E0 224: Computational Complexity Theory
CSA, IISc Bangalore

Instructor: Chandan Saha *Scribe: Prerak Dhoot*

Lecture 11: September 10, 2014

1 Log Space reduction

Consider two languages *L1,L2*.
If we use the traditional polynomial time reduction to reduce *L1* to *L2* and given that *L2* $\in$ L(log-space) we cannot infer that *L1* $\in$ L. For if a machine M is using poly-time to reduce an instance of *L1* into an instance of *L2*, it might mean that M is using poly-space(to store the intermediate results).Hence, it is necessary that M be a log-space machine to ensure that the conversion process is carried out in log-space.

Note: It may be possible that x is reduced to $f(x)$ where $x \in L1$ and $f(x) \in L2$ but $|f(x)|$ uses poly-space. In such a case, it would not be possible to write the complete $f(x)$ on the output tape of the log-space machine. To overcome this problem we use a technique called ***"Implicit Log-space Reduction"***

1.1 Definition: Implicitly Log-space Computable

A function $f : \{0,1\}^* \to \{0,1\}^*$ is implicitly log-space computable if for every $x \in \{0,1\}^*$, $|f(x)| = \mathcal{O}(|x|^c)$ for a constant 'c' and the following two languages are in L.

1) $B_f = \{ \langle x, i\rangle : f(x)_i = 1 \}$
2) $B_{f'} = \{ \langle x, i\rangle : i \leq |f(x)| \}$

As mentioned above $|f(x)|$ may be using poly-space, but we do not construct it all at once. We only generate the needful bit using a log-space machine and perform computation on it. We thereby ensure that the complete process is carried out in log-space.

We finally define log-space reduction in the following manner.

1.2 Definition: Log-space Reduction

A language B reduces in log-space to a language C (denoted by $B \leq_l C$) if there is an implicitly log-space computable function f such that $x \in B$ iff $f(x) \in C$.

Lemma 1.1 *If $B \leq_l C$ and $C \leq_l D$ then $B \leq_l D$*

Proof: Let f and g be the two implicitly log-space computable functions such that $x \in B$ iff $f(x) \in C$ and $y \in C$ iff $g(y) \in D$.

We infer that $x \in B \leftrightarrow f(x) \in C \leftrightarrow g(f(x)) \in D$

Goal: To show that $g(f(x))$ is implicitly log-space computable.

Let M_f be a log-space machine that reduces $\langle x, i \rangle$ to $f(x)_i$

$$\langle x, i \rangle \xrightarrow[(M_f)]{\text{log-space}} f(x)_i \qquad (i \leq |f(x)|)$$

Let M_g be a log-space machine that reduces $\langle y, j \rangle$ to $g(y)_j$

$$\langle y, j \rangle \xrightarrow[(M_g)]{\text{log-space}} g(y)_j \qquad (j \leq |g(y)|)$$

Consider a function $h = g \circ f$ i.e $h(x) = g(f(x))$
To show that $B \leq_l D$, we would like to design a log-space machine M_h to perform the reduction from an instance of B into an instance of D using implicit log space reduction.

$$\langle x, k \rangle \xrightarrow[(M_h)]{\text{log-space}} h(x)_k \,/\, g(f(x))_k \qquad (k \leq |g(f(x))|)$$

Here is how the log-space machine M_h works. Given an input $\langle x, k \rangle$ to M_h, M_h starts simulating M_g on $f(x)$. For this, M_g needs access to $f(x)$. However we do not have access to $f(x)$(since we are never computing $f(x)$ all at once) and therefore we pretend that $f(x)$ is there on a fictitious tape

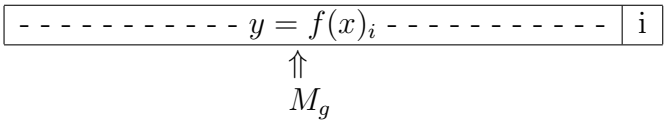

Fig: Working of M_h

The moment M_h needs to access a bit (say the i'th bit of $f(x)$), it postpones M_g , saves the contents of M_g on it's own tape and starts M_f with input $\langle x, i \rangle$ to generate output $y = f(x)_i$. M_h then feeds the output of M_f as an input to M_g to give $g(f(x))_k$, which gives us the required reduction from B to D in log-space. Note that M_h always saves the position 'i' on it's tape since $f(x)$ is on a fictitious tape. Storing the position 'i' takes $log|f(x)|$ space.♠

Corollary 1.1 *Suppose* $B \leq_l C$ *and* $C \in \mathsf{L}$ *then* $B \in \mathsf{L}$

Let M_c be the log-space machine which decides C.

