1 Savitch’s Theorem

We’ll prove an amazing result: Nondeterministic space algorithms can be simulated efficiently by deterministic space algorithms, with only quadratic loss in space usage. That is, nondeterminism does not give us extra power in the case of space-bounded computation!

Recall that $\text{NL} = \text{NSpace}(\log n)$, and $\text{NPSPACE} = \bigcup_k \text{NSpace}(n^k)$.

First, using the notion of configuration graphs, we can show the following.

**Theorem 1.** $\text{NL} \subseteq \text{P}$

**Proof.** The proof is exactly the same as that for $\text{L} \subseteq \text{P}$. □

Configuration graphs of space-bounded TMs are a very useful tool for analyzing space-bounded computation. As the next theorem shows a reachability problem for graphs exactly captures the complexity of the class $\text{NL}$.

Define $\text{ST-CON} = \{(G, s, t) \mid G \text{ is a directed graph with a path from } s \text{ to } t\}$.

**Theorem 2.** $\text{ST-CON}$ is $\text{NL}$-complete under logspace reductions.

**Remark:** Note that we restricted the class of reductions to logspace computable ones. This is necessary to make the notion $\text{NL}$-completeness nontrivial. (As you recall from Problem Set 1, basically every language in $\text{P}$ is $\text{P}$-complete under polytime reductions.)

**Proof.**

1. $\text{ST-CON} \in \text{NL}$: Given a graph $G = (V, E)$ and $s, t \in V$, nondeterministically guess a path from $s$ to $t$, by keeping track of a current vertex on a path and a next vertex on a path. After guessing a next vertex on a path, make it the new current vertex (erasing the old current vertex) so as to re-use the space, and run in $O(\log n)$ space only. If ever $t$ is a current vertex on a guessed path, then Accept.

2. $\text{ST-CON}$ is $\text{NL}$-hard: Given any language $L \in \text{NL}$ decided by some logspace NTM $M$, and an input $x$, construct the configuration graph of $M$ on $x$. This can be done in logspace (can you see why?) Make $s$ to be the start configuration, and $t$ the accepting configuration. (Note: we can always modify any given NTM so that it has only one accepting configuration: after entering $q_{\text{accept}}$, the machine will erase all of its worktapes, and go to the first non-blank symbol of its input tape.) □

Another representation of $\text{NL}$, which suggested to Immerman the idea of the proof of $\text{NL} = \text{coNL}$ (which we will do later today), is first-order logic with a transitive closure operator. Transitive closure is essentially reachability: a pair of vertices $(s, t)$ is in the

---

1This lecture is a modification of notes by Valentine Kabanets
transitive closure of a graph if \( t \) is reachable from \( s \). In the logic form, transitive closure of a relation \( E \) is \( X \) satisfying the following:

\[
X_0(u, v) \leftarrow E(u, v) \lor u = v \\
X_{i+1}(u, v) \leftarrow \exists z < n X_i(u, z) \land X_i(z, v)
\]

That is, it is the smallest value of \( X \) such that \( X_{i+1} = X_i \). In a more general case, instead of the edge relation \( E \) we can put any first-order formula \( \phi \), which may have \( X \) as a free variable.

Recall from the proof of Fagin’s theorem that the correctness of a run of a Turing machine can be described by a first-order formula (with arithmetic operators). Now, together with the fact that a computation of an NL machine can be thought of as a reachability in its configuration graph, it is easy to see that \( FO(TC) \) (first-order with transitive closure) captures NL.

**Theorem 3 (Savitch’s Theorem).** \( ST - CON \in \text{Space}(\log^2 n) \).

**Corollary 4.**

1. \( \text{NL} \subseteq \text{Space}(\log^2 n) \).
2. \( \text{NPSPACE} = \text{PSPACE} \).

**Proof of Savitch’s Theorem.** We will design a \( \log^2 n \)-space algorithm that, given a directed graph \( G = (V, E) \) with \( |V| = n \) nodes, and \( s, t \in V \), will accept iff \( t \) is reachable from \( s \). (For convenience, we assume that \( (u, u) \in E \) for every node \( u \in V \).) The idea of the proof is similar to the second line of the definition of transitive closure.

