Lesson 2: The class NL

**Theme:** Some classical results on the class NL.

We recall the notion of log-space reduction. Let \( F : \Sigma^* \to \Sigma^* \) be a function. We say that \( F \) is computable in logarithmic space, if there is a 3-tape DTM \( M \) such that on input word \( w \), it works as follows.

- Tape 1 contains the input word \( w \) and its content never changes.
- There is a constant \( c \) such that \( M \) uses only \( c \log |w| \) space in tape 2.
- The head in tape 3 can only “write” and move right, i.e., once it writes a symbol to a cell, the content of that cell will not change.

Tape 1 is called the *input tape*, tape 2 the *work tape* and tape 3 the *output tape*.

**Definition 2.1** A language \( L \) is log-space reducible to another language \( K \), denoted by \( L \leq_{\log} K \), if there is a function \( F : \Sigma^* \to \Sigma^* \) computable in logarithmic space such that for every \( w \in \Sigma^* \), \( w \in L \) if and only if \( F(w) \in K \).

**Remark 2.2** The relation \( \leq_{\log} \) is transitive in the sense that if \( L_1 \leq_{\log} L_2 \) and \( L_2 \leq_{\log} L_3 \), then \( L_1 \leq_{\log} L_3 \).

**Definition 2.3** Let \( K \) be a language.

- \( K \) is **NL-hard**, if for every language \( L \in \text{NL} \), \( L \leq_{\log} K \).
- \( K \) is **NL-complete**, if \( K \in \text{NL} \) and \( K \) is NL-hard.

Define the following language \( \text{PATH} \).

\[
\text{PATH} \overset{\text{def}}{=} \{(G, s, t) : G \text{ is directed graph and there is a path in } G \text{ from vertex } s \text{ to vertex } t\}
\]

**Theorem 2.4** \( \text{PATH} \) is NL-complete.

**Theorem 2.5** (Savitch 1970) \( \text{NL} \subseteq \text{DSPACE}[\log^2 n] \).

To prove Theorem 2.5 it suffices to show that \( \text{PATH} \in \text{DSPACE}[\log^2 n] \). See Appendix A.

**Theorem 2.6** (Immerman 1988 and Szelepcsényi 1987) \( \text{NL} = \text{coNL} \).

To prove Theorem 2.6, we consider the complement language of \( \text{PATH} \):

\[
\text{PATH} \overset{\text{def}}{=} \{(G, s, t) : G \text{ is directed graph and there is no path in } G \text{ from vertex } s \text{ to vertex } t\}
\]

Note that \( \text{PATH} \) is coNL-complete. To prove Theorem 2.6 it suffices to show that \( \text{PATH} \in \text{NL} \). See Appendix B.
Appendix

A  Proof of Theorem 2.5

Algorithm 1 below decides the language PATH.

Algorithm 1
Input: $(G, s, t)$, where $G$ is a directed graph and $s$ and $t$ are two vertices in $G$.
Task: ACCEPT iff there is a path in $G$ from $s$ to $t$.
1: Let $n$ be the number of vertices in $G$.
2: ACCEPT iff $\text{CHECK}_G(s, t, \lceil \log n \rceil) = \text{true}$.

It uses Procedure CHECK$_G$ defined below.

Procedure CHECK$_G$
Input: $(u, v, k)$ where $u$ and $v$ are two vertices in $G$, and $k$ is an integer $\geq 0$.
Task: Return true, if there is a path in $G$ of length $\leq 2^k$ from $u$ to $v$. Otherwise, return false.
1: if $k = 0$ then
2: return true iff $(u = v$ or $(u, v)$ is an edge in $G$).
3: for all vertex $x$ in $G$ do
4: $b := \text{CHECK}_G(u, x, k - 1)$.
5: if $b = \text{true}$ then
6: $b := \text{CHECK}_G(x, v, k - 1)$.
7: if $b = \text{true}$ then
8: return true.
9: return false.

Note that when computing CHECK$_G(u, x, k - 1)$ and CHECK$_G(x, v, k - 1)$, Procedure CHECK$_G$ can use the same space. Thus, it uses only $O(k \log n)$ space. Since $k$ is initialized with $\lceil \log n \rceil$, Algorithm 1 uses $O(\log^2 n)$ space in total.

B  Proof of Theorem 2.6

Consider the following algorithm.

Algorithm No-path
Input: $(G, s, t)$ where $G$ is directed graph and $s$ and $t$ are two vertices in $G$.
Task: ACCEPT iff there is no path in $G$ from $s$ to $t$.
1: $m :=$ the number of vertices in $G$ reachable from $s$.
2: {Note: This value $m$ is computed with Procedure COUNT-VERTEX$_G$ below.}
3: for all vertex $x$ in $G$ do
4: Guess if $x$ is reachable from $s$.
5: if the guess is “yes” then
6: $m := m - 1$.
7: Guess a path from $s$ to $x$.
8: if it is not possible to guess such a path then REJECT.
9: if there is such a path and $x = t$ then REJECT.
10: ACCEPT iff $m = 0$.
The number of vertices reachable from $s$ can be computed with Procedure $\text{COUNT-VERTEX}_G$ defined below.

**Procedure** $\text{COUNT-VERTEX}_G$

**Input:** $u$ where $u$ is a vertex in $G$.

**Task:** Return the number of vertices in $G$ reachable from vertex $u$, where the number is written in binary form.

1. Let $n$ be the number of vertices in $G$.
2. $m := 1 + \text{the outdegree of } u$.
3. $\{\text{Note: } m \text{ is initialized with the number of vertices reachable from } u \text{ in } \leq 1 \text{ steps.}\}$
4. for $i = 2, \ldots, n$ do
5.     $m' := 0$.
6.     for all vertex $x$ in $G$ do
7.         Guess if there is a path from $u$ to $x$ with length $\leq i$.
8.         if the guess is “yes” then
9.             Verify it by guessing such a path (of length $\leq i$).
10.        $m' := m' + 1$.
11.        if the guess is “no” then
12.           Verify that indeed there is no such a path (of length $\leq i$).
13.        $m := m'$.
14. $\{\text{Note: On each iteration, } m \text{ is the number of vertices reachable from } u \text{ in } \leq i \text{ steps.}\}$
15. return $m$

The verification in Line 12 above is done with the following procedure.

**Procedure** $\text{VERIFY}_G$

**Input:** $(u, x, m, i)$ where $u$ and $x$ are vertices in $G$, $i \geq 2$ is an integer and $m$ is the number of vertices in $G$ reachable from $u$ in $\leq i - 1$ steps.

**Task:** Verify that $x$ is not reachable from $u$ in $\leq i$ steps.

1. $\ell := m$.
2. for all vertex $y$ in $G$ do
3.     Guess if there is a path from $u$ to $y$ with length $\leq i - 1$.
4.     if the guess is “yes” then
5.         $\ell := \ell - 1$.
6.     Guess a path (of length $\leq i - 1$) from $u$ to $y$.
7.     Verify that the edge $(y, x)$ does not exist in $G$.
8. Verification is complete iff $\ell = 0$.

Note that if any of the verification in Lines 9 and 12 in Procedure $\text{COUNT-VERTEX}_G$ and Line 7 in Procedure $\text{VERIFY}_G$ fails, the whole algorithm rejects immediately.

The correctness of Procedure $\text{COUNT-VERTEX}_G$ can be established by induction on $i$. The correctness of Algorithm $\text{NO-PATH}$ follows immediately from $\text{COUNT-VERTEX}_G$. 