Lecture 12

Gidon Rosalki

2025-06-22

This content will not appear in the exam this year!

We are going to discuss the classes NL, coNL, applying Savitch's theorem on NL (and sublinear classes), and Immerman's theorem NL = coNL, and use and proof.

1 Sublinear space classes

Theorem 1 (Savitch)**.** *For every function $f(n)$ computable in space $O(f(n))$ it holds that*

$$NSpace(f(n)) \subseteq Space\left(f^2(n)\right)$$

We showed this for every $n \leq f(n)$, and will now show this for $\log(n) \leq f(n) \leq n$

Proof . Let there be a language $A \in \text{NSPACE}(f(n))$, and let there be a NTM M, that recognises A, in space $O(f(n))$. For a word w, we will look at the configurations graph $G_{M,w}$, which has a node for each configuration, an edge for each possible transition of δ of M, and two special configurations c_0, the starting configuration, and c_{acc}, the accepting configuration. $w \in A$ **if and only if** There exists a run that accepts $M(w)$ **if and only if** $(G_{M,w}, c_0, c_{\text{acc}}) \in \text{PATH}$. That is to say, there is a directed path from c_0, to c_{acc} in $G_{M,w}$.
We have built a DTM M' that **decides** if $(G_{M,w}, c_0, c_{\text{acc}}) \in \text{PATH}$, while only using a relatively small amount of space $O\left(f^2(n)\right)$. The full proof is located in a previous lecture.
M' passes over every middle node in the graph (all the nodes). If there exists a path for this 3-tuple of nodes, then it returns true. If not, it returns false, and moves on to the next tuple. Correctness is trivial. The space compleity is the sum of the complexity of 3 nodes, times the depth of the recursion. This is the log of number of nodes in the graph, so overall we get $O\left(\log^2(\text{num nodes in graph})\right)$. The number of the nodes in the graph is

$$|Q| \cdot n \cdot O(f(n)) \cdot |\Gamma|^{O(f(n))}$$

So, since $\log(\#\text{conf}) = O(f(n))$, in total the machine M' uses $O\left(f^2(n)\right)$ space.

□

What is the number of configurations for a machine that is NL? (Where $f(n) = O(\log(n))$

$$\#\text{conf} = |Q| \cdot (n+1) \cdot O(\log(n)) \cdot |\Gamma|^{O(\log(n))}$$

Which is comprised of the internal states, the head location on the input tape, the location of the head on the work tape, and the content of the work tape.

Note: We will note that in the above count, we did not count the possibilities of the input tape (which are $|\Sigma|^n$), since w is given, which is to say does not vary between different nodes of the graph $G_{M,w}$, we did not count it.

So in total we get

$$\log(\#\text{conf}) = O(\log(n))$$

2 Immerman's theorem

Theorem 2.

$$NL = coNL$$

Which is to say

$$NSpace(\log(n)) = coNSpace(\log(n))$$

Or

$$A \in NL \Leftrightarrow \overline{A} \in NL$$

We have already seen that NPSpace = PSpace. Reminder: The fact that PSpace $\subseteq$ NPSpace is trivial. The other direction, NPSpace $\subseteq$ PSpace can be shown through Savitch's theorem. Let there be $A \in \text{NPSpace} = \bigcup_{k=1}^{\infty} \text{NSpace}\left(n^k\right)$. That is to say, there exists k such that $A \in \text{NSpace}\left(n^k\right)$. From Savitch, it holds that $A \in \text{Space}\left(\left(n^k\right)^2\right) = \text{Space}\left(n^{2k}\right)$, and so $A \in \bigcup_{k=1}^{\infty} \text{Space}\left(n^k\right) = \text{PSpace}$.

Let us move on to our actual theorem, NPSPACE = coNSPACE, which is to say $A \in \text{NPSPACE} \Leftrightarrow \overline{A} \in \text{NPSPACE}$. Proof: Let there be $A \in \text{NPSPACE}$. We will show that $\overline{A} \in \text{NPSPACE}$. $A \in \text{NPSPACE}$, however NPSPACE = PSPACE, so $A \in \text{PSPACE}$. This is to say that there exists a DTM that decides A, in polynomial space, and so it is also true that $\overline{A} \in \text{PSPACE}$, since the same DTM that decides A can decide $\overline{A}$ by swapping its accepting and rejecting states, with the same space usage, and so it also holds that $\overline{A} \in \text{NPSPACE}$.

Can we use the same method to show that NL = coNL? No. Using Savitch in order to transition from non-deterministic to deterministic increases the space usage to its square. For some space that is polynomial, that matters not, but for NL, this gives us

$$B \in NL \implies B \in \text{SPACE}\ \left(\log^2(n)\right)$$

It s not clear if from here we can return to coNL, or NL, it is possible to achieve a DTM that decides $\overline{B}$, but it will require $O\left(\log^2(n)\right)$.

