CIS 511: Theory of Computation **Mar 21, 2017**

Lecture 16: Space Complexity, Continued

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.*

Logarithmic Space

Recall, L and NL are the classes of problems recognizable in logarithmic space by a deterministic and non-deterministic Turing machine, respectively. These machines are modeled as having a read-only input tape and a read/write work tape. The space complexity is determined by the number of cells used on the work tape.

The language $PATH = \{\langle G, s, t\rangle | G \text{ is a digraph with a path from } s \text{ to } t\}$.

Claim 16.1 *$PATH$ is in NL.*

Proof: *Nondeterministically guess a sequence of neighboring vertices, for at most n steps. This needs only $O(log(n))$ space to track the current vertex and a counter.* ■

It turns out, $PATH$ is one of the 'hardest' languages in NL, that is, it is NL-Complete.

We need to show that $PATH$ is in NL (done) and that any NL language can be reduced to it. What is our notion of reduction? Before, we had polynomial time reductions, but $NL \subset P$, so a polynomial time (and hence polynomial space) could be powerful enough to solve our language. We therefore restrict our reductions to the class L, that is, a log-space reduction.

Definition 16.2 *A **log-space transducer** is a machine with a read-only input, a read-write work tape, and a write-only output tape. The work tape has size $O(log(n))$ and the write tape can have polynomial length in n.*

Definition 16.3 *A language A **log-space reduces** to a language B (written $A \preceq_m^L B$) if there is a log-space transducer computing a function f such that $x \in A \iff f(x) \in B$.*

Claim 16.4 *$PATH$ is NL-Complete.* **Proof:**

Let A be any language in NL. We show that there exists a log-space transducer reducing A to $PATH$.

We have some input $w \in \Sigma^*$ to A and we want to construct a log-space transducer to convert it into a $\langle G, s, t\rangle$. The high level idea is to construct a graph with vertices corresponding to configurations of a machine recognizing A, s and t corresponding with initial and accept configurations, and edges for configurations which are one computation step apart.

A configuration of the A machine is specified by the location of the head in the input tape, the current state, and the contents of the work tape. This requires only $O(log(n))$ bits to specify, as we need $O(log(n))$ to keep

track of the position in the input, plus $O(log(n))$) to represent the work tape, plus some constant to track the current state. The number of possible configurations is $2^{O(log(n))}$, which is polynomial in n.

The graph G has one node for each possible configuration, the node s will be the initial configuration, and the node t will be the accept configuration in canonical form. There will be an edge (c_i, c_j) if the machine for A can go from configuration c_i to C_j in one step. There may be multiple edges from each node, corresponding to nondeterministic choice. There exists an s-t path in this graph if and only if $w \in A$.

We now only need to show that a log-space transducer can compute this reduction. The transducer will first construct the vertices of G, then the edges.

First, the transducer knows the length of each configuration, so it uses its $O(log(n))$ space to iterate through the possible configurations and if it's a valid configuration c, writes c to its output tape. After this, all of the nodes in the graph are written to the output. Then, for the edges, check each pair of configurations to determine if the second is reachable from the first in 1 step.

This procedure only requires $O(log(n))$ work space and produces the appropriate G, s, t, hence is is a log-space reduction and the proof that $PATH$ is NL-Complete.

■

Note that any function computable by a log-space transducer can be computed in polynomial time, a transducer can never repeat a configuration (else it would not be a decider), and there are only polynomially many configurations.

Claim 16.5 *$NL \subseteq P$*

Proof: *We know that $PATH \in P$ (we can do breadth-first search). Therefore, given any $A \in NL$ and input w, we can, in polynomial time, log-space reduce A to an instance of $PATH$ with input $f(w)$ and use a polynomial time algorithm for $PATH$. This gives a polynomial time algorithm for A.* ■

The Class *Co-NL*

Definition 16.6 *A language A is in Co-NL if its complement is in NL.*

From before, a language B is in NL if there is a nondeterministic log-space Turing machine M such that given w in B, $M(w)$ accepts on some path. If $w \notin B$, all computation paths reject. Co-NL is the opposite. If $w \in C \in$ Co-NL, then all paths accept. If $w \notin C$, there exists at least one rejecting path.

Claim 16.7 *$NL =$ Co-NL*

Proof:

We have $PATH$ as an NL-Complete. Let's define the language $\overline{PATH} = \{\langle G, s, t\rangle | G$ is a digraph with no s-t path$\}$. The same reduction as before shows that $\overline{PATH}$ is Co-NL-Complete. We prove this theorem by showing $\overline{PATH} \in NL$. This implies $NL \subseteq$ Co-NL, which by symmetry of complements gives us $NL =$ Co-NL.

We want a nondeterministic log-space Turing machine M which, given $\langle G, s, t\rangle$ and a number c equal to the number of nodes reachable from s, accepts if there is no path from s to t in G. This machine will guess whether there is or is not a path from s to v_1. If there is one, using a log-space $PATH$ subroutine, continue guessing paths from s to v_2. If every path in this side rejects, we know there is no path from s to v_1, so we continue down the no branch. When we find a path from s to a vertex, we increment a counter. We continue checking for each vertex except t. If there is no s-t path, the counter will reach c before we finish exhausting

vertices.

We now need to think about how to get 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.

■