To show $B \in \mathsf{L}$ we would like to have a function f to be an implicitly log-space computable function such that $x \in B$ iff $f(x) \in C$.

Consider M_f to be a log-space machine which reduces $\langle x, i \rangle$ to $f(x)_i$

$$\langle x, i \rangle \xrightarrow[(M_f)]{\text{log-space}} f(x)_i \qquad (i \leq |f(x)|)$$

Whenever M_c needs some $f(x)_i$ bit , we run M_f over $\langle x, i \rangle$. We then compute $f(x)_i$ on M_c using log-space. Thus, we can decide if $x \in B$ in log-space, first by using the implicit log-space reduction to reduce $\langle x, i \rangle$ into $f(x)_i$ and then by computing $f(x)_i$ on log-space machine M_c. Thus $B \in L$.♠

2 Class NL-complete:

Class NL-complete is defined analogous to class NP-complete.

Definition: A language $C \in \mathsf{NL}$ is NL-complete if every language $B \in \mathsf{NL}$ reduces in log-space to C.

2.1 PATH is in NL (Non-deterministic log-space)

$PATH = \{\langle G, s, t \rangle : G$ is a directed graph and vertex 't' is reachable from vertex 's' in G $\}$.

Proof: Consider a non-deterministic machine N which stores a vertex-index(i.e a sequential number assigned to a vertex). Let the vertex-index be initialized to the index of vertex 's'. Let V be the set of vertices in G. N selects a vertex non-deterministically, and checks if it is adjacent to vertex 's' from the adjacency matrix. If the guessed vertex is not adjacent, we terminate such paths otherwise N updates the vertex-index to the index of the guessed vertex. We recursively follow the same procedure. Also, everytime we find a new vertex in a path ,we decrement a counter initially set to $|V|$. Once the counter reaches zero, we halt the machine, since a path can have a maximum length of $|V|$. Finally if the vertex-index of 't' has been encountered, it implies vertex 't' is reachable from vertex 's' in G. Note that every path will be using a space equivalent to storing a vertex-index i.e $|log(V)|$. Also the counter used to count the number of vertices encountered in the path takes $|log(V)|$ space. Hence, by using the above NL-machine we can decide if an instance $\langle G, s, t \rangle \in$ PATH. Thus PATH is in NL. ♠

Theorem 2.1 *PATH is* NL*-complete*

$PATH = \{\langle G, s, t\rangle : G$ is a directed graph and vertex 't' is reachable from vertex 's' in G $\}$

Proof: We have already proved above that PATH is in NL. We now prove that if $B \in$ NL then $B \leq_l$ PATH. Thus, we would like to devise a function f that is an implicitly log-space computable function such that $x \in B \leftrightarrow f(x) \in$ PATH given that B $\in$ NL.

Let M_b be the NL machine that decides B. Every configuration of M_b takes $c.log(x)$ space and hence there can be at most $2^{c.log(x)}$ configurations which is $\mathcal{O}(poly|x|)$

Since $f(x)$ will be an instance of PATH , let $f(x) = \langle\ G_{(M_b,x)}, c_{start}, c_{accept}\ \rangle$. We now show how to obtain such a graph $G_{(M_b,x)}$ using implicit log-space reduction where c_{accept} is reachable from c_{start} in $G_{(M_b,x)}$ iff $x \in B$. Note that the start configuration c_{start} and accepting configuration c_{accept} are special in a way that they can be distinguished from all other configurations.

Let the configurations of M_b be the nodes of the graph in PATH. Since the total number of configurations are $\mathcal{O}(poly|x|)$, the adjacency matrix of a graph with $\mathcal{O}(poly|x|)$ nodes will also be polynomial in size($\mathcal{O}(poly|x|)$).
Therefore, $|G_{(M_b,x)}| = |x|^c$

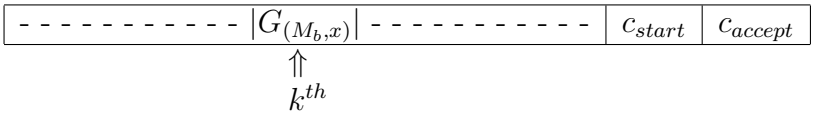

Fig: $f(x)$ on a fictitious tape.

