CSE531: Computational Complexity Theory January 24, 2012

Lecture 5 Space

Lecturer: Anup Rao

1 L

The smallest space class that makes sense is $\mathbf{L} = \mathsf{DSPACE}(\log n)$. As usual the non-deterministic version of this class is when the machine can make non-deterministic choices, and is called $\mathbf{NL} = \mathsf{NSPACE}(\log n)$. There is a subtle issue about the definition of **NL**: if we allow the machine to remember the non-deterministic choices that it made for free (for example by giving it access to a guess tape that it can read from), then the power of the class changes significantly. This issue will be addressed in your homework.

Another interesting class is $\mathbf{PSPACE} = \bigcup_c \mathsf{DSPACE}(n^c)$. The corresponding non-deterministic class is actually equal to **PSPACE**, as we shall prove below.

A very useful trick to remember when composing space bounded computations is the following: If it takes space $s_1(n)$ to compute f and space $s_2(n)$ to compute g, then one might be able to compute the composition $f(g(x))$ in space $s_1(n) + s_2(n)$. The idea is that in the computation of f, every time we need to lookup an output symbol of $g(x)$, we can recompute it. Thus, as long as $s_1(n), s_2(n)$ are enough to store pointers into the output locations, we actually only need to sum the spaces to compute the composition.

Unlike the case of time bounded computations, in the space bounded world we actually have non-trivial results bounding the power of non-determinism. The main result is Savitch's theorem:

Theorem 1 (Savitch)**.** *For any space-constructible $s : \mathbb{N} \to \mathbb{N}$,*

$$\mathsf{NSPACE}(s(n)) \subseteq \mathsf{DSPACE}(s(n)^2).$$

Proof Given any $f \in \mathsf{NSPACE}(s(n))$, and an input x, we can construct its configuration graph. Then $f(x) = 1$ if and only if this graph has a path from the start configuration to the accept configuration. We can assume that there is a single accept configuration by requiring that the machine erase its work tape before outputting 1. Indeed, this graph can be computed in space $s(n)$, since $s(n) \geq \log n$.

Thus to complete the proof, it is enough to show that checking whether two vertices are connected in a directed graph of size $2^{s(n)}$ requires space $s(n)^2$.

We shall give a recursive algorithm that can compute the values $A(u, v, i)$ as defined below:

Define

$$A(u,v,i) = \begin{cases} 1 & \text{if there is a path from } u \text{ to } v \text{ of length } 2^i, \\ 0 & \text{else.} \end{cases}$$

Note that $A(u, v, i) = 1$ if and only if $\exists z$ such that $A(u, z, i-1) = 1$ and $A(z, v, i-1) = 1$. Thus, we can compute:

$A(u, v, i)$

- For all z, recursively compute $A(u, z, i-1)$ and $A(z, v, i-1)$, and output 1 if both computations result in 1.
- Otherwise output 0.

If the size of the graph is 2^s, there are $s+1$ recursive calls, where $A(u, v, 0)$ can be computed trivially by looking up the corresponding bit in the input. In each recursive call, the algorithm needs to store only the vertices u, v, z, which takes $O(s)$ space. Thus the total space used is $O(s^2)$. ■

Corollary 2. $\mathbf{NL} \subseteq \mathbf{L}^2 = \mathsf{DSPACE}(\log^2 n)$.

Corollary 3. $\mathbf{PSPACE} = \mathbf{NPSPACE}$.

The above theorem shows that the problem of *connectivity* is **NL**-complete. In recent times, progress have been made on showing that undirected connectivity (connectivity in undireted graphs) can be computed in log space:

Theorem 4 (Reingold)**.** *Given an undirected graph of size n, there is an algorithm in $\mathsf{DSPACE}(\log n)$ that can compute whether or not any two vertices are connected.*

Given any set of boolean functions S, we write coS to denote the set $\{f : 1 - f \in S\}$. Thus $co\mathbf{NP}$ is the set of functions for which there is an efficiently verifiable proof that $f(x) = 0$.

Theorem 5. *For space constructible $s(n)$, $\mathsf{NSPACE}(s(n)) = co\mathsf{NSPACE}(s(n))$.*

Proof As usual we focus on the configuration graph. To prove the theorem, it will be enough to be able to verify that there is *no* path from two vertices u, v in the graph, in $s(n)$ space. This would show that if $f(x) = 1$ can be certified in space $s(n)$, then $f(x) = 0$ can also be certified in space $s(n)$. The other direction is completely symmetric.

We shall prove how to do this by designing a sequence of algorithms. Let C_i denote the set of vertices that are reachable from u in i steps. Suppose the graph is of size at most 2^s.

Claim 6. *Given any vertex v and a number $i \leq 2^s$, there is a non-deterministic algorithm that can verify that $v \in C_i$ using space $O(s)$.*

The algorithm simply guesses a path from u to v and checks that the path is a valid path of the graph by checking each edge in order.

Claim 7. *Given the size of $|C_{i-1}| = c$, and a vertex v, there is a non-deterministic algorithm that can verify that $v \notin C_i$ using space $O(s)$.*

Since the algorithm is given the size of $C_{i-1}i$, the algorithm guesses each of the vertices of C_{i-1} in increasing order, and for each one, it checks that the vertex is different from the last vertex that was guessed, and then uses Claim 6 to verify that the vertex is indeed a member of C_{i-1}. It also makes sure that the given vertex is not v and not a neighbor of v. It maintains a count of all the number of vertices guessed and checks that $|C_{i-1}|$ vertices are given.

Finally, we argue that given the size of C_{i-1}, we can certify the size of c.

Claim 8. *Given the size of $|C_{i-1}| = c'$, there is a non-deterministic algorithm that can certify that $|C_i| = c$ using space $O(s)$.*

For each vertex v of the graph (in increasing order), the algorithm uses Claims 6 and 7 to check whether $v \in C_i$ or $v \notin C_i$, and it maintains a count of the number of vertices in C_i.

Thus, we obtain an algorithm that can verify that $v \notin C_n$ in $O(\log n)$ space. ■