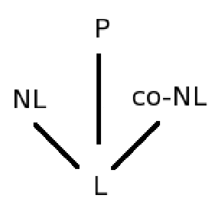
CS 221: Computational Complexity	Prof. Salil Vadhan
Lecture Notes 5	
February 8, 2010	Scribe: Brad Seiler

1 Agenda

- PATH is **NL**−Complete
- $\mathbf{NL} \subseteq \mathbf{P}$
- $\mathbf{NL} \subseteq \mathbf{L}^2$
- $\mathbf{NL} = \mathbf{co\text{-}NL}$
- TQBF is **PSPACE**−Complete

2 Recap

We learned about $\mathbf{NL} = \mathbf{NPSPACE}(\log n)$.
We have a hierarchy pictured below:



Also recall the PATH problem: PATH $= \{(G, s, t) \mid G$ directed graph with path from s to $t\}$

3 PATH is NL−Complete

Theorem 1 PATH *is* **NL**-*complete with respect to* $\leq_\ell$

Proof:

1. PATH $\in$ **NL** from last time.
2. Claim: PATH is **NL**−hard.
   Given $A \in \mathbf{NL}$, let M be a nondeterministic logspace Turing machine that recognizes A. We seek a logspace mapping reduction from A to PATH.

   Map $x \mapsto (G_{M,x}, s, t)$ where $G_{M,x}$ is the configuration graph of M on input x defined by:

- The vertices are the configurations of M on the input x (note that there are poly(n) vertices.)
- s is the starting configuration
- t is the accepting configuration, which is unique WLOG.
- Include the directed edge $(u, v) \iff$ one step of M can go from configuration u to configuration v.

$x \in A \iff M$ has an accepting computation on input $x \iff \exists$ a path from s to t in $G_{M,x}$. ■

4 NL ⊆ P

Corollary 2 **NL** ⊆ **P**

Proof: PATH ∈ **P** by BFS or DFS. ■

Corollary 3 *For space-constructible* $s(n) \geq \log n$, $\mathbf{NSPACE}(s(n)) \subseteq \bigcup_c \mathbf{DTIME}(c^{s(n)})$

Proof: We know that

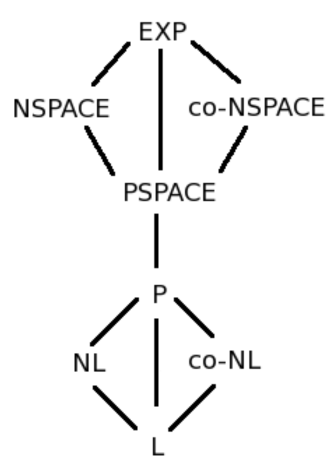
$$\mathbf{NSPACE}(\log n) \subseteq \bigcup_k \mathbf{DTIME}(n^k)$$

which implies by translation/padding that

$$\mathbf{NSPACE}(\log f(n)) \subseteq \bigcup_k \mathbf{DTIME}(f(n)^k)$$

for $f(n) \geq n, f(n)$ time constructible. Take $f(n) = 2^{s(n)}$ to complete the proof. ■

This updates the diagram. The translation principle used above gives us a similar hierarchy on **PSPACE**.



5 $\mathbf{NL} \subseteq \mathbf{L}^2$

Theorem 4 (Savitch's Theorem) $\text{PATH} \in \mathbf{L}^2 = \mathbf{SPACE}(\log^2(n))$

Proof: Define

$$\text{REACH}_G(u, v, i) = \begin{cases} 1 & \exists \text{ path from } u \text{ to } v \text{ of length} \leq i \\ 0 & \text{otherwise} \end{cases}$$

$$(G, s, t) \in \text{PATH} \iff \text{REACH}_G(s, t, n) = 1$$

where n is the number of vertices. We describe a recursive algorithm for $\text{REACH}_G(u, v, i)$.

Base cases: $i = 0$ accept iff $u = v$.
$i = 1$ accept iff $u = v$ or (u, v) is in G.

Recursion: For each vertex w, check $\text{REACH}_G(u, w, \lceil i/2 \rceil)$ and $\text{REACH}_G(w, v, \lceil i/2 \rceil)$. If both accept, halt and accept. If you run out vertices, reject.

The space used by $\text{REACH}_G(s, t, n)$ is bounded above by:

$$(\text{depth of recursion})(\text{space per level}) = (\log n)(\log n + O(1)) = O(\log^2 n)$$

We note that the *time* of the algorithm is fairly large. We can bound it by:

$$(\text{number of recursive calls per level})^{(\text{depth})}(\text{time per level}) = (n)^{\log n} \cdot O(n) \approx 2^{\log^2 n}.$$

■

Corollary 5 $\mathbf{NL} \subseteq \mathbf{L}^2$ *and additionally (by translation)* $\mathbf{NPSPACE} = \mathbf{PSPACE}$.