Every k^{th} bit in $G_{(M_b,x)}$ corresponds to some $\langle c_i, c_j\rangle^{th}$ entry in the adjacency matrix of $G_{(M_b,x)}$.

•	c_1	c_2	....	....	c_m
c_1	0	1	...	...	...
c_2	0	0	1	...	1
...	...	...	...	...	0
...	...	...	...	...	1
c_m	...	...	...	...	0

Fig: Adjacency matrix of a $G_{(M_b,x)}$ with m nodes.

The adjacency matrix contains an entry 1 at position $\langle c_i, c_j\rangle$ iff there's an edge from c_i to c_j in $G_{(M_b,x)}$.
Everytime we need to find out some entry $\langle c_i, c_j\rangle$, we run M_b on configuration c_i. We can get only two configurations from a given configuration using the transition function of M_b. In all we need $\mathcal{O}(\log x)$ space to store these two new configurations. We then verify if c_j is one of those. If c_j can be reached then entry $\langle c_i, c_j\rangle = 1$ else 0. Thus the adjacency matrix is constructed in such a way that the entry $\langle c_i, c_j\rangle = 1$ iff c_j can be reached from c_i using the transition function of M_b. A path in $G_{(M_b,x)}$ would hence imply a set of consecutive configurations generated using the transition function of M_b. Every configuration of M_b

used to accept 'x', respectively represents a vertex in the path of c_{start} to c_{accept}. Thus, there is a path from c_{start} to c_{accept} in the graph iff M_b accepts x.♠

Remark: It is unknown whether PATH $\in$ L and is still an open question. If it is proved that PATH $\in$ L, it would immediately imply NL = L since PATH is known to be NL-complete. Also PATH $\notin$ L would imply NL $\neq$ L.

3 Certificate of NL

Let a language $B \in$ NL. Also Let M_b be the NL machine for B.
Given the above two statements, we infer the following points.

1) The verifier uses the certificate u as one of the M_b's non-deterministic choices on input x.

2) M_b uses log-space to verify u.(given the non-deterministic choices of u)

Thus, a certificate(u) can take polynomial space but it may not be possible to store such a certificate on a log-space machine. So the alternative is to store 'u' on a separate tape which is read once. The verifier then needs to read u from left to right only once, never moving the head to the left.

3.1 Alternate Definition of Class NL using a certificate

Definition: A language $B \in$ NL iff there exists a log-space machine M such that $x \in B \leftrightarrow \exists u \in \{0,1\}^{q(|x|)}$ such that $M(x,u) = 1$, where u is a read once certificate and q is a polynomial function.

Remark: The above discussion shows that if $B \in$ NL then indeed there is a log-space machine M and a polynomial function q such that $x \in B \leftrightarrow \exists u \in \{0,1\}^{q(|x|)}$ such that $M(x,u) = 1$ where u is present on a separate 'read once' input tape of M.

3.2 Proving the Converse...

Suppose there is a log-space machine M and a polynomial function q such that $x \in B \leftrightarrow \exists u \in \{0,1\}^{q(|x|)}$ such that $M(x,u) = 1$ where u is given on a separate 'read once' input tape of M then show that $B \in$ NL.

Proof: The machine M' deciding B will essentially try to simulate the verifier except that it does not have an access to the certificate. For each bit the verifier reads, M' non-deterministically guesses a bit. Note that the bit read by the verifier is one of the bit guessed by M'. As the verifier reads the certificate from left to right once, M' keeps on guessing bits non-deterministically. Similarly, as the verifier keeps on computing bits the instance it reads (since the certificate is 'read once'), M' will also keep on computing bits as soon as they are guessed. We know that the size of certificate u is $|q(x)|$ so we halt the machine after counting $|q(x)|$ steps. Such a counter takes $c.log(x)$ bits. When the machine halts, we can decide if $x \in B$. Thus, for every x, we can decide if $x \in B$ using the NL machine described above. Therefore $B \in$ NL. ♠

4 References

S. ARORA and B. BARAK *"Computational Complexity: A Mordern Approach,"* Cambridge University Press, 2009