Computational Complexity **Fall 2018**

NL Completeness

Lecturer: Weiming Feng *Scribes: Weiming Feng*

1 L and NL

The complexity classes **L** and **NL** are defined as follows

- $\mathbf{L} = \mathbf{SPACE}(\log n)$;
- $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

There are many computational problems in complexity classes **L** and **NL**. Here are two examples: bracket matching and PATH.

Bracket matching: Let $S \in \{(,)\}^*$ be a sequence which consists of two symbols '(' and ')'. The matched bracket sequence is defined as follows.

- the empty sequence is a matched bracket sequence;
- if S is a matched bracket sequence, then (S) is also a matched bracket sequence;
- if S_1, S_2 are two matched bracket sequences, then S_1S_2 is also a matched bracket sequence.

For example "()(())" is a matched bracket sequence, but "(()" is not a matched bracket sequence. The bracket matching problem asks to decide whether a sequence $S \in \{(,)\}^*$ is a matched bracket sequence.

This problem is in class **L**. Consider a Turing machine M that scans the sequence S from left to right. M maintains a integer cnt. Initially, set $\mathsf{cnt} = 0$. In i^{th} step, if the i^{th} symbol of S is '(', then $\mathsf{cnt} \leftarrow \mathsf{cnt} + 1$; if the i^{th} symbol of S is ')', then $\mathsf{cnt} \leftarrow \mathsf{cnt} - 1$. The sequence S is accepted by M if and only if $\mathsf{cnt} \geq 0$ during the whole procedure and $\mathsf{cnt} = 0$ after the n^{th} step, where $n = |S|$ is the length of the input sequence.

Exercise: Verify that M solves bracket matching problem with $O(\log n)$ space in work tape. [1]

PATH: Let $G = (V, E)$ be a directed graph and $s, t \in V$ be two vertices. The PATH problem is defined as

$$\mathsf{PATH} \triangleq \{(G, s, t) \mid \text{there is a path from } s \text{ to } t \text{ in } G\}.$$

The PATH problem is in class **NL**.

Exercise: Verify $\mathsf{PATH} \in \mathbf{NL}$.

[1] Exercises are NOT assignments. You do NOT need to submit the solutions of exercises in this note. Exercises in this note are EASY and most solutions can be found in the textbook. You should try to solve exercises by yourself then refer to the textbook.

Read-once certificate: Alternatively, we can define the class **NL** based on read-once certificate. This definition is similar with the certificate-based definition of **NP**. Let M be a logspace Turing machine with an additional read-once certificate tape. The certificate tape is a read-only tape. Initially, the head on such tape is located on the left most place. On each step, the head either stays in the current place or moves to the right. Remark that the head cannot move to the left. Hence the machine can read each bit in certificate tape only once.

Definition 1 (Class **NL**)**.** A language $L \in \mathbf{NL}$ if there exists a deterministic Turing machine M with an additional read-once certificate tape and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for any $x \in \{0,1\}^*$

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)}, \text{ s.t. } M(x,u) = 1,$$

where $M(x,u)$ is the output of M when x is placed on its input tape and u is place on its additional read-once certificate tape, and M uses at most $O(\log |x|)$ bits on its work tape.

Exercise: Verify that two definitions of **NL** are equivalent.

2 Logspace reduction and NL-complete problem

We focus on the complexity question: whether $\mathbf{L} = \mathbf{NL}$. We want to find a class of **NL**-complete problems such that if one of **NL**-complete problem is in **L** then all problems in **NL** are in **L**. We can not use polynomial reduction to defined the complete problems for **NL**. We claim that any language $L_1 \in \mathbf{NL}$ is polynomial-time reducible to the trivial language $L_2 = \{1\}$. For any $x \in \{0,1\}^*$, define

$$f(x) \triangleq \begin{cases} 1 & \text{if } x \in L_1 \\ 0 & \text{if } x \notin L_1. \end{cases}$$

Recall $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P}$. It implies that f can be computed by a polynomial Turing machine. Hence, $x \in L_1$ if and only if $f(x) \in L_2$. It is easy to see $L_2 \in \mathbf{NL}$. The trivial language $L_2 = \{1\}$ is **NL**-complete under the polynomial-time reduction.