We know that PATH can be solved in polynomial time, and in can be solved in polylogarithmic space. However, it is open whether these two bounds can be achieved simultaneously:

Open Problem 6 $\text{PATH} \in \mathbf{TISP}(\text{poly}(n), \text{polylog}(n))$*?*

In contrast, for *undirected* graphs, such an algorithm is known. In fact, it was recently shown (2005) that logarithmic (rather than polylogarithmic) space is achievable.

Theorem 7 (Reingold's Theorem) $\text{UPATH} \in \mathbf{L} = \mathbf{TISP}(\text{poly}(n), \log(n))$.

6 $\mathbf{NL} = \mathbf{co\text{-}NL}$

NB: This construction seems obvious, but until it was demonstrated it was widely believed in the field that this equality was not true.

Theorem 8 (Immerman-Szelepcsényi Theorem) $\mathbf{NL} = \mathbf{co\text{-}NL}$.

Proof: It suffices to show that $\overline{\text{PATH}}$ is in **NL**.
To show this, prove that if $(G, s, t) \notin$ PATH, then there is a poly-length "certificate" of this fact that can be checked in logspace given *one-way* access to the certificate. (The certificate corresponds to the nondeterministic choices of the **NL** algorithm.)
Fix (G, s, t). Define:

$$C_i = \{v \mid v \text{ reachable from } s \text{ within } i \text{ steps}\}$$

Note that C_n is the whole connected component of s. Define $c_i = |C_i|$.
Given only $c_i = |C_i|$ for some i, we can certify:

1. $v \notin C_{i+1}$ for a given $v \in G$.

2. The value of $c_{i+1} = |C_{i+1}|$.

Proof:

1. To certify that $v \notin C_{i+1}$, provide

   $$(u_1, path_1), (u_2, path_2), \ldots, (u_k, path_k)$$

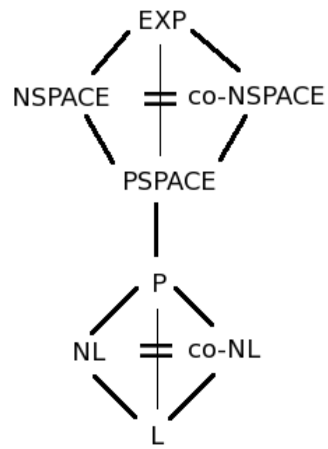
   where $k = c_i$, $path_j$ is a path of length $\leq i$ from s to u_j, and require that $u_1 < u_2 < \ldots < u_k$ relative to the original input order, and $\forall j, v \neq u_j$ and (u_j, v) is not an edge. This way, given that we believe that there are only k nodes reachable in i steps, this certifies all k nodes, so v cannot be one of them and is not immediately reachable from one of them. Ordering restriction means only the last vertex needs to be remembered to prevent repeats.

2. Go over all vertices in the input order. For each vertex, provide either a certificate that $v \in C_{i+1}$ (construct a path) or provide a certificate that $v \notin C_{i+1}$ by the above. The program need only check the certificate and increment the appropriate counter to assure that all vertices are accounted for.

So we repeatedly use (2) to get certificates for $c_1 = |C_1|$, then $c_2 = |C_2|$, then $c_3 = |C_3|$, and so on, until we have certified $c_{n-1} = |C_{n-1}|$, and then we use (1) to certify that $t \notin C_n$. (All these certificates can be concatenated together. The logspace verifier only needs to remember the value of c_i after verifying the i'th certificate.) ■

Corollary 9 *For space-constructible* $s(n) \geq \log n$, $\mathbf{NSPACE}(s(n)) = \mathbf{co}\text{-}\mathbf{NSPACE}(s(n))$.
$\mathbf{NSPACE}(n) =$ *context-sensitive languages is closed under complement.*

New diagram:



7 TQBF is PSPACE−Complete

On **PSPACE**:
Recall 3SAT: $\exists x \in \{0,1\}^n \;:\; \varphi(x) = 1$, where φ is a propositional formula in 3-**CNF**.
TQBF: $\exists x_1 \forall x_2 \exists x_3 \ldots Q_n x_n \;:\; \varphi(x_1, \ldots, x_n) = 1$, where φ is again in 3-**CNF**. Can be viewed as a game alternating between an existential and a universal player. Zermelo's Theorem says that one of the players has a winning strategy, and the question is which one? It can be shown that **PSPACE** $\leftrightarrow$ complexity of playing games optimally.

Theorem 10 TQBF *is* **PSPACE**−*complete.*

Proof: Next time. (Proof by `import __future__`.) ■