Notes on Complexity Theory Last updated: September, 2007

Lecture 5

Jonathan Katz

1 Lower Bounds via Crossing Sequence Arguments

We show two lower bounds that are proved using related ideas. The results are not necessarily so interesting in their own right, but the proofs are interesting as rare examples in complexity theory when we can prove a direct lower bound on the power of a (uniform) complexity class!

1.1 One-Tape Turing Machines

We first show a language L requiring quadratic time to decide on a one-tape Turing machine. Besides being interesting in its own right, it separates the power of one-tape and two-tape machines (since L can be decided in linear time on a two-tape Turing machine).

For a finite string x over any alphabet, let $\bar{x}$ be the reverse of x. Taking $\Sigma = \{0, 1, \#\}$, define the language of palindromes as:

$$L \stackrel{\text{def}}{=} \{x\bar{x} \mid x \in \Sigma^*\}.$$

Theorem 1 *Deciding L requires time $\Omega(n^2)$ on a one-tape Turing machine.*

Proof Let M be a one-tape Turing machine deciding L. For each n which is a multiple of four, consider the following subset of L:

$$L_n \stackrel{\text{def}}{=} \{x \#^{\frac{n}{2}} \bar{x} \mid x \in \{0,1\}^{\frac{n}{4}}\}.$$

For any $x \in L_n$ and position $i \in \{1, \ldots, n\}$ on the tape of M, let $C_i(x)$, the *crossing sequence on x at position i*, denote the ordered sequence of states that M is in as the head of M crosses over the line between the ith and $(i+1)$th cells (in either direction). Define

$$C(x) \stackrel{\text{def}}{=} \left\{ C_i(x) \mid \frac{n}{4} \leq i \leq \frac{3n}{4} \right\}.$$

It is easy to verify that $C(x)$ cannot be empty for any $x \in L_n$.

The main observation is that if $x, y \in L_n$ and $x \neq y$, then $C(x) \cap C(y) = \emptyset$. The reason is as follows: say $C(x) \cap C(y) \neq \emptyset$. Then there are some i, j with $\frac{n}{4} \leq i, j \leq \frac{3n}{4}$ such that $C_i(x) = C_j(y)$. Let x' be the prefix of x of length i, and let y' be the suffix of y of length $n - j$. If we follow the computation of M on the input string $x'y'$, we see that M will accept. (This is not obvious, and convincing yourself of this requires some thought.) However, $x'y' \notin L$ and so this contradicts the fact that M decides L.

Let t_x denote the length of the *shortest* crossing sequence in $C(x)$, and let $t_{\max}(n) = \max_{x \in L_n}\{t_x\}$. We show that $t_{\max}(n) = \Omega(n)$. Since $C(x)$ contains $\Omega(|x|)$ crossing sequences,

and each element in each crossing sequence corresponds to a move of M, this implies that M runs in time $\Omega(n^2)$.[1]

The number of crossing sequences of length $\leq t$ is:

$$\sum_{i=0}^{t} |Q_M|^i = \frac{|Q_M|^{t+1} - 1}{|Q_M| - 1}.$$

Since no crossing sequence can appear in $C(x)$ and $C(y)$ for distinct $x, y \in L_n$, but there are $2^{\frac{n}{4}}$ distinct strings in L_n, we must have

$$\frac{|Q_M|^{t_{\max}(n)+1} - 1}{|Q_M| - 1} \geq 2^{\frac{n}{4}},$$

where Q_M is the set of states of M. Since $|Q_M|$ is constant, this implies $t_{\max} = \Omega(n)$. ■

1.2 Sublogarithmic Space

Next, we explore what can be done in sub-logarithmic space. We state the following two results without proof.

Theorem 2 *There exists a language L such that $L \in$ SPACE$(\log\log n)$ but $L \notin$ SPACE(1). That is,* SPACE(1) *is a proper subset of* SPACE$(\log\log n)$.

For proof, see [1, Lecture 4].

Theorem 3 SPACE(1) *is exactly the class of regular languages.*

It is not hard to show that regular languages can be decided with zero workspace (simply encode the states of the finite automaton as states of the Turing machine). It is also not too difficult to show that any constant-space Turing machine can be simulated by a 0-space Turing machine (simply include states for all possible strings that can be written on the work-tape). The difficult part is to show that any language decided by a 0-space Turing machine is regular; the difficulty is that the Turing machine may go back and forth across its input tape, while a finite automaton is allowed only one "pass" over its input. However, 0-space Turing machines are equivalent to so-called *two-way finite automata* and the latter are known to be equivalent to (regular) finite automata [4] (see [3, Section 2.6]).

What about classes below SPACE$(\log\log n)$? We show that all classes below this level collapse to SPACE(1).

Theorem 4 SPACE$(o(\log\log n)) =$ SPACE(1).

Proof Fix a machine M using space $s(n)$ on inputs of length n, with $s(n) = o(\log\log n)$. If M accepts only a finite number of strings (or, in particular, if M accepts nothing) we are done. Let a *semi-configuration* of M, during its computation on some input, consist

[1] Note: we do not prove that M runs in time $\Omega(n^2)$ for all $x \in L_n$, only that it runs in time $\Omega(n^2)$ for *at least one* $x \in L_n$.

of (1) the current state of M; (2) the symbol being scanned by the input head; (3) the contents of the work tape; and (4) the position of the work head on the work tape. (Note that, in contrast to a usual configuration, we do not include the position of the input head.) On input of length n, the number of possible semi-configurations of M is at most $N \stackrel{\text{def}}{=} |Q_M| \cdot (|\Sigma_M| + 1)^{s(n)+1} \cdot s(n) = 2^{O(s(n))}$.