We introduce the following logspace reduction.

Definition 2 (Logspace reduction)**.** Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomially-bounded function (there exists a constant c such that $|f(x)| \le |x|^c$ for all $x \in \{0,1\}^*$). We say f is *implicitly logspace computable* if the language $L = \{< x, i > | \ f(x)_i = 1\}$ and $L' = \{< x, i > | \ |f(x)| \le i\}$ are in **L**.

Language A is logspace reducible to language B, denoted as $A \le_l B$, if there exists a function $f : \{0,1\}^* \to \{0,1\}^*$ that is implicitly logspace computable and $x \in A$ if and only if $f(x) \in B$ for all $x \in \{0,1\}^*$.

Informally, f is implicitly logspace computable if there is a $O(\log n)$-space Turing machine that computes every bit of $f(x)$. Note that x is an instance of size $|x| = n$. The reduction function f maps x to another instance $f(x)$. The size of the new instance $|f(x)|$ can be a polynomial of n. However, the logsapce Turing machine only has $O(\log n)$ space in work tape. It can not store the whole $f(x)$. Thus, the Turing machine should be able to computes every bit of $f(x)$.

Lemma 1. *If $A \le_l B$ and $B \le_l C$, then $A \le_l C$; if $A \le_l B$ and $B \in \mathbf{L}$, then $A \in \mathbf{L}$.*

The proof of Lemma 1 is in the textbook.

Definition 3 (**NL**-complete problem)**.** A language L is said to be **NL**-complete if and only if $L \in \mathbf{NL}$ and for any language $L' \in \mathbf{NL}$, it holds that $L' \leq_l L$.

Theorem 2. *PATH is **NL**-complete.*

Proof Sketch: It is easy to verify $\mathsf{PATH} \in \mathbf{NL}$. We prove that for any language $L \in \mathbf{NL}$, it holds that $L \leq_l \mathsf{PATH}$. By the definition of the class **NL**, let M be the logspace non-deterministic Turing machine that decides whether $x \in L$ for any $x \in \{0,1\}^*$. For any input $x \in \{0,1\}^n$ of length n, let $f(x)$ be a PATH instance (G, s, t), where G is the configuration graph [2] of M on input x, s is the start configuration and t is the accepting configuration. Suppose the graph G is represented as an adjacency matrix A. For any two configurations C_1 and C_2, $A(C_1, C_2) = 1$ if there is a directed edge from C_1 to C_2 in configuration graph G and $A(C_1, C_2) = 0$ if otherwise. We claim that $A(C_1, C_2)$ can be computed with $O(\log n)$ space. Since M is a logspace Turing machine, then $|C_1| = |C_2| = O(\log n)$. Given C_1 and C_2, it is easy to check whether $A(C_1, C_2) = 1$ by simulating one step of M. □

Remark. Note that $f(x) = (G, s, t)$ in above reduction. The size of configuration graph G is poly(n). Any logspace Turing machine can not store the whole adjacency matrix A in its work tape. Hence, the Turing machine should be able to computes every bit of $f(x)$ (i.e. every bit of adjacency matrix A).

3 NL= coNL

The class **coNL** is defined as $\mathbf{coNL} \triangleq \{\overline{L} \mid L \text{ is a language and } L \in \mathbf{NL}\}$. Similarly, we say a language L is **coNL**-complete if $L \in \mathbf{coNL}$ and $L' \leq_l L$ for all $L' \in \mathbf{coNL}$. We prove the following theorem in this section.

Theorem 3. ***NL=coNL.***

At first, we introduce a **coNL**-complete problem $\overline{\mathsf{PATH}}$. Let $G = (V, E)$ be a directed graph and $s, t \in V$ be two vertices. The $\overline{\mathsf{PATH}}$ problem is defined as

$$\overline{\mathsf{PATH}} \triangleq \{(G, s, t) \mid \text{there is no path from } s \text{ to } t \text{ in } G\}.$$

Exercise: Verify that $\overline{\mathsf{PATH}}$ is **coNL**-complete.

