Logarithmic Space and Time

21.1 Logarithmic Space

If we want to consider space that is sublinear, meaning smaller than $O(n)$, we have to not count the input tape. To make it fair, we insist that the input tape is read only. Recall that $\mathcal{L}$ is the set of all languages that can be decided using only $O(\log n)$ workspace. $\mathcal{NL}$ is the nondeterministic counterpart. Logarithmic space is not much: it can hold a counter, an index into the input tape, or the number of a node in a graph. For example, we saw that $\{0^n1^n\}$ is in $\mathcal{L}$: use one counter to count the 0s and one to count the 1s and compare the counters. An exercise asked you to show that, for example, Palindromes is in $\mathcal{L}$.

**Example.** The language $\text{PATH}$ is in $\mathcal{NL}$. Recall that the decision problem $\text{PATH}$ has input a graph and two nodes $s$ and $t$, and the question is whether there is a path from the start node $s$ to the target node $t$.

We need an $\mathcal{NL}$-algorithm. The idea is to guess the path one node at a time; that is, guess the next node and check that there is an edge from the current node to the next. Accept if you ever reach the target node $t$. (If desired, one can add a counter to stop an infinite computation.) The storage is the number of the current node and next node, as well as pointers into the input-tape to search for the requisite edge.

It is not known whether $\text{PATH}$ is in $\mathcal{L}$. Recall that we do know that $\mathcal{L} \neq \mathcal{PSPACE}$. This is a consequence of a general hierarchy theorem that shows that as the bound on the space grows, the set of languages grows. (The key tool is to use diagonalization to build a machine that accepts a language that is different from every machine that runs in “smaller” space.)

21.2 Completeness

We need a restricted version of a reduction, which we call $\leq_\ell$. The point is that we want it to be true that, if $A \leq_\ell B$ and $B \in \mathcal{L}$, then $A \in \mathcal{L}$. So the reduction should use only logarithmic space. We define a logspace transducer to have a read-only input tape, a write-only output tape, and $O(\log n)$-size work-tape. We say that $A \leq_\ell B$ if there is a function $f$ computable by a logspace transducer such that $f(w) \in B$ if and only if $w \in A$. A language being $\mathcal{NL}$-complete is defined as expected. And the standard implications follow. For example, if $A \leq_\ell B$ and $B \in \mathcal{NL}$, then $A \in \mathcal{NL}$. The proofs are left as an exercise.

Here is an example of an $\mathcal{NL}$-complete language.
**Thm.** \( \text{PATH} \) is \( \mathcal{NC} \)-complete.

**Proof sketch.** Let language \( A \in \mathcal{NL} \) be accepted by machine \( N \). If \( w \in A \) then there is an accepting computation of \( N \). Our logspace transducer, on input \( w \), is going to output a graph \( G_w \) with designated start and target nodes. A path will correspond to a possible computation, so that there is a path from start to target node iff \( w \in A \).

We build the graph by having every possible configuration of \( N \) be a node of \( G_w \). Note that a configuration takes \( O(\log n) \) space to store. For, though we do not have to include the input-tape because it never changes, we do have to include the position of the head on the input-tape and the contents of the work-tape.

And the important part is that we can iterate through valid configurations in \( O(\log n) \) space. The idea is to simply generate every possible string of every possible length and discard those strings that do not have the correct syntax. Then, by iterating through pairs of configurations, we can output the graph \( G_w \) as a list of arcs. Each arc says that it is possible to go from the first configuration to the second configuration in one step of \( N \). (Note that the logspace transducer is given \( N \) as well as \( w \). So, for example, it can compute from \( w \) what symbol would be read on the input tape for the head position of a given configuration, and compute from \( N \) how the configuration changes.)

Then the question on the graph is whether it is possible to go from the start configuration to the accept configuration. This completes the reduction.

\[ \diamond \]

### 21.3 The complement of \( \text{PATH} \)

The language \( \overline{\text{PATH}} \) is by definition the set of all strings not in \( \text{PATH} \). However, it is usually intended to mean the set of all graph, start and target node where there is no path from the start to the target node.

The following result is due to both Immerman and Szelpcsenyi. Note that it uses the fact that \( \text{PATH} \) is in \( \mathcal{NL} \! \)!

