Notes on Complexity Theory Last updated: September, 2011

Lecture 4

Jonathan Katz

1 Diagonalization

In this lecture and the next one, we discuss two types of results that are related by the technique used in their proofs. Both kinds of results are also fundamental in their own right.

The common proof technique is called *diagonalization*. It is somewhat difficult to formally define the term, but roughly the idea is that we want to show the existence of some language L (with certain properties) that cannot be decided by any Turing machine within some set $S = \{M_1, \ldots\}$.[1] We do so by starting with some L_0 (with the property we want) and then, for $i = 1, \ldots$ changing L_{i-1} to L_i such that none of $M_1, \ldots, M_i$ decide L_i. Of course part of the difficulty is to make sure that L_i has the property we want also.

Actually, one can also prove the existence of an undecidable language using this technique (though not quite as explicitly as stated above). Consider an enumeration $x_1, \ldots$ of all binary strings, and an enumeration $M_1, \ldots$ of all Turing machines.[2] Define L as follows: $x_i \notin L$ iff $M_i(x_i) = 1$. (A picture really helps here. For those who have seen it before, this is exactly analogous to the proof that there is no bijection from the integers to the reals. In fact, that gives a 1-line proof of the existence of undecidable languages: the set of languages is uncountable, while the set of Turing machines is countable.) Say some machine M decides L, and let i be such that $M = M_i$. But then consider x_i: if $M(x_i) = 1$ then $x_i \notin L$ and so M is wrong; if $M(x_i)$ rejects or doesn't halt then $x_i \in L$ and M is again wrong!

1.1 Hierarchy Theorems

It is natural to wonder whether additional resources actually give additional power. We show that this is the case (at least to a certain extent) for space and time. We first give a definitions of "well-behaved" functions.

Definition 1 *A function $f : \mathbb{N} \to \mathbb{N}$ is* space constructible *if it is non-decreasing and there exists a Turing machine that on input 1^n outputs the binary representation of $f(n)$ using $O(f(n))$ space. Note that if f is space constructible, then there exists a Turing machine that on input 1^n marks off exactly $f(n)$ cells on its work tapes (say, using a special symbol) without ever exceeding $O(f(n))$ space.*

For space bounds, it is often assumed that $f(n) \geq \log n$ as well. We will make this assumption throughout this class, unless explicitly stated otherwise. Note that non-trivial algorithms using sub-logarithmic space do exist; in particular, $\text{SPACE}(1)$ is a proper subset of $\text{SPACE}(\log \log n)$ (see [2, Lecture 4]). Nevertheless, sub-logarithmic space causes difficulties because there is not even enough space to store a counter indicating the position of the input-tape head.

[1] Note that the set of all Turing machines is countably infinite, and so S is countable.

[2] An efficiently computable enumeration is obtained by letting M_i denote the Turing machine represented by the binary representation of i.

Definition 2 *A function $f : \mathbb{N} \to \mathbb{N}$ with $f(n) \geq n$ for all n is* time constructible *if it is non-decreasing and there exists a Turing machine that on input 1^n outputs the binary representation of $f(n)$ in $O(f(n))$ steps. Note that if f is time constructible, then there exists a Turing machine that on input 1^n runs for $O(f(n))$ steps and then halts.*

All functions you would "normally encounter" are space and time constructible; functions that aren't are specifically constructed counterexamples.

We first show that more space gives more power.

Theorem 1 (Space hierarchy theorem) *Let $G(n) \geq \log n$ be space constructible, and $g(n) = o(G(n))$. Then* $\text{SPACE}(g(n))$ *is a proper subset of* $\text{SPACE}(G(n))$.

Proof We show the existence of a language L such that $L \in \text{SPACE}(G(n))$ but $L \notin \text{SPACE}(g(n))$. We define L by describing a Turing machine M_L, using space $O(G(n))$, that decides it. M_L does the following on input $w = (M, y)$ of length $|w| = n$:

1. Run $M(w)$ with at most $G(n)$ space and for at most $2^{2G(n)}$ steps (these bounds are imposed on M), using space at most $3 \cdot G(n)$.
2. If $M(w)$ accepts within the given time and space bounds, then reject. Otherwise, accept.

In step 1, we can use the fact that G is space constructible to mark off exactly $G(n)$ tape cells for M to use. We can similarly mark off an additional $2G(n)$ cells to use as a counter for checking the number of steps M makes, and one last set of $G(n)$ cells to use for any remaining computation. By construction, M_L uses space $\tilde{G}(n) = 4 \cdot G(n)$.

We need to show that no machine using space $O(g(n))$ can decide L. Assume the contrary. Then there exists a machine M'_L deciding L and using space $\tilde{g}(n) = O(g(n))$. Choose k large enough so that $\tilde{g}(k) < G(k)$, so that[3] M'_L makes fewer than $2^{G(k)}$ steps on inputs of length k, and so that[4] the simulation of M'_L on inputs of length k can be performed in $G(k)$ space. Consider the input $w = (M'_L, 1^k)$. If we run $M_L(w)$ then (1) M_L has enough time and space to simulate the entire execution of $M'_L(w)$, and thus (2) $M_L(w)$ outputs the opposite of whatever $M'_L(w)$ outputs. We conclude that M_L and M'_L do not decide the same language. ■

We have a completely analogous time hierarchy theorem, though the result is quantitatively (slightly) weaker.