Theorem 3 can be proved by the following lemma.

Lemma 4. $\overline{\mathit{PATH}} \in \mathbf{NL}$.

Exercise: Verify that $\overline{\mathsf{PATH}} \in \mathbf{NL}$ implies $\mathbf{NL} = \mathbf{coNL}$.

Proof of Lemma 4. We prove that if two vertices s and t are disconnected in graph $G(V, E)$, then there is a read-once certificate.

[2] The definition of configuration graph can be found in the textbook and the slides of lecture 4.

Let $C_i \subseteq V$ be the set of vertices that are reachable from s in G within at most i steps

$$C_i \triangleq \{u \in V \mid \text{dist}_G(s, u) \leq i\}.$$

If we can provide a read-once certificate to indicate whether $t \in C_{n-1}$, then the lemma is proved. This is because that the distance between s and t is at most $n-1$ if s and t is connected, where n is the number of vertices in graph G.

We claim that the following three kinds of certificates can be designed.

- A certificate that vertex v is in C_i for all $v \in C_i$
- A certificate that vertex v is not in C_i for all $v \notin C_i$, assuming the verifier already knew the size of set C_i.
- A certificate that $|C_i| = c$, assuming the verifier already knew the size of set C_{i-1}.

Since $C_0 = \{s\}$, then the size of C_0 must be 1. We can use the third kind of certificate iteratively to convince the verifier of the sizes of the sets $C_1, C_2, \ldots, C_{n-1}$. Finally, we use the first two kinds of certificates to convince the verifier that whether $t \in C_{n-1}$.

Certifying that $v \in C_i$: The certificate is a sequence of vertices $v_0, v_1, \ldots, v_k$ such that $v_0 = v$, $v_k = s$, $k \leq i$ and $\{v_i, v_{i-1}\} \in E$ for all $1 \leq i < k$.

Certifying that $v \notin C_i$ given $|C_i|$: Let $u_1, u_2, \ldots, u_k$ be all vertices in C_i, where $k = |C_i|$. Assume that $u_1 < u_2 < \ldots < u_k$. The certificate is the sorted list of certificates that $u_j \in C_i$ for $1 \leq j \leq k$ (certificate sequence: $u_1 \in C_i, u_2 \in C_2, \ldots, u_k \in C_i$). The verifier checks each certificate in the list, checks the total number of certificates is indeed $|C_i|$ and checks each vertex u for which a certificate is given is indeed larger than the previous vertex. If $u_j \neq v$ for all $1 \leq j \leq k$, then it must hold that $v \notin C_i$. Conversely, if $v \in C_i$, there must exist a $u_j = v$.

Certifying that $v \notin C_i$ given $|C_{i-1}|$: Similarly, let $u_1, u_2, \ldots, u_k$ be all vertices in C_{i-1}, where $k = |C_{i-1}|$. Assume that $u_1 < u_2 < \ldots < u_k$. The certificate is the sorted list of certificates that $u_j \in C_i$ for $1 \leq j \leq k$ (certificate sequence: $u_1 \in C_i, u_2 \in C_2, \ldots, u_k \in C_i$). The verifier checks each certificate in the list, checks the total number of certificates is indeed $|C_{i-1}|$ and checks each vertex u for which a certificate is given is indeed larger than the previous vertex. If vertex v and all of v's neighbors does not in $u_1, u_2, \ldots, u_k$, then it must hold that $v \notin C_i$. Conversely, if $v \in C_i$, then either vertex v or one of v's neighbors must be in set C_{i-1}.

Certifying that $|C_i| = c$ given $|C_{i-1}|$: The certificate is the list of n certificates for each vertex 1 to n in ascending order (certificate for vertex 1, certificate for vertex $2, \ldots$, certificate for vertex n). For each vertex $u \in V$, the certificate either indicates $u \in C_i$ or $u \notin C_i$. The verifier checks each certificate in the list, checks the total number of certificates is indeed n and checks each vertex u for which a certificate is given is indeed larger than the previous vertex. If there are exactly c certificates indicating that a vertex belong to set C_i, then the size of C_i must be c. □