CIS 511: Theory of Computation	**Mar 23, 2017**

Lecture 17: Space Hierarchy

Professor Sampath Kannan — *Zach Schutzman*

NB: *These notes are from CIS511 at Penn. The course followed Michael Sipser's Introduction to the Theory of Computation (3ed) text.*

Finishing up $NL = Co\text{-}NL$

Claim 17.1 $NL = Co\text{-}NL$ *(continued from last time)*

Proof:

(Continued)

We were thinking about how to get the count of reachable vertices, c. Let c_i be the number of vertices reachable from s by paths of length at most i. We can, given c_i, compute c_{i+1}. We have $c_0 = 1$, so if we have a procedure to compute the increments, we can inductively find c_{n-1}. The idea will be to note that vertices reachable in at most $i+1$ steps have an edge from something reachable in at most i steps.

Similarly, denote s_i the set of vertices reachable from s in i or fewer steps.

The following procedure will compute c_{i+1}: For each vertex v_j, $v_j \in s_{i+1}$ if $v_j \in s_i$ or there exists a vertex u such that $u \in s_i$ and (u, v_j) is an edge in G. We will nondeterministically guess whether each vertex v is in s_i. To confirm it, we use a NL procedure for $PATH$ limited to paths of length i. We'll keep a counter for the number of things we find in s_i and the number of things in s_{i+1}. For each vertex, if we find it in s_i, then it is in s_{i+1}, so we increment both of our counters. Else, if there is an edge from something in s_i to it, then we increment our counter for s_{i+1}. Otherwise, we move on to the next vertex. After checking all vertices we now have the number of things in s_{i+1} on the correct branch of the $PATH$ algorithm, which is the one that correctly computed the number of things in s_i.

Inductively, we now know our c, as we do this procedure to find $c = c_{n-1}$. Since we have c, we have an NL algorithm to decide $\overline{PATH}$, hence $L = Co\text{-}NL$.

$\blacksquare$

We have $L \subseteq NL = Co\text{-}NL$. Whether L equals NL is unknown. The language $UPATH$, which is $PATH$ on undirected graphs is obviously in NL. It was proven about 10 years ago that $UPATH$ is actually in L.

We can say that Savitch's Theorem still applies. That is, $NSPACE(f(n)) \subseteq DSPACE(f(n)^2)$. Therefore, $NL \subseteq L^2$, deterministic log-squared space.

Space Hierarchy Theorem

We now are ready to show our first proper separation of time and space classes.

Definition 17.2 *A function $f(n)$ is 'nice' if it is* ***fully space-constructible****. That is, there is a Turing machine which given input, say 1^n, can compute $f(n)$ using no more than $f(n)$ space.*

Space-constructible functions include all the familiar ones, like monotonic polynomials, exponentials, logarithms, roots, etc.

Claim 17.3 (The Space Hierarchy Theorem) *Let $f(n)$ be a 'nice' function. Then there is a language L which can be decided by an $f(n)$-space deterministic Turing machine, but not by any deterministic Turing machine using $o(f(n))$ space. That is, for $f(n)$, there is a language requiring at least $O(f(n))$ space to decide by a deterministic Turing machine.*

Proof:

We prove this by constructing such a language. via diagonalization. Define a deterministic Turing machine M which uses $f(n)$ space and $L(M)$ cannot be decided by any Turing machine using $o(f(n))$ space. We'll think of M as being different from every machine which uses $o(f(n))$ space.

M on input $\langle x \rangle$ first checks if $\langle x \rangle$ is a description of a Turing machine. If not, reject. If it is, M should run $\langle x \rangle$ x. If x finishes in $f(n)$ space, then M outputs the opposite answer as x. M stops and rejects if $\langle x \rangle$ exceeds $f(n)$ space.

We also need to count the steps of x to make sure we don't exceed $2^{O(f(n))}$ time.

We also have an issue because $O(f(n))$ and $o(f(n))$ are asymptotic notions, so we may get something wrong for relatively small n.

We'll modify the behavior of M slightly. On any input, it checks to see if it is the description of a Turing machine, followed by 01^*. If not, reject. If so, simulate x on the whole input. As before, if x properly terminates, we output the complement of its answer. Now, we have that if x uses $g(n)$ space, we know there exists an n_0 such that for all $n > n_0$, $g(n) \leq f(n)$, because there is a sufficiently long input string such that x requires no more space than M.

$\blacksquare$