**CS 6810: Theory of Computing** **Fall 2023**

Lecture 7: Sep 11, 2023

*Lecturer: Eshan Chattopadhyay* *Scribe: Sanjit Basker*

# 1 PSPACE and NPSPACE, continued

Recall the definition of a configuration of a Turing Machine: it is the contents of the input/output tapes and all of the work tapes, the locations of the working heads, and the state of the Turing Machine. For a Turing Machine with space complexity $s(n)$, there are $2^{O(s(n))}$ configurations reachable from the starting configuration on an input of size $n$, where the big-O notation hides a constant that depends on the machine but not the size of the input.

These configurations form a graph $G_{M,x}$. For a nondeterministic TM $M$, each node will have outdegree at most 2.

Also recall that we can assume WLOG that $M$ has a single halting configuration: if not, we can add some "cleanup" phases in which the machine clears the contents of its working tapes and writes a single 1 on the output tape.

Consider the language $PATH = \{\langle G, s, t\rangle : \text{there exists a path from } s \text{ to } t\}$.

**Theorem 1.1** (Savitch)**.** *There exists an $O(\log^2 n)$ space algorithm for $PATH$.*

*Proof.* Let $T(u, v, i)$ be 1 if there is a (directed) path of length $\leq 2^i$ from $u$ to $v$. The base cases occur when $i = 0$, in which case you simply check whether the edge appears in $G$. For the recursive case, we use the relation

$$T(u, v, i) = \bigvee_{w \in V} (T(u, w, i-1) \wedge T(w, v, i-1)),$$

since if there is a path of length $\leq 2^i$ from $u$ to $v$, it must contain some node $w$ halfway through the path. There are $O(n^2 \log n)$ subproblems, because you can pick a pair of vertices $(u, v)$ and $\log n$ choices for $i$. We want the answer to $T(u, v, \log n)$.

The implementation requires some care to ensure that it uses $O(\log^2 n)$ space: starting with the call $T(u, v, n)$ we first write down $u$ and $v$ in $O(\log n)$ space. Then, we iterate through the possibilities for $w$: for each possibility, we solve the first subproblem, store its result, solve the second subproblem, and then take the and of both results. If we re-use space for the two subproblems, our space used obeys the recurrence $s(i) \leq c \log(n) + s(i-1) \implies s(n) \in O(\log^2 n)$.
The analogy is that our "stack depth" is $O(\log n)$ and the size of each stack frame is $O(\log n)$. □

**Corollary 1.2.** *For space-constructible $s(n)$, $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{DSPACE}(s^2(n))$.*

**Corollary 1.3.** *$\mathsf{PSPACE} = \mathsf{NPSPACE}$.*

# 2 More space complexity

Now, consider the set $\mathsf{NL}$. This is the set of languages $L$ for which there exists a non-deterministic TM which, on inputs of size $n$ takes $O(\log n)$ space and decides $L$. The class $\mathsf{coNL}$ is defined similarly.

Note that $\mathsf{NL} \subseteq P$, since we can apply the above construction to get a configuration graph with $2^{c \log n} \leq n^c$ vertices and traverse the graph from the starting configuration.

The definition of a reduction in $NL$ requires some care, because if we allowed any poly-time Karp reduction, then the reducer would have the full power of a poly-time TM (e.g. it could solve the problem and then "reduce" to a fixed yes- or no-instance).

**Definition 2.1.** *The function $f : \{0,1\}^* \to \{0,1\}^*$ is implicitly computable in $s(n)$ space if there exists a TM $M$ that runs in $s(n)$ space, and which on input $\langle x, i \rangle$ outputs $(f(x))_i$, the $i$th bit of $f(x)$.*

*We also place the requirement that for some $c$, $|f(x)| \leq c|x|^c$.*

**Claim 2.2.** *If $f$ and $g$ are implicitly computable in $O(\log n)$ space, then $f = f \circ g$ is also implicitly computable in $O(\log n)$ space.*

*Proof.* The idea is to run $M_f$ but compute the bits of $g(x)$ on the fly.

To be more concrete, suppose we are implicitly computing the $i$th bit of $(f \circ g)(x)$. We simulate $M_f$, except that instead of giving it the input $g(x)$ (the entirety of which could be longer than we can fit on one of our tapes), for every step of $M_f$ we first check the location of $M_f$'s input read head and compute the value at that location.

The additional bookkeeping (the location of the input head) can be done with $O(\log n)$ bits, and the rest of the simulation is constant overhead. □

**Definition 2.3.** *Language $A$ is logspace reducible to language $B$ if there is a function $f : \{0,1\}^* \to \{0,1\}^*$ that is implicitly computable in $O(\log n)$ space, and $x \in A$ iff $f(x) \in B$.*

**Definition 2.4.** *A language $L$ is $\mathsf{NL}$-complete if for every language $B \in \mathsf{NL}$, $B$ is logspace reducible to $L$.*

**Lemma 2.5.** *$PATH$ is $\mathsf{NL}$-complete.*

*Proof.* First, we show that $PATH \in \mathsf{NL}$, so we should define a nondeterministic TM that computes it in $O(\log n)$ space. This machine will have two "variables," a current vertex and a counter. While the counter is smaller than the number of vertices in the input graph, it increments the counter and then nondeterministically chooses one of the vertices adjacent to the current one (concretely, it could pad the number of vertices up to the next power of 2, nondeterministically select a random number in the range, and halt and reject if the vertex is nonexistent or not adjacent). If it ever reaches $t$, it halts and accepts.

The counter and current vertex variables each take $O(\log n)$ space. Some additional bookkeeping may be needed in order to search the adjacency list of the current vertex, but this is also $O(\log n)$ space.

Now, we show that $PATH$ is $\mathsf{NL}$-complete. If $L$ is in $\mathsf{NL}$, we know that there's some log-space NDTM $M$ which uses at most $c \log n$ tape cells to decide $L$. We will again use the configuration graph in the reduction to $PATH$, but with the added restriction that our reduction must be implicitly computable in $O(\log n)$ space.

The input to $PATH$ is $\langle G, s, t \rangle$ where $G$ is the configuration graph, $s$ is the starting configuration of $M$, and $t$ is the ending configuration of $M$. The configuration graph will contain at most $n^c$ vertices. Assuming some encoding from the configurations of $M$ into the integers, we can use the adjacency matrix representation of the graph: when asked for the bit of the input at an index $i < n^{2c}$, we'll parse it as corrsponding to two configurations $c_1$ and $c_2$ of $M$ processing $x$, we would use the transition function of $M$ to determine if $c_2$ is one of the two configurations that can follow $c_1$.

The input also contains another $2cn \log n$ bits, hard-coded to be the identifiers of the starting and accepting configurations of $M$ processing $x$ (and possibly some other information at the beginning describing the size of the graph), which are implicitly computable in $O(\log n)$ space. □

**Corollary 2.6.** *$\overline{PATH}$ is co-*NL *complete.*

We'll prove the following next class:

**Theorem 2.7.** *NL = coNL.*

We also state an interesting theorem proven recently.

**Theorem 2.8** (Immerman, Szelepscensyi)**.** NL = co-NL