This lecture began with Def. 4.1 in the Arora-Barak book [AB] (space-bounded computation, both deterministic and nondeterministic), the notion of “configuration graphs” (as defined in the text immediately preceding Claim 4.4 in [AB]), the fact that
\[ \text{DTIME}(S(n)) \subseteq \text{SPACE}(S(n)) \subseteq \text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))}), \]
and the notation PSPACE, NPSPACE, L, and NL (see Def. 4.5 in [AB]).

**Proof that PATH is in NL:**
A PATH instance is a triple \((G,s,t)\), where \(G\) is a directed graph, and \(\{s,t\} \subseteq V(G)\). The yes instances are those in which there is a path from \(s\) to \(t\) in \(G\). Note that, if \(V(G) = \{1,2,\ldots,n\}\), the instance \((G,s,t)\) is of length \(c \cdot n^2\), for some positive constant \(c\), assuming that we encode \(G\) as an \(n \times n\) matrix of bits in which the \((i,j)\)th bit is a 1 if and only if the arc \((i,j)\) is in \(A(G)\). (Note “arc” instead of “edge” and \(A(G)\) instead of \(E(G)\), in order to emphasize that \(G\) is a *directed* graph. PATH is a totally different, easier problem for undirected graphs.) So we seek a nondeterministic algorithm that decides PATH in space \(O(\log(c \cdot n^2)) = O(\log n)\). Here is one such algorithm:

\[
\text{PATH}(G,s,t) \\
\{ \\
\hspace{1em} i \leftarrow 0; \\
\hspace{1em} u \leftarrow s; \\
\hspace{1em} \text{WHILE}(i \leq n) \\
\hspace{2em} \{ \\
\hspace{3em} \text{IF } (u = t) \text{ THEN OUTPUT(ACCEPT) AND HALT; } \\
\hspace{3em} \text{GUESS } u' \in V(G); \\
\hspace{3em} \text{IF } ((u,u') \in A(G)) \text{ THEN } u \leftarrow u'; \\
\hspace{3em} i \leftarrow i + 1; \\
\hspace{2em}\} \\
\hspace{1em}\text{OUTPUT(REJECT) AND HALT; } \\
\}
\]

Things to notice about this algorithm:

- If there is a path from \(s\) to \(t\), then there must be one of length less than or equal to \(n\), because there are only \(n\) nodes in \(G\).

- We cannot simply guess a path of length at most \(n\) in one fell swoop, because that would require \(\Omega(n \log n)\) bits of workspace. Thus, we guess one node at a time and verify that all of the requisite arcs are there.
It is clear that the values of the variables \( i, u, \) and \( u' \) require \( O(\log n) \) workspace. Not as apparent, but still not hard, is that the bit on the input tape that tells us whether \((u, u') \in A(G)\) can be read in space \( O(\log n) \) using a counter.

In fact, PATH is NL-complete; we do not yet have the right notion of reduction to prove that, but we will get to it.

**Proof of Savitch’s Theorem:**

Let \( L \) be a language recognized in space \( O(s(n)) \) by nondeterministic Turing Machine \( W \), and let \( x \in \{0, 1\}^n \) be an input that may or may not be in \( L \). Consider the configuration graph \( G_{W,x} \). We will define a deterministic machine that, on input \( x \), decides whether there is a path from \( C^x_{\text{START}} \) to \( C^x_{\text{ACCEPT}} \), where these are the unique START and ACCEPT nodes in \( V(G_{W,x}) \). Recall that, if there is a path from \( C^x_{\text{START}} \) to \( C^x_{\text{ACCEPT}} \), there is one of length \( O(2^c \cdot s(n)) \), for some positive constant \( c \), i.e., that \(|V(G_{W,x})| = O(2^c \cdot s(n))\).

The deterministic algorithm that we provide actually solves the more general decision problem \( \text{REACH}(u, v, i) \), which is 1 if there exists a path from \( u \) to \( v \) in \( G_{W,x} \) of length at most \( 2^i \) and 0 if there is no such path. The algorithm is defined recursively.

For \( i = 0 \) (the base case of the recursion), the algorithm simply checks whether \( v \) is one of the two configurations that can be reached from \( u \) in one step, i.e., in one application of one of the transition functions \( \delta_0 \) and \( \delta_1 \) that define \( W \). (Think about why that can be done in space \( O(s(n)) \).)

For \( i > 0 \), we ask whether there is a configuration \( z \) such that \( \text{REACH}(u, z, i - 1) \) and \( \text{REACH}(z, v, i - 1) \) are both 1. The two crucial points are:

- We can cycle through all possible candidates for \( z \) and, having concluded that a particular \( z_j \) did not have the requisite property, reuse the space we just used for \( z_j \) to do the computation for \( z_{j+1} \).
- For a particular \( z \), we can compute \( \text{REACH}(u, z, i - 1) \) and then reuse the space to compute \( \text{REACH}(z, v, i - 1) \).

Let \( S_{M,i} \) be the space required to compute \( \text{REACH}(u, v, i) \) on a configuration graph \( G_{W,x} \) with \( M \) nodes. To decide whether there is exists a path from \( u \) to \( v \), we would use space at most \( S_{M,\log M} \). We have the recurrence relation

\[
S_{M,i} = S_{M,i-1} + O(\log M),
\]

because space \( S_{M,i-1} \) is needed for recursive calls, and space \( O(\log M) \) is needed to write down the “midpoint configuration” \( z \). Solving this recurrence relation gives us \( S_{M,\log M} = O((\log M)^2) \). For nondeterministic machine \( W \), we have \( M = O(2^c \cdot s(n)) \), and thus \( S_{M,\log M} = O((s(n))^2) \).

Note that Savitch’s Theorem implies that \( \text{PSPACE} = \text{NPSPACE} \).