We design algorithm \( \text{Path}(x, y, i) \) which accepts iff there is a path from \( x \) to \( y \) of length at most \( 2^i \). Note that \( t \) is reachable from \( s \) iff \( \text{Path}(s, t, \log n) \) accepts. (Do you see why?)

**Algo** \( \text{Path}(x, y, i) \)

if \( i = 0 \) then
   Accept iff \( (x, y) \in E \)
end if

for every \( z \in V \)
   if \( \text{Path}(x, z, i - 1) \) accepts AND
      \( \text{Path}(z, y, i - 1) \) accepts \% we re-use space here!
   then
      Accept
   end if
end for

Reject \% if no “middle” point \( z \) was found, we reject
end Algo

It is not hard to see that algorithm \( \text{Path} \) is correct. To analyze the space used, note that the depth of the recursion is \( \log n \), and that the size of each “stack record” during the recursion is the size of \( (x, y, i) \in O(\log n) \). Thus, the total space used is \( O(\log^2 n) \). \( \Box \)
It is still a big open problem to decide if $\text{NL} = \text{L}$. To show this, it would suffice to give a deterministic logspace algorithm for ST-CON, the problem of st-connectivity for directed graphs on $n$ vertices. Interestingly, Reingold has recently showed that the st-connectivity problem for undirected graphs is solvable in deterministic logspace! The algorithm for doing this is highly nontrivial and not at all obvious; it is based on the algebraic characterization of connectivity in graphs (in terms of the so-called eigenvalue gap, the difference between the two largest eigenvalues of the adjacency matrix of a given graph). This algorithmic success renewed the interest in the $\text{NL} \text{ vs. } \text{L}$ question.

Next we will see another surprising result showing that $\text{NL} = \text{coNL}$, something we don’t expect to be true in the setting of time complexity classes. (This might be taken as another piece of evidence pointing to the possibility of $\text{NL} = \text{L}$...)

2 $\text{NL} = \text{coNL}$

Next we turn to an even more amazing result in complexity which proves the closure of $\text{NL}$ under complementation. (This is like $\text{NP} = \text{coNP}$ for space-bounded machines!) The question whether nondeterministic space is closed under complementation was open for 23 years; it was first stated in 1964 by Kuroda in relation to the class of context sensitive languages, which Kuroda proved to be exactly the class $\text{NSpace}(n)$. In 1987, Neil Immerman and Robert Szelepcsényi, a Slovakian undergraduate student, independently proved that $\text{NSpace}(s(n)) = \text{coNSpace}(s(n))$, for any proper complexity function $s(n) \geq \log n$. This was a big shock to the CS community for two reasons: (1) it was widely believed that $\text{NL} \neq \text{coNL}$, and (2) the proofs by Immerman and Szelepcsényi were quite simple. Immerman says that his proof comes from his attempts to understand $\text{FO}(\text{TC})$: there, he realized that positive occurrences of transitive closure can simulate negative occurrences, and from that designed an $\text{NL}$ algorithm for unreachability.

Since $\text{ST-CON}$ is $\text{NL}$-complete, in order to prove $\text{NL} = \text{coNL}$, it suffices to prove that $\text{ST-CON} \in \text{coNL}$, i.e., that it can be checked in nondeterministic logspace whether $t$ is not reachable from $s$.

Theorem 5 (Immerman-Szelepcsényi). $\text{ST-CON} \in \text{coNL}$

Proof. Idea: To check if $t$ is not reachable from $s$, enumerate all nodes that are reachable from $s$ and check that $t$ is not among them.

This sounds too easy. The trick is to do this enumeration of all nodes in logspace, and ensuring that indeed all nodes reachable from $s$ were enumerated. We need some clever idea to do this. The clever idea is to count.

Let us imagine for a moment that we are given a number $N = \#$ of nodes reachable from node $s$. (Later we’ll show how to compute this $N$ in $\text{NL}$.) The following $\text{NL}$ algorithm check if $t$ is not reachable from $s$ in a given directed graph $G = (V, E)$, where $|V| = n$.

