CS 6810: Theory of Computing	Fall 2023

Lecture 8: September 14, 2023

Lecturer: Eshan Chattopadhyay — *Scribe: Omkar Bhalerao*

1 NL = coNL

In the previous lecture, we introduced the following theorem:

Theorem 1.1 (Immerman, Szelepscensyi)**.** $\mathsf{NL} = \mathsf{coNL}$.

We continue by proving this theorem. To do so, we first define the language $\overline{\text{PATH}}$:

Definition 1.2. $\overline{\text{PATH}} = \{\langle G, s, t\rangle : \textit{there is no s-t path in digraph } G = (V, E)\}$.

We now outline our proof statement along with some corollaries.

Theorem 1.3. *There exists an $O(\log n)$ space nondeterministic algorithm for $\overline{\text{PATH}}$.*

Corollary 1.4. $\mathsf{NL} = \mathsf{coNL}$.

Corollary 1.5. *For all "nice" $S(n) > \log n$, $\mathsf{NSPACE}(S(n)) = \mathsf{coNSPACE}(S(n))$.*

As an aside, note that one implication of Corollary 1.5 is that $\mathsf{NPSPACE} = \mathsf{coNPSPACE}$; however, we have already shown this, since we know that $\mathsf{NPSPACE} = \mathsf{PSPACE}$ and that PSPACE is closed under its complement. In addition, we contrast this theorem to Savitch's theorem from the previous lecture: that $\mathsf{NSPACE}(S(n)) \subseteq \mathsf{DSPACE}(S(n)^2)$. In other words, there's no overhead when going from NSPACE to $\mathsf{coNSPACE}$, unlike when going from NSPACE to DSPACE.

To begin the proof, we have the following definition:

Definition 1.6. $C_i = \{v \in V : \textit{there exists a path of length at most } i \textit{ from } s \textit{ to } v\}$.

As such, $\overline{\text{PATH}}$ is equivalent to asking whether $t \notin C_n$.

Claim 1.7. *There exists an $O(\log n)$ nondeterministic algorithm that can decide $[v \in C_i]$ for all i.*

The proof of this is similar to the proof given in the previous lecture for deciding PATH, but we will outline it below:

Proof. We will describe such an algorithm below:

On the work tape, have a section representing a counter, a section representing the current guess, and a section representing the next guess. Begin by writing s to the current guess section. Then, at each iteration, nondeterministically choose a neighbor u of s, write it to the next guess section, and increment the counter. Then, write the next guess to the current guess section, and repeat. If we see node v at any point in this process, accept; otherwise, after the counter reaches i, reject.

Since at most $O(\log n)$ bits are needed to store the counter and the labels of all the nodes, this algorithm indeed only uses $O(\log n)$ space. □

Claim 1.8. *Given $|C_{i-1}| = r$, there exists a nondeterministic $O(\log n)$ space algorithm that can decide $[v \notin C_i]$ for all i.*

Proof. We describe such an algorithm below:

First, our NDTM will nondeterministically guess a set $T \subseteq V$ such that $|T| = r$. Then, the machine will use claim 1 on all nodes in its guess to check whether it is correct. Though in general representing the set T could take space larger than $O(\log n)$, it's possible to sequentially go through each element of T to ensure that no more than $O(\log n)$ space in total is used. Once a correct guess T is found, our NDTM will simply sequentially go through all the nodes in T and check if v is connected to each of the nodes. If v is connected to any one of the nodes, our algorithm will reject. Otherwise, it will accept. □

Claim 1.9. *Given $|C_{i-1}| = r_{i-1}$, there exists a nondeterministic $O(\log n)$ space algorithm that either rejects, or outputs the correct size of C_i, and the said output happens for at least one execution path.*

Proof. Again, we describe an algorithm.

