# Computational Complexity Theory : Lecture 8

Instructor : Chandan Saha
Scribes : Sanal S Prasad and Ishan Banerjee

September 1, 2015

## 1 NL-completeness

We would like to define **NL**-completeness. For this, we have to define the appropriate kind of reduction. Our definition of reduction should be guided by the question $\mathbf{L} \stackrel{?}{=} \mathbf{NL}$.

**First Attempt** : We should be looking at some kind of "log-space reduction". We would like to define $\mathtt{L}_1 \leq_l \mathtt{L}_2$. Suppose we define as follows : We say that $\mathtt{L}_1$ log-space reduces to $\mathtt{L}_2$ if there is a log-space computable function $f$ such that $x \in \mathtt{L}_1 \Leftrightarrow f(x) \in \mathtt{L}_2$

With this definition we would like to show that if $\mathtt{L}_1 \leq_l \mathtt{L}_2$ and $\mathtt{L}_2 \in \mathbf{L}$, $L_1 \in \mathbf{L}$. Let's try to prove this.

**Attempted proof:** Suppose $x \in \{0,1\}^*$. $\mathtt{L}_2 \in \mathbf{L}$ so it has a TM M which runs in log space. We would like to compute $f(x)$ give it to M and then find out if $f(x) \in \mathtt{L}_2$ using M. However $f(x)$ can take space that is not logarithmic in $|x|$. In that case just writing down $f(x)$ will take too much space. So this is not going to work.

### 1.1 Log space reduction

First a definition.

**Definition 1.1** (Implicit log space computable)**.** A function $f : \{0,1\}^* \to \{0,1\}^*$ is called implicit log space computable if

1. $|f(x)|$ is polynomial in $|x|$ , i.e $|f(x)| \leq |x|^c$ for some $c \in \mathbb{R}$.
2. $\mathtt{L}_1 := \{(x,i) | f(x)_i = 1\}$ is in $\mathbf{L}$ ($f(x)_i$ is the $i$th bit of $f(x)$).
3. $\mathtt{L}_2 := \{(x,i) | i \leq |f(x)|\}$ is in $\mathbf{L}$.

In some sense, we want each bit to be computable in log space. The last condition is to ensure that we don't ask for the $i$th bit of $f(x)$ when $f(x)$ has less than $i$ bits.

**Definition 1.2.** A language $\mathtt{L}_1$ is log space reducible to $\mathtt{L}_2$ if there is an implicit log space computable function $f$ such that $x \in \mathtt{L}_1 \iff f(x) \in \mathtt{L}_2$.

**Lemma 1.1.** *If* $\mathtt{L}_1 \leq_l \mathtt{L}_2$ *and* $\mathtt{L}_2 \in \mathbf{L}$*, then* $\mathtt{L}_1 \in \mathbf{L}$*.*

**Proof:** Let $M$ be a log space machine that decides $\mathtt{L}_2$. Construct a new machine $M'$ which takes $y$ as input and simulates $M$ on input $f(y)$. Whenever $M$ asks for the $i$th bit of $f(y)$, we compute it in log space and give it to $M$. Since $M$ itself runs in $\log(|f(y)|)$ space and $f(y)$ is at most polynomial in $|y|$ the machine runs in $O(\log|y|)$ space. $\square$

**Lemma 1.2.** *If* $\mathtt{L}_1 \leq_l \mathtt{L}_2$ *and* $\mathtt{L}_2 \leq_l \mathtt{L}_3$*, then* $\mathtt{L}_1 \leq_l \mathtt{L}_3$.

**Proof:** Suppose $f$ is the reduction function from $\mathtt{L}_1$ to $\mathtt{L}_2$ and $g$ is the reduction function from $\mathtt{L}_2$ to $\mathtt{L}_3$. It is clear that $g \circ f$ is a reduction from $\mathtt{L}_1$ to $\mathtt{L}_3$. We must show that it is implicit log space computable. Let $M$ be a machine that computes the $i$th bit of $g$. Just simulate $M$ on input $f(x)$ computing the $i$th bit of $f(x)$ (in log space) when it is asked for. Again, this is clearly $O(\log|x|)$ for similar reasons as in the previous question. $\square$

**Definition 1.3** (**NL** completeness)**.** A language $\mathtt{L} \in \mathbf{NL}$ is **NL** complete if for every $\mathtt{L}' \in \mathbf{NL}$, $\mathtt{L}' \leq_l \mathtt{L}$.

Let's consider the following language: $\mathtt{PATH} = \{(G, s, t) | G$ is a digraph and $t$ is reachable from $s$ in $G\}$

**Theorem 1.3.** $\mathtt{PATH}$ *is* **NL** *complete.*

**Proof:** (1) $\mathtt{PATH}$ is in **NL**.
We give the sequence of vertices connecting $s$ and $t$ as the read once certificate. The verifier checks if every two contiguous vertices are indeed connected.This can be done in log space.

