Lecture 16:
PSPACE

Valentine Kabanets

November 18, 2016

1 Determinism vs. Nondeterminism

We have looked at the "P vs. NP" question, which is basically a question whether nondeterminism can be efficiently removed, without a huge (exponential) increase in the running time. Similar questions can be asked also in the framework of space (memory) complexity. Surprisingly, nondeterminism turns out not to help in the case of space-bounded computation!

2 Space classes

We extend our usual definition of a TM to have a *read-only input tape*, a *read/write work tape*, and possibly a *write-only output tape*. When counting the memory/space used by a TM, we only count the number of tape cell on the work-tape(s) scanned by the TM over the course of its computation on an input x. Taking the maximum space used over all inputs x of size n yields a (worst-case) space complexity of this TM. Note that separating the input tape from the work tape makes it possible to talk about TMs that use *sublinear* space (e.g., space $O(\log n)$).

For a "nice" space measure function $s(n)$, define the class $SPACE(s)$ to be the class of all languages that can be decided by a DTM in space at most $O(s(n))$ on inputs of size n. Similarly, $NSPACE(s)$ is the class of languages decided by a NTM in space $O(s(n))$, where each nondeterministic computation of the NTM uses at most $O(s(n))$ space.

Of special interest to us will be the following classes:

- $\mathsf{PSPACE} = \cup_{c \geq 0} \mathsf{SPACE}(n^c)$,
- $\mathsf{NPSPACE} = \cup_{c \geq 0} \mathsf{NSPACE}(n^c)$,
- $\mathsf{L} = \mathsf{SPACE}(\log n)$,
- $\mathsf{NL} = \mathsf{NSPACE}(\log n)$.

We'll prove the following amazing results:

1. $\mathsf{NPSPACE} = \mathsf{PSPACE}$ ($\mathsf{NP} = \mathsf{P}$ in the space setting), and
2. $\mathsf{NL} = \mathsf{coNL}$ (NL is closed under complement).

3 NPSPACE = PSPACE

Given a $\mathsf{NSPACE}(s(n))$ TM M and an input x of length n, we define the *configuration graph* G_x whose vertices are all possible configurations of M on x, and an edge connects two configurations u and v iff M can move from u to v in one step. (Recall that M is *nondeterministic*, and so u can have many outgoing edges.)

Each configuration can be described using $O(\log n)$ bits for the position on the input tape, plus $O(s(n))$ for the work-tape contents (including the position on the work-tape), plus $O(1)$ bits for the current state of M. Overall, we use at most $O(s(n))$ bits for any $s(n) \geq \log n$. (In the sequel, we only consider space $s(n)$ TMs for $s(n) \geq \log n$.) It follows that our configuration graph G_x of a space $s(n)$ TM M on an n-bit input x has at most $N \leq 2^{O(s(n))}$ vertices.

We claim that M accepts x iff there is a path from the start configuration to the accept configuration in the configuration graph G_x. Note that if there is such a path, then there is a simple path (without any loops), and the length of a simple path is at most N. Thus, the question "Does M accept x?" is equivalent to "Does G_x have a path from start to accept of length at most N?"

Theorem 1 (Savitch)**.** *For any nice space bound $s(n) \geq \log n$, $\mathsf{NSPACE}(s) \subseteq \mathsf{SPACE}(s^2)$.*

Proof. Given a space $s(n)$ NTM M and an input x of size n, we consider the configuration graph $G_x = (V, E)$ on $N \leq 2^{O(s(n))}$ nodes. We design algorithm $Path(x, y, i)$ which accepts iff there is a path from x to y of length at most 2^i. Note that *accept* is reachable from *start* iff $Path(start, accept, \log N)$ accepts.

Algo $Path(x, y, i)$
    if $i = 0$ **then**
        Accept iff $x = y$ OR $(x, y) \in E$
    end if
    for every $z \in V$
        if $Path(x, z, i-1)$ accepts AND
          $Path(z, y, i-1)$ 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 $Path$ 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 $O(\log^2 N) = O((s(n))^2)$. □

Remark 1. *Using $O(N)$ space, we can check connectivity of a digraph on N nodes in deterministic $\mathsf{poly}(N)$ time (using BFS, for example). Savitch's algorithm uses $O(\log^2 N)$ space only, but it runs in time $2^{O(\log^2 N)} = N^{O(\log N)}$, which is* not polynomial *in N.*

1. *Is it possible to have a sublinear- space algorithm that also runs in polynomial time?*
2. *Is it possible to improve upon Savitch's algorithm to get an algorithm running in space $O(\log^{2-\epsilon} N)$, for some constant $\epsilon > 0$ (e.g., $O(\log^{1.99} N)$)?*