CPSC 468/568: Lecture 6 (January 29, 2015)

This lecture began with Def. 4.1 in your textbook (space-bounded computation, both deterministic and nondeterministic), the notion of "configuration graphs" (as defined in the text immediately preceding Claim 4.4 in your book), and the fact that

$$\mathrm{DTIME}(S(n)) \subseteq \mathrm{SPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))}),$$

which is Theorem 4.2 in your book.

The first two inequalities of Theorem 4.2 are trivial, and the third is easy to prove. Let W be a nondeterministic TM that runs in space $S(n)$; we seek a deterministic algorithm that runs in time $2^{O(S(n))}$, on input $x \in \{0,1\}^n$, and decides whether $x \in L(W)$.

As explained in class, each configuration of W can be encoded in $c \cdot S(n)$ bits, where the constant c depends on the alphabet size, number of states, and number of writable tapes in W. (Recall that the contents of the input tape are not included in the configuration. So this is true even if $S(n) = o(n)$, as long as $S(n) \geq \log n$.) Thus, the configuration graph $G_{W,x}$ has at most $2^{c \cdot S(n)}$ nodes. Moreover, the out-degree of any node in this (directed) graph is two, because we can assume without loss of generality that W has exactly two transition functions δ_0 and δ_1.

Therefore, in $\mathrm{DTIME}2^{O(S(n))}$, we can explicitly *construct* $G_{W,x}$ (using $2^{(O(S(n)))}$ space as well as time) and use a linear-time DFS or BFS algorithm to determine whether it contains a path from its START configuration $C^{W,x}_{\mathrm{START}}$ to its ACCEPT configuration $C^{W,x}_{\mathrm{ACCEPT}}$. The input x is in $L(W)$ if and only if the graph contains such a path.

We concluded this lecture with a fundamental fact about the relationship of nondeterministic space-bounded computation and deterministic space-bounded computation.

Savitch's Theorem: If S is a space-constructible function, and $S(n) \geq \log n$, then $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{SPACE}((S(n))^2)$.

Proof. Let L be a language recognized in space $O(S(n))$ by nondeterministic Turing Machine W, and let $x \in \{0,1\}^n$ be an input that may or may not be in L. Consider the configuration graph $G_{W,x}$. We will define a deterministic machine that, on input x, decides whether there is a path from $C^{W,x}_{\mathrm{START}}$ to $C^{W,x}_{\mathrm{ACCEPT}}$, where these are the unique START and ACCEPT nodes in $V(G_{W,x})$. Recall that, if there is a path from $C^{W,x}_{\mathrm{START}}$ to $C^{W,x}_{\mathrm{ACCEPT}}$, there is one of length $O(2^{c \cdot S(n)})$, for some positive constant c, *i.e.*, that $|V(G_{W,x})| = O(2^{c \cdot S(n)})$.

The deterministic algorithm that we provide actually solves the more general decision problem $\mathrm{REACH}(u, v, i)$, which is 1 if there exists a path from u to v in $G_{W,x}$ of length at most 2^i and 0 if there is no such path. The algorithm is defined recursively.

For $i = 0$ (the base case of the recursion), the algorithm simply checks whether v is one of the two configurations that can be reached from u in one step, *i.e.*, in one application of one of the transition functions δ_0 and δ_1 that define W. (Think about why that can be done in space $O(S(n))$.)

For $i > 0$, we ask whether there is a configuration z such that $\mathrm{REACH}(u, z, i-1)$ and $\mathrm{REACH}(z, v, i-1)$ are both 1. The two crucial points are:

- We can cycle through all (exponentially many) candidates for z and, having concluded that a particular z_j did not have the requisite property, reuse the space we just used for z_j to do the computation for z_{j+1}.
- For a particular z, we can compute $\text{REACH}(u, z, i-1)$ and then reuse the space to compute $\text{REACH}(z, v, i-1)$.

Let $\mathcal{S}_{M,i}$ be the space required to compute $\text{REACH}(u, v, i)$ on a configuration graph $G_{W,x}$ with M nodes. To decide whether there is exists a path from u to v, we would use space at most $\mathcal{S}_{M,\log M}$. We have the recurrence relation

$$\mathcal{S}_{M,i} = \mathcal{S}_{M,i-1} + O(\log M),$$

because space $\mathcal{S}_{M,i-1}$ is needed for recursive calls, and space $O(\log M)$ is needed to write down the "midpoint configuration" z. Solving this recurrence relation gives us $\mathcal{S}_{M,\log M} = O((\log M)^2)$. For nondeterministic machine W, we have $M = O(2^{c \cdot S(n)})$, and thus $\mathcal{S}_{M,\log M} = O((S(n))^2)$. ∎

Note that Savitch's Theorem implies that PSPACE = NPSPACE.