Theorem 2 (Time hierarchy theorem) *Let G be time constructible. If $g(n) \log g(n) = o(G(n))$, then* $\text{TIME}(g(n))$ *is a proper subset of* $\text{TIME}(G(n))$.

Proof The high-level structure of the proof is the same as in the proof of the previous theorem. We define L by giving a Turing machine M_L, using time $O(G(n))$, that decides it. M_L does the following on input $w = (M, y)$ of length $|w| = n$:

1. Run $M(w)$ using at most $c \cdot G(n)$ steps for some fixed constant c (see below).
2. If $M(w)$ accepts within the given time bound, then reject. Otherwise, accept.

[3] This condition is achievable because M'_L runs in time at most $O(n2^{O(g(n))})$ (something we will show later in the course), which is asymptotically smaller than $2^{2G(n)}$.

[4] This condition is achievable because universal simulation with constant space overhead is possible.

We can implement step 1 using the fact that G is time constructible: in alternating steps, simulate $M(w)$ and run a Turing machine that is guaranteed to stop within $O(G(n))$ steps; halt the entire computation once the latter machine halts. We thus have that M_L runs in time $O(G(n))$.

We need to show that no machine using time $O(g(n))$ can decide L. Assume the contrary. Then there exists a machine M'_L deciding L in time $O(g(n))$. Consider an input of the form $w = (M'_L, 1^k)$. If we run $M_L(w)$ then, for k large enough, M_L has enough time to simulate the entire execution of $M'_L(w)$. (Here we use the fact that universal simulation is possible with logarithmic overhead.) But then, for k large enough, $M_L(w)$ outputs the opposite of whatever $M'_L(w)$ outputs. We conclude that M_L and M'_L do not decide the same language. ■

The barrier to getting a tighter time hierarchy theorem is the logarithmic time overhead in universal simulation. If a better simulation were possible, we would obtain a tighter separation.

There is a non-deterministic time hierarchy as well; the details are more complicated because it is not possible to simply "flip" the output of a non-deterministic machine. (Do you see why?)

Theorem 3 (Non-deterministic time hierarchy theorem) *Let g, G be time constructible. If $g(n+1) = o(G(n))$, then* $\text{NTIME}(g(n))$ *is not contained in* $\text{NTIME}(G(n))$.

Proof We sketch a proof different from the one in the book. We will also rely on the fact that non-deterministic universal simulation with only *constant* time overhead is possible.

Once again, we define a language L by describing a machine that decides it. Consider the non-deterministic machine M_L that on input $w = (M, 1^k, y)$ of length $|w| = n$, where M is now interpreted as a non-deterministic Turing machine, does:

1. If $|y| < G(|M| + k)$ then run $M(M, 1^k, y0)$ and $M(M, 1^k, y1)$ for at most $G(n)$ steps (each), and accept iff they both accept.

2. If $|y| \geq G(|M|+k)$ then accept iff $M(M, 1^k, \varepsilon)$ rejects when using non-deterministic choices y. (Here ε denotes the empty string.) Note that if M does not halt on this computation path (e.g., because y is not long enough), then M_L rejects.

By what we have said before regarding universal simulation of non-deterministic Turing machines, and using the conditions of the theorem, M_L runs in time $O(G(n))$.

Say there exists a non-deterministic machine M'_L running in time $\tilde{g}(n) = O(g(n))$ and deciding L. Consider an input of the form $w = (M'_L, 1^k, \varepsilon)$ for k sufficiently large. We have

$$\begin{array}{rcll} w \in L & \Leftrightarrow & M'_L(M'_L, 1^k, 0) = M'_L(M'_L, 1^k, 1) = 1 & \text{Definition of } M_L \\ & \Leftrightarrow & (M'_L, 1^k, 0), (M'_L, 1^k, 1) \in L & M'_L \text{ decides } L \\ & \Leftrightarrow & M'_L(M'_L, 1^k, 00) = M'_L(M'_L, 1^k, 01) = 1 & \\ & & M'_L(M'_L, 1^k, 10) = M'_L(M'_L, 1^k, 11) = 1 & \text{Definition of } M_L \end{array}$$

Let t be the smallest integer with $t \geq G(|M'_L| + k)$. Continuing the above line of reasoning we get

$$\begin{array}{rcll} w \in L & \Leftrightarrow & \forall y \in \{0,1\}^t : M'_L(M'_L, 1^k, y) = 1 & \text{As above...} \\ & \Leftrightarrow & \forall y \in \{0,1\}^t : (M'_L, 1^k, y) \in L & M'_L \text{ decides } L \\ & \Leftrightarrow & \forall y \in \{0,1\}^t : M'_L(M'_L, 1^k, \varepsilon) \text{ rejects} & \\ & & \qquad \text{on computation path } y & \text{Definition of } M_L \\ & \Leftrightarrow & M'_L(M'_L, 1^k, \varepsilon) = 0 & \text{Definition of non-determinism} \\ & \Leftrightarrow & w \notin L & M'_L \text{ decides } L \end{array}$$

This is a contradiction, so we conclude that no such M'_L can exist. ■

Bibliographic Notes

The proof of the non-deterministic time hierarchy theorem given here is due to [1].

References

[1] L. Fortnow and R. Santhanam. Robust Simulations and Significant Separations. *ICALP (1)*, 2011. Available at `http://arxiv.org/abs/1012.2034`.

[2] O. Goldreich. Introduction to Complexity Theory (July 31, 1999).