CS 6810: Theory of Computing **Spring 2026**

Lecture 9: February 24, 2026

Lecturer: Eshan Chattopadhyay *Scribe: Kevin Chen*

1 NL = co-NL

We will prove the following theorem promised from last lecture.

Theorem 1.1 (Immerman–Szelepcsényi (1987))**.** *NL = co-NL.*

To prove this, we consider the language

$$\mathsf{PATH} = \{(G, s, t) : G \text{ is a directed graph such that there is a path from } s \text{ to } t\}.$$

Last lecture, we showed PATH is NL-complete. In particular, this implies its complement

$$\overline{\mathsf{PATH}} = \{(G, s, t) : G \text{ is a directed graph such that } t \text{ is not reachable from } s\}.$$

is co-NL-complete. We will show $\overline{\mathsf{PATH}}$ is also in NL. This will be enough to show NL = co-NL.

Remark 1.2. *As an aside, a closely-related problem is*

$$\mathsf{UPATH} = \{(G, s, t) : G \text{ is an undirected graph such that there is a path from } s \text{ to } t\}.$$

In 2005, Omer Reingold showed $\mathsf{UPATH} \in \mathsf{L}$.[1] *This suggests the fact that* $\overline{\mathsf{PATH}} \in \mathsf{NL}$ *shouldn't be too surprising.*

We now describe an NL algorithm to determine whether or not $(G, s, t) \in \overline{\mathsf{PATH}}$, where $G = (V, E)$. Define

$$C_i = \{\text{all nodes reachble from } s \text{ using at most } i \text{ steps}\}.$$

Say there are $n = |V|$. Then the decision problem for determining whether $(G, s, t) \in \overline{\mathsf{PATH}}$ is equivalent to the problem for determining whether $t \notin C_{n-1}$.

Claim 1.3. *For each* i *and* $v \in V$*, there is an* NL *algorithm to decide* $v \in C_i$*.*

Proof. The proof is similar to showing PATH ∈ NL from last lecture. Have an NDTM store three variables: a counter, the current node, and a guess for the next node. Initialize the counter to 0 and the current node v. The NDTM non-deterministically guesses a next node u that is a neighbor of s, puts u in the next node variable, and increments the counter. The NDTM moves u to the current node and repeats this process. If v is seen at any point in the process, accept. Otherwise, reject after the counter reaches i. □

Claim 1.4. *Given that* $|C_{i-1}| = r$*, there is an* NL *algorithm that accepts an input* v *if and only if* $v \notin C_i$*.*

[1] See the paper for this result here.

Proof. We go through subsets $T \subseteq V$ with $|T| = r$ to guess the set C_{i-1}. It is worth noting that the naive way of guessing T is not be feasible using small space. To be able to do so using small space, sequence the nodes from $1, 2, \ldots, n$. Have the NDTM represent a counter for number of elements found in T so far (initialized to 0 at the start of the algorithm) and remember the last node added u (initialized to the empty string at the start of the algorithm). Then guess the node u' of C_{i-1} that appears sequentially after u, the last node added in T. Use Claim 1.3 to check $u' \in C_{i-1}$. If $u' \in C_{i-1}$, then replace u' with u and increment the counter by 1. Otherwise, continue to the sequentially next node after u', and repeat this process.

After each time the NDTM updates the last node u added in T, check whether u is a neighbor of v. If it is, then reject. If at any point in the algorithm the counter for size of T reaches r, then you accept. This whole process takes $O(\log n)$ space. □

Claim 1.5. *Given $|C_{i-1}| = r_{i-1}$, there is an NL algorithm that either rejects or outputs $|C_i|$. Moreover, there is at least one execution path for the output $|C_i|$.*

Proof. We describe an NDTM as follows: Start a counter at r_{i-1}. Sequentially process all nodes in G. For each node, non-deterministically guess either $v \in C_i$ or $v \notin C_i$. The branch that guessed $v \in C_i$ uses Claim 1.3 to verify its guess, while the second branch that guessed $v \notin C_i$ uses Claim 1.4 to verify its guess. If there is no rejection and the branch that guessed $v \in C_i$ is correct, add 1 to the counter. By Claims 1.3 and 1.4, there is an execution path amongst one of the branches $v \in C_i$ and $v \notin C_i$ that will verify the correct guess, while all other execution paths reject. This means this NDTM has at least one execution path that outputs $|C_i|$, while all other execution paths reject. Moreover, Claims 1.3 and 1.4 show that this NDTM takes $O(\log n)$ space to run and is thus in NL. □

Going back to finding an NL algorithm to determine whether $(G, s, t) \in \overline{\mathsf{PATH}}$, we know $|C_0| = 1$. From Claim 1.5, we can sequentially compute $|C_1|, \ldots, |C_{n-1}|$. We can then use Claim 1.4 to check $t \notin C_n$. This whole process takes a logarithmic amount of space, so $\overline{\mathsf{PATH}} \in \mathsf{NL}$.

2 Boolean circuits

Definition 2.1. *A **Boolean circuit** is a directed acyclic graph with three types of nodes:*

1. ***Leaves** or **input nodes** labeled as input variables (e.g., $x_1, \ldots, x_n$). The in-degree or fan-in of each input node is 0.*
2. ***Internal nodes** labeled as logical gates (e.g., $\wedge, \vee, \neg$). Unless otherwise specified, the in-degree of each $\wedge$ and $\vee$ gate is 2.*
3. ***Output nodes**. The out-degree of each output node is 0.*

A Boolean circuit naturally computes a Boolean function $f\colon \{0,1\}^n \to \{0,1\}^m$, where n is the number of input nodes and m is the number of output nodes. For example, the Boolean function corresponding to the Boolean circuit in Figure 1 is given by $f(x_1, x_2) = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2) = x_1 \oplus x_2$.

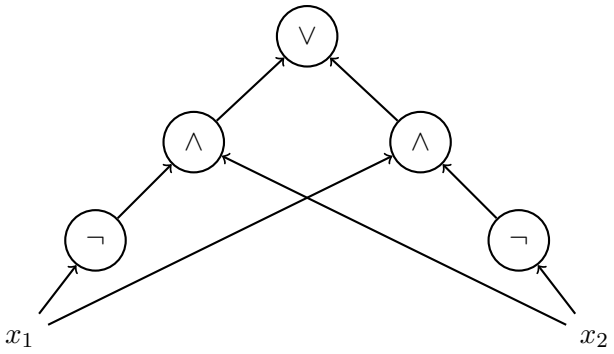

Figure 1: Example of a Boolean circuit computing the XOR of two bits.

An important difference between Boolean circuits and Turing machines is that Boolean circuits can only accept inputs of a fixed size. In other words, a circuit is essentially computing a function $\{0,1\}^n \to \{0,1\}$, where as a TM computes a function $\{0,1\}^* \to \{0,1\}$. So it doesn't make sense for a circuit to compute a language $L \subseteq \{0,1\}^*$.

To remedy this, we work with a family of circuits. We say a family of circuits $\{C_n : n \geq 0\}$, where C_n is a circuit with n inputs, computes a language $L \subseteq \{0,1\}^*$ if for each $n \geq 0$, the circuit C_n computes $L_n \coloneqq L \cap \{0,1\}^n$. We call this a **non-uniform model of computation**.

There are two major **complexity measures**:

1. **Size**: Number of edges in circuit.
2. **Depth**: Longest path from an input node to an output node.

We say a language L is computed by an $S(n)$**-sized circuit family** if there exists $\{C_n : n \geq 0\}$ that computes n and the size of C_n is at most $S(n)$ for all n.