**Thm.** \( \overline{\text{PATH}} \) is in \( \mathcal{NC} \)

**Proof sketch.** The idea is to determine the balls around the start node. We define the ball \( A_i \) as the set of all nodes within distance \( i \) of the start node. Note that we can determine \( A_{i+1} \) as follows:

*for every node \( v \): iterate through the nodes \( x \) in \( A_i \) and see if there is an edge from \( x \) to \( v \).*

However, we do not have sufficient space to store \( A_i \). The wonderful idea is that:

*one can instead determine membership in \( A_{i+1} \) using only the size of \( A_i \), which we call \( c_i \).*

7
To iterate through the nodes in $A_i$, we instead: guess $c_i$ distinct nodes $x$, and for each of these nodes guess the path from the start node to $x$.

Now, note that we make considerable use of the nondeterminism. The branches that guess the wrong nodes, or the branches that guess the wrong path, will simply die. The only branches that remain alive have correctly determined $A_i$ and therefore $c_i$.

We also reject if ever a path is found from the start to the target. So the overall algorithm is to start with $c_0 = 1$ and repeatedly calculate $c_{i+1}$ from $c_i$. If we finish calculating $c_n$ and never find a path from the start to the target, then (and only then) do we accept. ♦

The class co-$\mathcal{NL}$ is defined as the set of complements of languages of $\mathcal{NL}$. As a consequence of the above it follows that $\mathcal{NL} = \text{co-$\mathcal{NL}$}$. On the other hand, it is believed, though unproven, that $\mathcal{NP} \neq \text{co-$\mathcal{NP}$}$.

### 21.4 Parallel Computation

In logarithmic time, we cannot read the entire input. So to get logarithmic time, we use multiple processors. We briefly introduce here the class $\mathcal{NC}$.

One can define a parallel TM as the following model: some number of processors, a common work tape, and each processor having their own tape for writing the address of a position in the work tape. Essentially this is equivalent to a CREW PRAM (concurrent read, exclusive write, parallel random-access machine). Then $\mathcal{NC}$ is defined to be the set of all languages that can be accepted with a polynomial number of processors and poly-logarithmic time. (Poly-logarithmic means at most $O(\log^i n)$ for some $i$.)

**Example.** Parity of $n$ bits is in $\mathcal{NC}$.

**Example.** $\text{PATH}$ is in $\mathcal{NC}$. Use recursion. If there is a path at all, then there is one of length at most $n - 1$ where $n$ is the number of nodes. To find whether there is a path from $s$ to $t$ of at most some specified length $k$, try all possible midpoints $u$ in parallel, testing for a $s$-to-$u$ path and a $u$-to-$t$ path of length $k/2$.

Here is the relationship with previous classes:

**Thm.** (a) $\mathcal{NL} \subseteq \mathcal{NC}$.
(b) $\mathcal{NC} \subseteq \mathcal{P}$.

**Proof idea.** (a) For the input-length, there exists configuration graph $CG$. For that input, determine if can reach accepting configuration. Takes recursion of depth $\log |CG|$, each level performing boolean matrix multiplication (one entry in transitive closure).

(b) Direct simulation. ♦
So, there is a big chain of containments:

\[ \mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{NC} \subseteq \mathcal{P} \]

**Exercises**

21.1. Show that $\mathcal{NL}$ is closed under the star operation.

21.2. (a) Show that if $A \leq_{\ell} B$ and $B \in \mathcal{L}$ then $A \in \mathcal{L}$.

(b) Explain why if $A \leq_{P} B$ and $B \in \mathcal{L}$ then it does not automatically follow that $A \in \mathcal{L}$.

21.3. Show that $\mathcal{NC}$ is closed under union. What about complementation?

21.4. Propose a definition of $\mathcal{NC}$-completeness, and discuss the implications of knowing that a language is $\mathcal{NC}$-complete. Do the same with $\mathcal{P}$-completeness.

21.5. Show that $Empty_{\text{NFA}}$ is in $\mathcal{NL}$.

21.6. Is $\text{HAMPATH}$ in $\mathcal{NL}$? In $\mathcal{L}$? Explain.

21.7. Show that the addition of two $n$-bit numbers is in $\mathcal{NC}$.

21.8. Convince your inner self that $\mathcal{L}$ contains all regular languages.