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

This material was presented in class on February 9 and 11, 2016. It uses definitions 4.1, 4.5, 4.16, and 4.19 and the notion of "configuration graph," all of which are presented clearly in the textbook and hence won't be repeated here.

We first observed 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 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.

Next, we covered a fundamental fact about the relationship of nondeterministic space-bounded computation and deterministic space-bounded computation. Recall that $S : \mathbb{N} \longrightarrow \mathbb{N}$ is *space-constructible* if there is a TM that, on input x, computes $S(|x|)$ in space $O(S(|x|))$.

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 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 REACH$(u, z, i-1)$ and 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 REACH$(u, z, i-1)$ and then reuse the space to compute REACH$(z, v, i-1)$.

Let $\mathcal{S}_{M,i}$ be the space required to compute 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.

We conclude with the proof that PATH in NL-complete under deterministic logspace reductions.

Claim 1 *PATH is in NL.*

Proof. A PATH instance is a triple (G, s, t), where G is a directed graph, and $\{s, t\} \subseteq V(G)$. The yes instances are those in which there is a path from s to t in G. Note that, if $V(G) = \{1, 2, \ldots, n\}$, the instance (G, s, t) is of length $c \cdot n^2$, for some positive constant c, assuming that we encode G as an $n \times n$ matrix of bits in which the $(i, j)^{th}$ bit is a 1 if and only if the arc (i, j) is in $A(G)$. (Note "arc" instead of "edge" and $A(G)$ instead of $E(G)$, in order to emphasize that G is a *directed* graph. PATH is a totally different, easier problem for undirected graphs.) So we seek a nondeterministic algorithm that decides PATH in space $O(\log(c \cdot n^2)) = O(\log n)$. Here is one such algorithm:

PATH(G, s, t)
{
  $i \leftarrow 0$;
  $u \leftarrow s$;
  WHILE$(i \le n)$
  {
    IF $(u = t)$ THEN OUPUT(ACCEPT) AND HALT;
    GUESS $u' \in V(G)$;

IF $((u,u') \in A(G))$ THEN $u \leftarrow u'$;
$i \leftarrow i+1$;
}
OUTPUT(REJECT) AND HALT;
}

Things to notice about this algorithm:

- If there is a path from s to t, then there must be one of length less than or equal to n, because there are only n nodes in G.
- We cannot simply guess a path of length at most n in one fell swoop, because that would require $\Omega(n \log n)$ bits of workspace. Thus, we guess one node at a time and verify that all of the requisite arcs are there.
- It is clear that the values of the variables i, u, and u' require $O(\log n)$ workspace. Not as apparent, but still not hard, is that the bit on the input tape that tells us whether $(u,u') \in A(G)$ can be read in space $O(\log n)$ using a counter.

∎

Claim 2 *Every set in NL is logspace-reducible to PATH.*

Proof. Let S be a set in NL and M be a nondeterministic logspace machine that recognizes S. We must exhibit a logspace reduction f from S to PATH, *i.e.*, an implicitly logspace-computable f such that $x \in S$ if and only if $f(x) \in$ PATH.

The directed graph G in $f(x)$ is the configuration graph $G^{M,x}$; the nodes s and t in $f(x)$ are the START and ACCEPT configurations $C^{M,x}_{\text{START}}$ and $C^{M,x}_{\text{ACCEPT}}$ in $V(G^{M,x})$. By definition of "configuration graph," we have that $x \in S$ if and only if $f(x) \in$ PATH; so it remains to prove that f is logspace-computable.

There is a constant c that depends on M but not on x such that the number of bits required to encode any configuration $C \in V(G^{M,x})$ is $c \log n$, where $n = |x|$. The number $|V(G^{M,x})|$ of configurations is $2^{c \log n} = n^c$, and $G^{M,x}$ can be written down explicitly (as an adjacency matrix) using n^{2c} bits. Therefore, the length $|f(x)|$ of the target instance $(G^{M,x}, C^{M,x}_{\text{START}}, C^{M,x}_{\text{ACCEPT}})$ is polynomial in $|x|$ (specifically, $n^{2c} + 2c \log n$, where $|x| = n$), and we can clearly determine in space logarithmic in $|x|$ whether $i \leq f(|x|)$.

It remains to show that each bit in $f(x)$ can be computed in space logarithmic in $|x|$. The configurations $C^{M,x}_{\text{START}}$ and $C^{M,x}_{\text{ACCEPT}}$ can each be written down explicitly in space $c \log n$, where $n = |x|$; so it is clear how to determine whether each of the last $2c \log n$ bits of $f(x)$ is a 1 in space $O(\log n)$. To determine the $(C,D)^{th}$ bit of the adjacency matrix in $f(x)$, we use the following logspace procedure: Write C on a work tape, $C' = \delta_0(C)$ on a second work tape, and $C'' = \delta_1(C)$ on a third work tape, where δ_0 and δ_1 are the transition functions of M; then output 1 if and only if $D = C'$ or $D = C''$. Say M has k writable tapes and (logarithmic) space complexity s. Recall that

$$C = (q, P_0, P_1, \ldots, P_k, \gamma_{1,1}, \gamma_{1,2}, \ldots, \gamma_{1,s}, \ldots, \gamma_{k,1}, \gamma_{k,2}, \ldots, \gamma_{k,s}),$$

where q is an element of the state set of M, P_0 is the position of M's input tape head, P_w is the position of the w^{th} writable-tape head in M, $1 \leq w \leq k$, and $\gamma_{w,j} \in \Gamma$ is the symbol in the j^{th} cell of the w^{th} writable tape. Similarly, let

$$C' = (q', P'_0, P'_1, \ldots, P'_k, \gamma'_{1,1}, \gamma'_{1,2}, \ldots, \gamma'_{1,s}, \ldots, \gamma'_{k,1}, \gamma'_{k,2}, \ldots, \gamma'_{k,s})$$

and

$$C'' = (q'', P''_0, P''_1, \ldots, P''_k, \gamma''_{1,1}, \gamma''_{1,2}, \ldots, \gamma''_{1,s}, \ldots, \gamma''_{k,1}, \gamma''_{k,2}, \ldots, \gamma''_{k,s}).$$

Computation of C' (resp. C'') can be done in space $O(|C| + |C'|) = O(\log n)$ (resp. $O(|C| + |C''|) = O(\log n)$), because each of its components can be looked up in a (constant-sized) table that specifies the transition function δ_0 (resp. δ_1). ∎