CSC 2401 Introduction to Computer Complexity - Fall 2001
Lecture #4, Tuesday, September 25, 2001
Lecturer: Allan Borodin
Scribe: Karolina Buchner

Lecture #4

4.1 Introduction

Today we introduce the basic diagonalization result used to prove negative results in complexity theory. First, recall the Turing computational model:

$$\mathcal{M}_x = \langle Q, \Sigma, \Gamma, \delta, F_1, F_2, q_0 \rangle$$

Note that the Turing machine can itself be described as a string over some fixed alphabet. Here $x \in \{0,1\}^*$ is a representation, or encoding, of the Turing machine $\mathcal{M}_x$. However, x may or may not be a syntactically correct encoding of a Turing machine.

Convention: if $x \in \{0,1\}^*$ doesn't represent a Turing machine encoding, then say $\mathcal{M}_x$ is a "made-up" Turing machine which ignores input and immediately rejects, i.e. $\mathcal{L}(\mathcal{M}_x) = \emptyset$.

We now have a 1:1 correspondence between every Turing machine and every string.

4.2 The Universal Turing Machine

Theorem: A **Universal Turing Machine U** *is a machine such that $\mathcal{U}(\langle x, y \rangle) = \mathcal{M}_x(y)$, i.e. U simulates $\mathcal{M}_x$, encoded by x, on input y.*

One of many possible encoding conventions for the pairing $\langle x, y \rangle$ is as follows:

$$\langle x, y \rangle = 1^{|x|}\, 0\, x\, y$$

Here $1^{|x|}$ tells **U** the length of the encoding x; 0 delimits the string of 1's and the beginning of the machine encoding x; the input y follows the machine encoding.

Another possible encoding convention is:

Construct a mapping h(0) = 01 and h(1) = 10 and apply this mapping to the individual bits of $x = x_1 x_2 \cdots x_u$ and $y = y_1 y_2 \cdots y_v$. Then the encoding is constructed as follows:

$$\langle x, y \rangle = h(x_1)\, h(x_2) \cdots h(x_u)\, 0\, 0\, h(y_1)\, h(y_2) \cdots h(y_v)$$

The string "00" does not encode anything and so can be used as a delimiter between x and y.

4.3 Basic Diagonalization

Theorem: $L = \{\langle x, y\rangle | \mathcal{M}_x \text{ accepts } y\}$ *is semi-computable (or equivalently "recursively enumerable"), but* **not** *computable.*

If $\mathcal{M}_x$ halts on input y, we can verify this by running the universal simulator **U**. However, we never know if $\mathcal{M}_x$ will halt or not.

Proof by Diagonalization:

If L were computable, then L' would be computable where $L' = \{w | \langle w, w\rangle \notin L\}$.

Then $\exists$ z such that $\mathcal{L}(\mathcal{M}_z) = L'$ and $\mathcal{M}_z$ halts on every input.

$$z \in L' \iff \langle z, z\rangle \notin L \iff z \notin \mathcal{L}(\mathcal{M}_z) = L'.$$

Contradiction.

4.4 Refined (Complexity Oriented) Universal Turing Machine Theorem

Theorem: $\exists$ *a Universal Turing Machine U (say with 2 work tapes) such that if* $\mathcal{M}_x$ *runs in time t(n) and space s(n) (i.e. for sufficiently large n), then* $U(\langle x, y\rangle)$ *runs in time* $t' \leq c_x t(n) \log t(n)$ *and space* $s' \leq c_x s(n)$.

Note: the Universal Turing Machine U has 2 work tapes and a fixed work alphabet Γ.

4.5 Time and Space Constructible Functions

A non-decreasing time bound function $t : \mathbf{N} \rightarrow \mathbf{N}$ is time constructible if

$\exists$ a Turing machine $\mathcal{M}$: on input 1^n, outputs $1^{t(n)}$ within $\mathcal{O}$(t(n)) steps $\leq$ c$\cdot$t(n), for some constant c.

Respectively, a non-decreasing space bound function $s : \mathbf{N} \rightarrow \mathbf{N}$ is space constructible if

$\exists$ a Turing machine $\mathcal{M}$: on input 1^n, outputs $1^{s(n)}$ using $\leq$ c$\cdot$s(n) space, for some constant c.

Comment: Most non-decreasing functions you've ever seen are time and space constructible. E.g.: $n, n \log n, n^2, n^3, n^k, 2^n, n!, 2^{(2^n)}, \cdots$, *etc.*

Unless otherwise noted, assume any time or space bounds are time, and respectively space, constructible. For time, assume t(n) $\geq$ n. For space, assume s(n) $\geq \log n$.

4.6 Time Hierarchy Theorem

Theorem: Let $t_2(n)$ *be constructible and let* $t_1(n) \geq n$ *be such that*

$$\lim_{n \to \infty} \frac{t_1(n) \ \log \ t_1(n)}{t_2(n)} = 0$$

or alternatively (if we only want to show that L below exists)

$$\liminf_{n\to\infty} \frac{t_1(n)\ \log\ t_1(n)}{t_2(n)} \to 0$$

Then DTIME(t_1) $\subset$ DTIME(t_2) and so

$$\exists\, L\ \in\ DTIME(t_2) - DTIME(t_1).$$

Proof by Diagonalization:

Let U' be a Turing machine which operates as follows:

on input y = 1^l 0 x, U' simulates U($\langle x, y\rangle$) = $\mathcal{M}_x(y)$ for $t_2(|y|)$ steps.

Note that y is a padded version of x, where l is arbitrary, giving arbitrarily "copies" of x.

That is, we first compute $t_2(|y|)$, to be used as a counter, and then while we simulate U($\langle x, y\rangle$) we decrease the counter and ensure we don't simulate U($\langle x, y\rangle$) for more than $t_2(|y|)$ steps.

Then U'

- accepts y if the simulation rejects within $\le t_2(|y|)$ steps or simulation doesn't complete within $t_2(|y|)$ steps
- rejects if U($\langle x, y\rangle$) did accept in $\le t_2(|y|)$ steps

Time Hierarchy Theorem: L = $\mathcal{L}(U') \in$ DTIME($\mathcal{O}(t_2)$) - DTIME(t_1).

Why not in t_1?

Proof:

Suppose L $\in$ DTIME(t_1). L = $\mathcal{L}(\mathcal{M}_z)$ and $\mathcal{M}_z$ runs in time $t_1(n)$.

Therefore,

$$time_{\mathcal{M}_z}(y) \le t_1(|y|), \overset{\infty}{\forall}\, y$$

By the refined Universal Turing Machine theorem,

$$U(\langle z, y\rangle)\ runs\ in\ time\ \le\ c_z\, t_1(n)\ \log t_1(n)\ , \overset{\infty}{\forall}\ n = |y|.$$

Since $\lim_{n\to\infty}\ \frac{t_1(n)\ \log\ t_1(n)}{t_2(n)} = 0$, then

$$for\ \ any\ c_z,\ \overset{\infty}{\forall}\ n\, t_2(n)\ >\ c_z\, t_1(n)\ \log t_1(n).$$

Therefore, U' will complete its simulation within $t_2(n)$ steps when y is sufficiently large. $U'(y)$ will accept iff $\mathcal{M}_z(y)$ rejects.

The time hierarchy theorem is due to Hartmanis and Stearns (1963); Rabin (1960) proved a related result showing that for any computable function t, there was a computable language L such that for any Turing machine $\mathcal{M}$ recognizing L, $\mathcal{M}$ uses more than $t(n)$ time on almost all inputs.