For any x accepted by M of length $|x| = n$, and any $i \in [n]$, define the *crossing sequence on x at position i*, denoted $\mathcal{C}^i(x)$, to be the ordered sequence of semi-configurations of M whenever the input head is on the ith position of the input tape. The length of any such sequence is at most N: if it is longer than that, then some semi-configuration repeats (with the input head being at the same location) and so M would go into an infinite loop — contradicting the fact that M accepts x. The total number of possible crossing sequences is therefore at most

$$\sum_{\ell=0}^{N} N^\ell = \frac{N^{N+1}-1}{N-1} = O(N^N) = O\left(\left(2^{O(s(n))}\right)^{2^{O(s(n))}}\right) = O\left(2^{2^{O(s(n))}}\right) = o(n),$$

using the fact that $s(n) = o(\log \log n)$.

Let n_0 be such that for all $n > n_0$ the number of possible crossing sequences on inputs of length n is less than $n/3$; such an n_0 must exist because the number of possible crossing sequences in $o(n)$. Let $n_1 \geq n_0$ be such that $s(n) < s(n_1)$ for all $n < n_1$; if such an n_1 does not exist then the function s is upper-bounded by a constant (and we are done). We show that M never uses more than space $s(n_1)$ on strings of *any* length, and hence M is in fact a constant-space machine.

Assume the contrary. Let x be a string of minimum length satisfying: (1) x is accepted by M; (2) $|x| \geq n_1$; and (3) the space used by $M(x)$ is greater than $s(n_1)$. (Recall our assumption that M accepts an infinite number of strings.) Let $|x| = n$. For any $i \in [n]$, the number of possible crossing sequences on x at position i is less than $n/3$, and so there must exist at least three positions $i, j, k \in [n]$ (with $i < j < k$) on the input tape such that

$$\mathcal{C}^i(x) = \mathcal{C}^j(x) = \mathcal{C}^k(x).$$

Write x as $\alpha a \beta a \gamma a \delta$ where the positions with symbol a correspond to positions i, j, k (note that these positions must share the same input symbol since that is part of a semi-configuration and the input tape is read-only). Let $\mathcal{C}^i(x) = \mathcal{C}^j(x) = \mathcal{C}^i_1, \ldots, \mathcal{C}^i_\ell$ for some ℓ, and recall that all the semi-configurations in this sequence must be distinct (since otherwise M goes into an infinite loop). Call a semi-configuration *right-moving* if M moves its input head to the right on this semi-configuration, and *left-moving* otherwise. Now, look at the execution of M on input $x' \stackrel{\text{def}}{=} \alpha a \gamma a \delta$. The executions of $M(x)$ and $M(x')$ are identical until they come to the first right-moving semi-configuration in the sequence $\mathcal{C}^i(x)$; say this is semi-configuration $\mathcal{C}^i_{r_1}$. The execution of $M(x')$ after this point is now identical to the execution of $M(x)$ beginning at semi-configuration $\mathcal{C}^j_{r_1}$ until $M(x')$ comes to the next left-moving semi-configuration following $\mathcal{C}^j_{r_1}$; call it $\mathcal{C}^j_{r_2}$. The execution of $M(x')$ after this point is now identical to the execution of $M(x)$ beginning at semi-configuration $\mathcal{C}^i_{r_2}$, etc. Things continue in this way until $M(x')$ reaches semi-configuration $\mathcal{C}^i_\ell = \mathcal{C}^j_\ell$ (since $r_1 < r_2 < \cdots$ this eventually happens) and then the execution of $M(x')$ will be identical to the execution of

$M(x)$ after this point. In particular, $M(x')$ accepts. A similar argument shows that $M(x'')$ accepts, where we define $x'' \stackrel{\text{def}}{=} \alpha a \beta a \delta$.

Let s_w denote the maximum number of work cells used by M on input the string w. If s_x work cells are being used by $M(x)$ when its input head is within the substring αa or the substring δ, then $s_{x'}, s_{x'} \geq s_x$. If s_x work cells are being used by $M(x)$ when its input head is within the substring γa then $s_{x'} \geq s_x$; similarly, if s_x work cells are being used by $M(x)$ when its input head is within the substring βa then $s_{x''} \geq s_x$. In any case, then, one of the strings x', x'' (call it $\tilde{x}$) is a shorter string than x which is accepted by M, and for which $M(\tilde{x})$ uses space at least $s_x \geq s(n_1)$. By choice of n_1, we have $|\tilde{x}| \geq n_1$. But this contradicts our choice of x as the *shortest* string satisfying these properties. ■

Bibliographic Notes

Section 1.1 is based on [2, Lecture 1]. Section 1.2 is adapted from [1, Lecture 4] and [2, Lecture 1].

References

[1] O. Goldreich. Lecture notes for *Introduction to Complexity Theory*, 1999. Available at http://www.wisdom.weizmann.ac.il/~oded/cc99.html.

[2] D. Kozen. Lecture notes for *CS682: Theory of Computation*, Spring 2004. Available at http://www.cs.cornell.edu/Courses/cs682/2004sp/.

[3] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[4] J.C. Shepherdson. The Reduction of Two-Way Automata to One-Way Automata. *IBM Res. and Dev.* 3, pp. 198–200 (1959).