Algo Unreach($G, s, t$)  
% given $N = \#$ nodes reachable from $s$
\[\text{count} = 0;\]
\[\text{for}\ \text{every node } v\]
\[\quad \text{“make a nondeterministic guess whether } v \text{ is reachable from } s”\]
\[\quad \text{if } \text{guess is Yes then}\]
\[\quad \quad \text{“nondeterministically try to guess a path from } s \text{ to } v \text{ of length at most } n”;\]
\[\quad \quad \text{if “guessed path does not lead to } v\text{” then Reject} \quad \text{end if}\]
\[\quad \quad \text{if } v = t \text{ then Reject}\]
\[\quad \text{else count} = \text{count} + 1;\]
\[\quad \text{end if}\]
\[\text{end if}\]
\[\text{end for}\]
\[\text{if } \text{count} < N \text{ then Reject}\]
\[\text{else Accept \% if count} = N\]
\[\text{end if}\]
\[\text{end Algo}\]

Clearly the algorithm Unreach runs in nondeterministic logspace; observe that \(N\) and \(\text{count}\) can be at most \(n\), and so they can be written as binary numbers of length at most \(\log n\).

**Claim 6.** Algorithm Unreach\((G, s, t)\) has an accepting computation iff \(t\) is not reachable from \(s\).

**Proof of Claim.** The algorithm makes sure that it enumerates all nodes reachable from \(s\), by comparing \(\text{count}\) with \(N\). The algorithm accepts iff node \(t\) was not one of these \(N\) nodes reachable from \(s\).

To compute \(N = \#\) nodes reachable from \(s\), we will iteratively compute (re-using space) the values \(R(i) = \#\) nodes reachable from \(s\) in at most \(i\) steps. Then we obtain \(N = R(n)\).

**Algo \#Reach\((G, s, t)\)**
\[
R(0) = 1 \% \ s \text{ is reachable from } s \text{ in 0 steps} \\
\text{for } i = 1..n \\
\quad R(i) = 0 \% \text{ initialize } R(i) \\
\text{for } \text{every node } v \\
\quad \% \text{ try all nodes } u \text{ reachable from } s \text{ in } \leq (i - 1) \text{ steps, and} \\
\quad \% \text{ check if } v \text{ is reachable in } \leq 1 \text{ steps from any such } u \\
\quad \text{count} = 0; \\
\quad \text{for } \text{every node } u \\
\quad \quad \text{“make nondeterministic guess whether } u \text{ is reachable from } s \text{ in } \leq (i - 1) \text{ steps”;} \\
\quad \quad \text{if “guess is Yes” then} \\
\quad \quad \quad \text{“nondeterministically try to guess a path from } s \text{ to } u \text{ of length } \leq (i - 1);\] \\
\quad \quad \text{if “guessed path does not lead to } u\text{” then Reject} \quad \text{end if}\]
\[
\quad \text{count} = \text{count} + 1; \% \text{ if } u \text{ is reachable, count it in} \\
\quad \text{if } u = v \text{ OR } (u, v) \in E \\
\]
then $R(i) = R(i) + 1$
    break; qw go to next iteration of “for $v$” loop
end if
end if
end for
if $count < R(i - 1)$ then Reject
end if
end for
end for
return $R(n)$;
end Algo

Remark: The algorithm \textit{Reach} needs to remember only two successive values $R(i)$ and $R(i + 1)$ at any point in time. So, it re-uses space when computing $R(1), \ldots, R(n)$. Thus, the algorithm can be made to run in $\text{NL}$.

Claim 7. Algorithm \textit{Reach} computes the number of nodes reachable from $s$.

Proof of Claim. The proof is by induction on $i$. For $i = 0$, $R(0) = 1$ is obviously correct.

For the induction step, assume that $R(i)$ is equal to the number of nodes reachable from $s$ in at most $i$ steps. We need to prove that $R(i + 1)$ is equal to the number of nodes reachable from $s$ in at most $(i + 1)$ steps. To prove this, notice that the algorithm increments $R(i + 1)$ on a node $v$ iff $v$ is reachable from $s$ in at most $(i + 1)$ steps. This is because $R(i + 1)$ is not incremented only if all nodes at distance $\leq i$ from $s$ were tried, and $v$ is not reachable in $\leq 1$ steps from any one of them.

Thus, to check if $t$ is not reachable from $s$, we first run the algorithm \textit{Reach} to compute $N$, then run the algorithm \textit{Unreach} with that $N$. The total space this nondeterministic procedure takes is $O(\log n)$, because each of the two algorithms is logspace.