Let us try a different tack. Given $A \in \text{coNL}$, and so $\overline{A} \in \text{NL}$, let there be a machine NL M for $\overline{A}$. Can we create an NL machine for A by swapping the states $q_{\text{acc}}, q_{\text{rej}}$? We cannot. This is becuse the machine NL for $\overline{A}$ ensures that if $w \in \overline{A}$, there is an accepting run, and if $w \notin A$, every run is rejected. Swapping the states $q_{\text{acc}}, q_{\text{rej}}$ will give a machine that if $w \in \overline{A}$, there is a run that rejects, and if $w \notin \overline{A}$ every run is accepted, and so if $w \notin A$, there is a run that accepts, and if $w \in A$, every run will accept. As we can see, this does not define the machine that we want.

Concrete example: Let us use

$$\text{PATH} = \{(G, s, t) : \text{There is a directed path from } s \text{ to } t \text{ in } G\}$$

We saw that PATH $\in$ NL-Complete. Furthermore, PATH $\in$ NL. We saw an NL machine for PATH, that beings from s, and non deterministically chooses a path, and returns q_{acc} **if and only if** the path finishes at t. Swapping the states $q_{\text{acc}}, q_{\text{rej}}$ creates a machine that does not ensure correctness for $\overline{\text{PATH}}$. For (G, s, t), if the machines returns q_{acc}, it is because there is not a path from s to t in G, which is to say that $(G, s, t) \notin \overline{\text{PATH}}$. However, given that the machine found a single path, there is a path from s to t in G, which is to say that the machine was meant to return q_{rej}.

Theorem 3.

$$NL = coNL \Leftrightarrow \overline{PATH} \in NL$$

Proof . $NL = coNL \implies \overline{\text{PATH}} \in NL$. If $NL = coNL$, then since PATH $\in$ NL, we can get that $\overline{\text{PATH}} \in NL$

$\overline{\text{PATH}} \in NL \implies NL = coNL$. Reminder, PATH $\in$ NL-Hard, let $A \in coNL$, then $\overline{A} \in \text{NL}$. So,

$$\overline{A} \leq_L \text{PATH}$$

and therefore

$$A \leq_L \overline{\text{PATH}}$$

through the same reduction. Since it is given that

$$\overline{\text{PATH}} \in \text{NL}$$

there

$$A \in \text{NL}$$

by the reduction theorem of language in NL. In conclusion

$$\text{coNL} \subseteq \text{NL}$$

However, we wanted equivalence, not containment. Let there be $B \in \text{NL}$, therefore

$$B \in \text{NL}$$
$$\implies \overline{B} \in coNL$$
$$\text{According to the containment} \implies \overline{B} \in NL$$
$$\implies NL \subseteq coNL$$
$$\implies NL = coNL$$

In conclusion, $\overline{PATH} \in NL$ is a necessary requirement, and sufficient to show that

$$NL = coNL$$

Therefore, in order to prove Immerman's theorem, we want to show an NL machine for

$$\overline{\text{PATH}} = \{(G, s, t) : \text{ In the directed graph } G, \text{ there is no path from } s \text{ to } t\}$$

□

2.1 Examples for use of Immerman

Example 1.

$$A = \{(G, u, v) : w, v \textit{ are not in the same strongly connected component in the directed graph } G\}$$

Reminder: 2 nodes u, v are in the same strongly connected component ***if and only if*** *there is a directed path from u to v, and also from v to u in G.*
Reminder: If $C, D \in NL$, then so too are $C \cap D$ and $C \cup D$.

Solution. We need to show that $A \in$ NL:
$\overline{\text{PATH}} \in$ NL from Immerman, and

$$\begin{aligned} A \in (G, y, v) &\Leftrightarrow \text{There is no path from } u \text{ to } v \text{ or there is no path from } v \text{ to } u \\ &\Leftrightarrow (G, u, v) \in \overline{\text{PATH}} \vee (G, v, u) \in \overline{\text{PATH}} \end{aligned}$$

That is to say, the union of the two language in NL, which is to say that A is in NL. □

Example 2.

$$\textit{2-Con} = \{G : G \textit{ is a directed graph, and has at least 2 strongly connected components}\}$$

Solution. We need to show that 2-con $\in$ NL:
We will create an NL-machine for 2-Con: M will run as follows: For every pair of nodes $u, v \in V(G)$, it will run the machine of $\overline{\text{PATH}}$ on (G, u, w). If the machine returns q_{acc}, then M will also return such, and stop. Otherwise, M will move onto the next pair u, v. If M has tried all the pairs u, v, then it will return q_{rej}.

