq x_{\sigma_2} \leq \cdots \leq x_{\sigma_n}\] at a permutation

**Claim:** \( \text{SORT} \) is logspace computable.

Proof: \( \text{SORT} \) sorts like quicksort, merge sort, or stack space.

Selection sort only needs counters and one or two \( X \) stored.
Note: The sort is slow \( \geq n^2 \) time but uses only \( O(\log n) \) space.

Fact: For any sorting algorithm \( T \cdot S = O(n^2) \),

so this slowdown is unavoidable.

Logspace reducibility and completeness

**Defn** \( A \leq_L^m B \) iff \( A \leq_m B \) by a mapping reduction that is logspace computable.

**Theorem** If \( A \leq_L^m B \) then \( A \leq_M^0 B \).

Proof idea: \# of configurations of logspace transducer

\( \text{cut copying the output tape which doesn't} \)

\( \text{given low time TM model is \color{red} \leq_M^0 } \)

**Defn** \( B \) is \( \text{NL-hard} \) iff \( \forall A \in \text{NL}, A \leq_L^m B \).

**Defn** \( B \) is \( \text{NL-complete} \) iff

\( \begin{align*}
\circ & B \in \text{NL} \\
\circ & B \text{ is \text{NL-hard}}
\end{align*} \)

Pimp: \( \text{PATH} \) is \( \text{NL-complete} \).

Proof: We already showed that \( \text{PATH} \in \text{NL} \).
We just need to show NL-hardness

Let $A \in NL$

Claim: $A \leq^L_{tm} \text{PATH}$

Proof: $A \in NL$

i. Form $M$ deciding $A$ using $O(\log n)$ space.

$M$ accepts $x$

$\exists$ path from $s$ to $t$ in directed graph $G_{mix}$

So $A \leq^L_{tm} \text{PATH}$

$x \rightarrow f \rightarrow \langle G, s, t \rangle$

$G_{mix}$, $G_{CoC}$ accepts

$x \in A \Leftrightarrow$ path in $G_{mix}$ from $s$ to $t$

Output is polynomial since $2^{O(n)}$ nodes.
Each edge takes only $O(\log n)$ bits to write down and is easy to hit through edges in order.