CS 6810: Theory of Computing **Fall 2021**

Lecture 8: Sep 21, 2021

Lecturer: Eshan Chattopadhyay *Scribe: Laurel Newman*

1 Recap

Recall the following complexity classes defined in last class: $L = \text{DSPACE}(\log n)$, $NL = \text{NSPACE}(\log n)$. It is conjectured that L = NL. We believe this problem to be more tractable than the question of P vs NP.

One of the main focus in this class is to prove the following theorem.

Theorem 1.1 (Immerman & Szelepscènyi). *NL = co-NL.*

Recall that PATH (defined below) is NL-complete (under logspace reductions). Thus, $\overline{PATH}$ is co-NL complete. We will show an NL algorithm for $\overline{PATH}$ to prove the above theorem.

Definition 1.2. $\text{PATH} = \{\langle G, s, t\rangle : G$ *a directed graph,* s, t *nodes in* G *s.t.* $\exists$ *a path from* s *to* $t\}$

2 $\overline{\textbf{PATH}}$ is in NL

Definition 2.1. $\overline{\text{PATH}} = \{\langle G, s, t\rangle : G$ *a directed graph,* s, t *nodes in* G *s.t.* $\nexists$ *a path from* s *to* $t\}$

The following is the main result of this section.

Lemma 2.2. *There is an* $O(\log n)$*-space NDTM for* $\overline{\text{PATH}}$*.*

We now note here that if we let $c_i = \{v$ is reachable from s in $\leq i$ steps$\}$, then proving Lemma 2.2 is equivalent to showing membership of t in c_n.

Claim 2.3. *There is an NDTM that on input* i, v *uses* $O(\log n)$*-space and accepts if* $v \in c_i$*.*

Proof Idea. This follows from the proof for PATH $\in$ NL, where we simply let $t = v$, and maintain that our counter of how many steps we take must halt after i steps. □

Claim 2.4. *For any* $i \geq 1$ *and any vertex* v*, with* $|c_{i-1}|$ *given (as input), there exists an* $O(\log n)$*-space NDTM which accepts if* $v \notin c_i$*.*

Proof. Sequentially go through $j \in \{1, \ldots, n\}$, checking membership in c_i:

Initialize $w = 0$, $j = 1$, where w is our counter for $|c_{i-1}|$ and j is our counter for which vertex we're considering. Note these both take $\log n$ bits to store.

We reuse space to non-deterministically check if $j \in c_{i-1}$ (which we can do by Claim 2.3, taking $O(\log n)$ space).

- if yes: check if v is a neighbor of j or if $v = j$.
  - if yes: reject.
  - else: let $w \leftarrow w + 1$
    and let $j \leftarrow j + 1$ if $j < n$.

- else: let $j \leftarrow j+1$ if $j < n$.

Once $j = n$, we are finished with checking all vertices, so we check that $w = |c_{i-1}|$:

- if yes: accept.
- else: reject.

□

Claim 2.5. *There exists an $O(\log n)$ space NDTM, which given $|c_{i-1}|$ and any computation path as input, either rejects or outputs $|c_i|$, and does not trivially always reject.*

Before we prove this final claim, we note how these claims enable us to prove Lemma 2.2:

Proof Sketch for Lemma 2.2. We have that $c_0 = \{s\}$, so $|c_0| = 1$ is known. From here, by repeatedly using the NDTM guaranteed by Claim 2.5 we can find $|c_1|, \ldots, |c_{n-1}|$, which by Claim 2.4 can then be used to determine if $t \in c_n$. □

Proof Sketch for Claim 2.5. We construct a counter for checking vertices $1, \ldots, n$ and a counter for $|c_i|$. We check for each vertex if it is in c_i (by Claim 2.3) and increment $|c_i|$ if it is. Again, we use sequential non-determinism. □

The above proof can be used to prove the more general result.

Theorem 2.6. *For any space-constructible $S(n)$,*

$$\text{NSPACE}(S(n)) = co\text{-NSPACE}(S(n)).$$

3 Boolean Circuits

Definition 3.1. *Boolean circuits are a non-uniform model of computation. That is, for each input length, we use a different algorithm. Thus, we will talk about* circuit families $C = \{C_n\}_{n\in\mathbb{N}}$ *computing functions or languages.*

3.1 Basic definitions

Circuits have a layer of 'input nodes,' which feed into layers of 'internal nodes,' forming a directed acyclic graph. These interal nodes all feed via some path eventually into an 'output node' or nodes. (We might consider for instance, the problem of sorting a list of numbers as a problem where we might require multiple output nodes.)

In circuits, in-degree is also called 'fan-in' and out-degree is also called 'fan-out.' Edges are also called 'wires.'

Nodes are also called 'gates.'

- input nodes: labelled with variables, have in-degree 0.
- output node(s): have out-degree 0.
- internal nodes: labelled with logical gates ($\wedge, \vee, \neg$). $\wedge$ and $\vee$ gates have no restrictions, while $\neg$ gates have in/out-degree 1.

The size of a circuit $s(C)$ is the number of wires in C. The depth of a circuit $d(C)$ is the length of the longest input node to output node path.

Definition 3.2. *A language* $L \in \{0,1\}^*$ *is computed by* $\{C_n\}_{n\in\mathbb{N}}$ *if* $\forall n \in \mathbb{N}$, $x \in \{0,1\}^n$, *then* $C_n(x) = L(x)$.

The size of $\{C_n\}_{n\in\mathbb{N}}$ is $T(n)$ (where $T : \mathbb{N} \to \mathbb{N}$) if $s(C_n) = T(n)$ for all $n \in \mathbb{N}$.

Definition 3.3. P/POLY *is the set of languages computed by poly-sized circuits.*

A major question is to understand the power of P/POLY? For instance, what is the relation between P and P/POLY. We discuss these in the next class.