(2) Any language $\mathtt{L} \in \mathbf{NL}$ reduces to $\mathtt{PATH}$ i.e. $\mathtt{L} \leq_l \mathtt{PATH}$.
Let $\mathtt{L} \in \mathbf{NL}$. Suppose $M$ decides $\mathtt{L}$ in log space. The reduction is as follows $x \mapsto (G_{M,x}, c_{start}, c_{accept})$, where $G_{M,x}$ is the configuration graph of $M$ on input $x$ and $c_{start}$ and $c_{accept}$ are the start and accept states respectively. Since $M$ takes log space $|G_{M,x}|$ is at most $2^{O(\log|x|)}$ which is polynomial in $|x|$. To compute the $i$th bit, we check if the $i$th bit lies in $G_{M,x}$ $c_{start}$ or $c_{accept}$. If it lies in $c_{start}$ or $c_{accept}$ it does not depend on $|x|$ and it can be found in constant space. If it is in the description of vertices it can be found in log space as the size of any vertex $c$ is logarithmic in the size of the input as the machine $M$ takes space logarithmic. If it is in the description of edges, we need to check whether two vertices are adjacent or not. We use the two transition functions to determine if there is an edge between the two vertices or not. This can again be done in log space. $\square$

Consider the language $\overline{\mathtt{PATH}} = \mathtt{PATH}^c$. It is clearly in $\mathbf{co} - \mathbf{NL}$. In fact, we have the following:

**Lemma 1.4.** $\overline{\mathtt{PATH}}$ *is* $\mathbf{co} - \mathbf{NL}$ *complete i.e.* $\overline{\mathtt{PATH}}$ *is in* $\mathbf{co} - \mathbf{NL}$ *and every language* $\mathtt{L} \in \mathbf{co} - \mathbf{NL}$ *is log space reducible to* $\overline{\mathtt{PATH}}$.

**Proof:** $\overline{\mathtt{PATH}}$ is in $\mathbf{co}-\mathbf{NL}$ as $\mathtt{PATH}$ is in **NL**. Let $\mathtt{L}$ be a language in $\mathbf{co}-\mathbf{NL}$. Then $\overline{\mathtt{L}}$ is in **NL** and hence log space reduces to $\mathtt{PATH}$. Simply take the same reduction function, $f$. We know that $x \in \overline{\mathtt{L}} \iff f(x) \in \mathtt{PATH}$. This implies that $x \in \mathtt{L} \iff f(x) \in \overline{\mathtt{PATH}}$. This establishes the theorem. $\square$

We have the following theorem

**Theorem 1.5** (Immerman-Szelepsceny). $\overline{\texttt{PATH}} \in \mathbf{NL}$. *This implies* $\mathbf{NL} = \mathbf{co-NL}$.

**Proof:** Given $(G, s, t)$, we need to certify that $t$ is not reachable from $s$. To do this, we first identify the vertices of G with numbers $1, 2, \cdots, n$. Let $N_j :=$ no. of vertices reachable from $s$ via a path of length at most $j$. And let $V_j :=$ set of all vertices reachable from s in path of length at most j. Note that $N_j$ is small i.e. storing $N_j$ requires only $O(\log n)$ bits.

Our proof will be by induction. Assume that the verifier knows $N_n$, then it can verify that $t$ is not reachable from $s$ using the following certificate: For each $v_i$ that is reachable from $s$, we provide $v_i$ and a path from $s$ to $v_i$ of length less than or equal to $n$. We give $v_i$'s in ascending order. The verifier can go through the certificate and verify that the $v_i$'s are reachable, and that they are given in ascending order. The verifier does this by first writing down $s$ and then $v_i$. Suppose our path is $v_{i_1}, \ldots, v_{i_n}$. Then the verifier will check for each $k$, whether $v_{i_k}$ is adjacent to $v_{i_{k+1}}$. At any one point, the verifier only stores the value of $v_i$ and the value of $v_{i_k}$ and the value of $v_{i_{k+1}}$. This requires only $O(\log n)$ space and is read-once. The verifier also checks that $t$ is not equal to $v_i$ and that $v_i < v_{i+1}$.

Suppose the verifier knows the value of $N_{k-1}$ and wishes to verify the value of $N_k$. For this, we give the certificate as follows : for all vertices, $v_i$, we provide a certificate that $v_i$ either is in $V_k$ or not in $V_k$. If $v_i$ is in $V_k$, we give a path from $s$ to $v_i$ of length at most $k$. This too requires only $O(\log n)$ space to verify and is read-once. If it is not in $V_k$, we use the following certificate: All the vertices of $V_{k-1}$ in ascending order, with their certificates (path). The verifier merely checks that the vertex is not adjacent to any of these vertices. This also takes $O(\log n)$ space and is read-once. This follows by a similar argument as above.

Our overall certificate will consist of a concatenation of certificates for $N_0, N_1 \ldots N_n$ and then a certificate that $t$ is not reachable. The verifier will store the value of $N_k$ and use it to verify the value of $N_{k+1}$ using the read-once certificate for $N_{k+1}$. It then reuses the space used by $N_k$ and stores $N_{k+1}$ and repeats the process. At each point of time, we are only using the amount of space necessary for verifying the value of $N_{k+1}$, given the value of $N_k$ and the certificate for $N_{k+1}$. But we already know that this takes $O(\log n)$ space. Therefore, at any point of time the verifier needs to store only $O(\log n)$ space. The certificate is also polynomial in size. To see this, observe that the certificate for each $N_k$ consists of at most $n$ vertices and the certificate for each of these takes at most $n^2$ space. $\square$