Correctness: If $G \in$ 2-Con, then there exists a pair of nodes u, v that for not in the same strongly connected component. W.l.o.g. there is no path from u to v, and so when running the machine $\overline{\text{PATH}}$ on (G, u, v), there is a run that returns q_{acc}, therefore there is a run of M that returns q_{acc}.
If $G \notin$ 2-con, which is to say all of G is a single strongly connected component, then for every pair of nodes u, v every run of the machine $\overline{\text{PATH}}$ on (G, u, v) returns q_{rej}, and therefore M always returns q_{rej}.

Space complexity: A variable for u, and for v: $O(\log(n))$, the space for running $\overline{\text{PATH}}$: $O(\log(n))$, in total $O(\log(n))$. □

2.2 Proof of Immerman

We need to show that $\overline{\text{PATH}} \in$ NL.

2.2.1 General idea

We will look at NL machines that carry out computations, and ensure that during the run:

1. The function that we compute $f(n)$ is written in the output, and the internal state is q_{acc}
2. Or we are not sure what is on the output, and the internal state is q_{rej}
3. For every input there is at least 1 run which finishes in q_{acc}

How does this help? Consider the following example.
Let us assume that there is a machine NL, as described just above (three requirements) that for the input (G, s), computes c, the number of nodes that can be reached from s in G. We will call this machine M'. With M', we may build an NL machine for $\overline{\text{PATH}}$.

$$\overline{\text{PATH}} = \{(G, s, t) : \text{ In the directed graph } G, \text{ there is no path from } s \text{ to } t\}$$

We will run M', once on (G, s), and compute c, and a second time on $(G \setminus \{t\}, s)$, and compute c'. That is to say, the second time we run M', and compute how many nodes can be reached from s, when we remove t from G. If $c = c'$, then $(G, s, t) \in \overline{\text{PATH}}$, and if $c \neq c'$, the $(G, s, t) \in$ PATH. M will therefore return q_{acc} **if and only if** $c = c'$.

We will use c_k to be the number of nodes that one can reach from s in G by k steps or less. So

- $c_0 = 1$, in a path of length ≤ 0, we can only reach s
- $c_1 =$ the number of neighbours of $s + 1$. In a path of length ≤ 1, we can reach s, and all its immediate neighbours

We will show a method to compute c_{k+1} by using (G, s, c_k). The machine M'' will run as follows: Beginning from $c_0 = 1$, and compute in a loop until c_n where n is the number of edges in G. Similarly, we will run M'' in order to compute c'_0, until c'_n for G', where the graph is like G, but without the node t. The machine M will compute c_n and c'_n, and answer q_{acc} **if and only if** $c_n = c'_n$.
NOTE: The machine M'' may transfer to q_{rej} at every stage of a run, but if $(G, s, t) \in \overline{\text{PATH}}$, then there is a run such

that M'' does not return q_{rej} at all. That is to say, correctly computes the values c_k. We will focus on the task of computing c_{k+1} from c_k.
For every node $v_1, \ldots, v_n$ we want to know with **certainty** if we can reach it via a path of length less than equal to $k+1$. If so, we will increase the counter c_{k+1} by 1, and if not, then we will not. At the end of the process, we will get a justified value for c_{k+1}.

How will will know with certainty if it is possible to reach v_1, within $k+1$ steps? It is possible **if and only if** there is a node v_i which can be reached in k or less steps, and from there there is an edge to v_1. Therefore, for every node $v_i \in V(G)$, it is the candidate to be the v_i we want to reach from s within k steps, and we will compute the path from s to v_i in length $\leq k$. If we succeed to reach v_i, then we will increase the counter D_k by 1, and check if there is an edge (v_1, v_i). After we pass over all the edges for v_i, we will check if $D_k = c_k$.

- If so - We will note that we have passed over all the nodes of distance $\leq k$
- If not - we have missed at least 1

Therefore, if so, and we have not managed to arrive to v_1, then we can be certain that we cannot reach it. However, if not, which is to say $D_k \neq c_k$, we will stop and return q_{rej}. Similarly to v_1, we will do this for every node $v_2, \ldots, v_n$, of course making sure to zero D_k every time.

Space usage: For D_k, c_k, c_{k+1} we need $O(\log(n))$, and for the guessed path from s, we need $O(\log(n))$, because we guess a path step by step, and count how many steps there were. For c, c', we need $O(\log(n))$. In total, $O(\log(n))$.
Correctness: For every k, if we did not return q_{rej}, we get that c_{k+1} was computed correctly from c_k. Therefore, both c, and c' are correctly computed (in a run where we do not return q_{rej}), which is to say M correctly solves $\overline{PATH}$