# NSPACE and Reductions

CS254
Chris Pollett
Oct 9, 2006.

# Outline

- Savitch’s Theorem
- Immerman-Szelepscenyi
- Reductions

# Savitch's Theorem

- Last day, we showed L $\subseteq$ NL $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE and from the space hierarchy theorem we know L $\neq$ PSPACE from the space hierarchy theorem.
- So one of the in between inclusions must be strict.
- Could it be possible for two of these classes to be the same? For instance, how likely is it that P=NP?
- Have collapses ever been proven before?
- The answer is yes. One example, follows from Savitch's Theorem

**Thm. (Savitch)** Reachability is in SPACE($\log^2$ n).

# Proof of Savitch's Theorem

- Let G be a graph with n nodes, let x and y be nodes of G, and let $i \geq 0$. We say that the predicate PATH(x,y,i) holds if: there is a path from x to y in G of length at most $2^i$.
- We can compute reachability of any two nodes in G if we can determine if PATH(x,y, [log n]) holds.
- We are going to build a TM with two tapes beside the input tape.
- We will assume the input tape has the adjacency matrix for G.
- We will assume the first tape has already copied onto it, the nodes x, y and the integer i in binary. This tape will typically store several triples of which (x,y,i) will be the leftmost.
- The other tape is used as scratch space…

# Proof of Savitch's Theorem cont'd

- We start with i=0 and try to compute PATH(x, y, 0). This involves checking whether x=y or whether then is a single edge in the adjacency matrix between x and y. Both of which can be done in log space.
- For i≥0 we can compute PATH(x,y,i) with the recursive algorithm:
  - For all nodes z test whether PATH(x, z, i-1) and PATH(y,z, i-1).
- We can generate each z in turn reusing space and perform the test. As there are n nodes it takes at [log n] bits to store z. Once a z is generated we add (x,z, i-1) to the main work tape. If there is a PATH(x,z,i), we replace (x,z,i-1) on the work tape with (y,z,i-1) and compute PATH(y,z,i-1). If there is no such path we move on to the next z.
- Notice at any given time in computing PATH(x,y, [log n]) we have at most log n many triples on the work tape and that each triple takes at most 3 [log n] to store. So this algorithm is in SPACE($\log^2 n$) as desired.

# A Corollary

**Corollary.** NSPACE(f(n)) $\subseteq$ SPACE($[f(n)]^2$ ) for any proper complexity function f(n)$\geq$n.

**Proof.** Recall from last day that the configuration graph on inputs of length n for a machine M whose language is in NSPACE(f(n)) has size at most $k^{\log n + f(n)}$ for some k. Whether an accepting configuration is reachable from the start configuration is an instance of reachability so can be solved in SPACE($[\log (k^{\log n + f(n)})]^2$ ) = SPACE( $[f(n)]^2$ ).

(By space compression the base of the log can be chosen to be k.)

**Corollary.** NPSPACE =PSPACE.

# Is L=NL?

- The square factor in the time bound for reachability prevents us from showing that L=NL.
- Nevertheless, Immerman-Szelepscenyi were able to show:

**Theorem.** Given a graph G and a node x, the number of nodes reachable from x in G can be computed by a nondeterminstic TM within log n space.

# Proof of Immerman-Szelepscenyi

The algorithm has four nested loops:

- The outer loop computes iteratively |S(1)|, |S(2)|, …, |S(n-1)| where S(k) is the set of nodes in G that can be reached from x by paths of length k. Thus, |S(n-1)| is the number we want to compute.
- The second loop uses |S(k-1)| to compute |S(k)|. Let $l$ be the current count. At the start of this loop $l$=0. The second loop checks for each node reusing space u=1,..,n if u is in S(k) and if it is increments $l=l+1$.
- The third loop is used by the second loop in determining if u is in S(k). It checks each node v reusing space if it is in S(k-1). Let m be a counter of the number of such v so far. To see if u is in S(k) we check whether u=v or there is an edge from v to u, in which case we report true. If we reach the last v and m < |S(k-1)|, then we reply no on this nondeterministic computation branch. If m=|S(k-1)|, then we say u is not in S(k).
- The fourth loop is used to say whether v is in S(k-1). Looping given x we nondeterministically guess k-1 nodes $u_i$, in turn, for each pair checking there is an edge between them. We check the last one is v.

# Another Corollary

If $f \geq \log n$ is a proper complexity function, then NSPACE(f(n)) = coNSPACE(f(n)).

**Proof.** Suppose L is in NSPACE(f(n)), decided by some M. We will show that $\overline{L}$ is decided by some nondeterministic machine $\overline{M}$. On input x, $\overline{M}$ runs the algorithm of the last theorem on the configuration graph of M on x. If while running this algorithm $\overline{M}$ discovers an accepting computation of u is in S(k), then it halts and rejects (it is decdiing the complement). Otherwise, if |S(n-1)| is computed and no accepting computation has been encountered, $\overline{M}$ accepts.