For every node in the neighbors of C_{i-1}, our algorithm will nondeterministically guess whether $v \in C_i$ or $v \notin C_i$, and then verifies those guesses using Claim 1 and Claim 2. It will also maintain a counter that starts at r_{i-1}. If the algorithm makes a correct guess, then either 1 or 0 is added to the counter. Since the Claim 1 and Claim 2 algorithms are guaranteed to be correct for at least one path of execution, it's therefore the case that there's at least one path of execution for which the Claim 3 algorithm is also correct. In addition, we will use the same sequential strategies as previously to ensure that no more than $O(\log n)$ total space is used. □

Now that we have these three claims, our overall algorithm to solve PATH is as follows: first, we set $|c_0| = 1$. From this, we compute $|c_1|, \ldots, |c_n|$ sequentially using Claim 3 (implicitly using Claims 1 and 2 in the process). Lastly, we answer $[t \notin C_n]$ using Claim 2. Thus, we've shown that $\overline{\text{PATH}} \in \mathsf{NL}$ and thus that $\mathsf{NL} = \mathsf{coNL}$.

2 PSPACE-completeness

We now switch topics to PSPACE-completeness.

Definition 2.1 (PSPACE-completeness)**.** *A language L is* PSPACE*-complete if $L \in$ PSPACE and for all $L' \in$ PSPACE, $L' \leq_P L$, where $\leq_P$ means that L' is polynomial-time Karp reducible to L.*

To aid us in talking about PSPACE-completeness, we will now introduce a canonical language to represent PSPACE:

Definition 2.2 (True Quantified Boolean Formulas (TQBF))**.** TQBF *is the language of true quantified Boolean formulas; in other words, formulas of the following form that evaluate to true:*

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)$$

where each Q_i is a quantifier (i.e. $\forall$ or $\exists$), and ϕ is a Boolean formula of variables $x_1, \ldots, x_n$.

We will show that TQBF is indeed PSPACE-complete.

Claim 2.3. TQBF $\in$ PSPACE.

Proof. We describe a polynomial-space algorithm for deciding TQBF.

First, our TM will brute-force over all variable assignments. It iterates over all quantifiers in the process, and in doing so, will ensure that for all $\forall$-quantified variables, both assignments result in a true formula, and for all $\exists$-quantified variables, at least one assignment results in a true formula. If this is true for all variables, our TM accepts; otherwise, it rejects.

This TM uses polynomial space because space can be reused across the different recursive calls. This is captured in the following recurrence, where n is the number of quantifiers, m is the size of the formula ϕ, and $S(n, m)$ is the space complexity at that level:

$$S(n, m) = S(n-1, m) + O(\text{poly}(n, m))$$

Again, since we reuse space across each recursive call, this recurrence is correct. Thus, the total space complexity used is still $O(\text{poly}(n, m))$, proving that TQBF $\in$ PSPACE. □

Claim 2.4. *For all $L \in$ PSPACE, $L \leq_P$ TQBF.*

Proof. For the sake of time, we omit the full proof; this will be given in the next lecture. We will, however, go through the setup steps for it.

First, note that if $L \in$ PSPACE, then there exists a TM M that uses cn^c space for some constant c and computes L. We will denote cn^c by $S(n)$ from here on out.

We define $G_{M,x}$ to be the configuration graph of M on input x. Then, by definition, $x \in L$ iff there exists a path from c_{start} to c_{accept} in $G_{M,x}$, where c_{start} is the starting configuration and c_{accept} is the accepting configuration (WLOG assume that there's exactly one accepting configuration).

We now define a formula $\phi_i(A, B) = 1$ for two configurations A, B if there exists a path from A to B in $G_{M,x}$ of length at most 2^i. Precisely, we take the Boolean variables that encode the configuration state of A and B and define a formula for which the above fact is true. Let ℓ be the total number of variables in A and B; then, the formula is one on 2ℓ variables, where $\ell \in O(S(n))$ by definition.

Clearly, we have that $\phi_0(A, B) = 1$ iff $(A, B) \in E(G_{M,x})$, where $E(G)$ represents the edge set of the graph G. As such, ϕ_0 can be encoded by the transition function of M. In the more general case where $i \neq 0$, we can simply write $\phi_i(A, B) = (\exists C.\phi_{i-1}(A, C) \wedge \phi_{i-1}(C, B))$. However, we're not quite done yet, because the formula $\phi_i(A, B)$ could be potentially exponentially sized in A and B. There is, however, a trick to ensure that $\phi_i(A, B)$ is polynomially sized, and we will cover that in the next lecture. □