6.841 Advanced Complexity	Feb. 10, 2003

Lecture 2

Lecturer: Madhu Sudan — *Scribe: Yael Tauman*

Today:

- Savitch's theorem
- Diagonalization
- Relativization
- Introduction to Alternations

1 Savitch's Theorem

Savitch's theorem considers the relationship between non-deterministic space and deterministic space.

Theorem 1 (Savitch) *For all constructible functions $s(n) \geq \lg(n)$, $NSPACE(s(n)) \subseteq SPACE(O(s(n)^2))$.*

The idea of the proof is first to take an NL complete language and prove that it is in $SPACE(O(\lg^2 n))$, and then to generalize this for every constructible function $s(n) \geq \lg n$. The NL complete language that we use is

$$PATH = \{(G, s, t, l) : \exists \text{ path of length} \leq l \text{ from } s \text{ to } t \text{ in the directed graph } G\}.$$

Proof The proof is carried out by proving the following two Lemmas.

Lemma 2 *$PATH \in SPACE(O(\lg^2 n))$.*

Lemma 3 *Lemma 2 suffice to prove Savitch's theorem.*

Proof-Sketch of Lemma 2: Consider the following deterministic algorithm for $PATH$. On input (G, s, t, l), the algorithm operates as follows:

1. Compute G^2.

   Recall the definition of G^2: The vertices in G^2 are the same as in G and for every pair of vertices (u, v), the edge (u, v) is in G^2 if and only if there exists a path of length ≤ 2 from u to v in G.

2. Output $PATH(G^2, s, t, \frac{l}{2})$.

Recall the following claim, given in the previous lecture.

Claim 4 *For any pair of languages L_1, L_2, if $L_1 \leq L_2$ and the reduction can be done using space s_1, and if $L_2 \in SPACE(s_2)$ then $L_1 \in SPACE(2s_1 + s_2)$.*[1]

Since the first step of the algorithm uses space $O(\lg n)$, the claim implies that the above algorithm uses space $O(\lg l \times \lg n) = O(\lg^2 n)$.

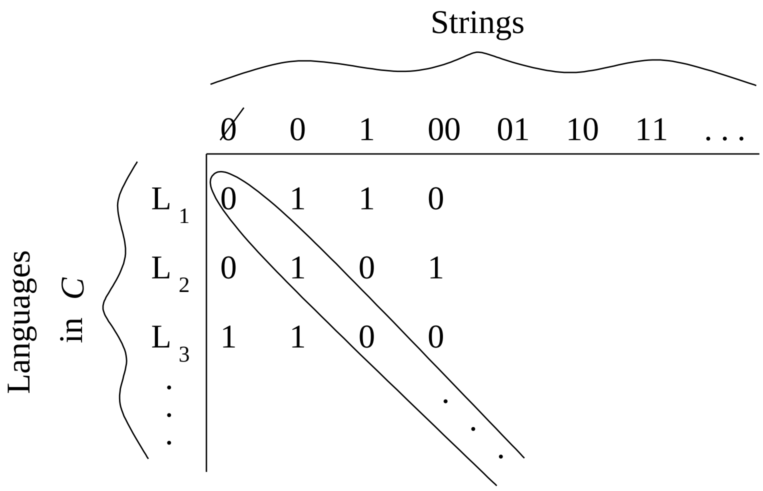
Proof-Sketch of Lemma 3: By a standard padding argument, one can see that if $s(n) \geq \lg n$ then we can always pad the input with enough zeros so that the algorithm using space $s(n)$, now uses space $\lg n'$, where $n' = 2^{s(n)}$ is the length of the input after it has been padded. Then this can be done in deterministic space $O(\lg^2 n') = O(s(n)^2)$. ■

In complexity theory, we get a kick out of findings relation between different complexity classes. More specifically, we would like to find reductions such as $NL = L$, but also, we would like to prove that certain reductions do not exist. Unfortunately, not much is known regarding the latter. The main tool used to achieve such impossibility results is the *diagonalization* method.

[1]This is a correction to the previous lecture where it was claimed that $L_1 \in SPACE(s_1 + s_2)$.

2 The Diagonalization Method

The diagonalization technique was introduced by Cantor in the 18'th Century. It is the most powerful tool to prove that certain reductions do not exist. For example, the time and space hierarchy theorems, mentioned in the previous lecture, are based on diagonalization. Other results that use diagonalization are Rice's theorem and Ladner's Theorem, which will be described towards the end of this lecture. The basic idea of the diagonalization technique is the following. Take a complexity class (a collection of languages) $\mathcal{C}$ and explicitly demonstrate $L \notin \mathcal{C}$ as follows.



Enumerate all languages in $\mathcal{C}$: $L_1, L_2, L_3 \ldots$ and enumerate all strings (which correspond to integers): $\emptyset, 0, 01, 10, 11, \ldots$ Correspond to every language L_k and every string w the value 1 if $w \in L_k$ and the value 0 otherwise. Demonstrate that $L \notin \mathcal{C}$ by proving that for every k, $k \in L$ iff $k \notin L_k$. That is, show that the language L corresponds to a flip of the main diagonal.

In many cases the diagonalization technique requires a special enumeration, namely, a *constructive* one. For example, let us use the diagonalization method to prove that $TIME(n^3) \subsetneq TIME(n)$.

1. Enumerate all linear-time Turing machines: $M_1, M_2, M_3, \ldots$
2. Define a Turing machine M as follows: $\forall k$, $M(k) \neq M_k(k)$.
3. Show that M runs in $TIME(n^3)$.

The question to be asked is how can we enumerate all linear-time Turning machines? One way is to enumerate *all* Turing machines rather than just the linear-time ones. However, then the Turing machine obtained from the diagonalization method will not be in computable in $TIME(n^3)$. We overcome this problem by enumerating *all* Turing machines: $M_1, M_2, \ldots$ and restricting their running time to some super-linear function, such as $n \lg n$. Thus, we get an enumeration of all Turing machines which run in $TIME(n \lg n)$: $M_1', M_2', \ldots$ Unfortunately, this will not necessarily give us all linear time Turing machines. The reason is that it may be the case, for example, that for infinitely many k's, machines M_k run in time $n + k$ (which is linear in n but exponential in the length of k). By restricting their running time to $n \lg n$, we truncate the running time of these (linear-time) Turing machines, since the running time of $M_k(k)$ is $\lg k + k$ whereas the running time of $M_k'(k)$ is only $\lg k \lg \lg k$.

We overcome this problem by enumerating the Turing machines in such a way that every machine appears infinitely often. This can be done, for example, as follows. Take any enumeration: $M_1, M_2, \ldots$, and transform it to the following *redundant* enumeration: $M_1, M_1, M_2, M_1, M_2, M_3, \ldots$. Now, when trying to diagonalize against M_k, since M_k appears infinitely often, it appears also in place $\geq 2^k$. The running time of $M_k(2^k)$ is $\lg(2^k)+k = 2k$ which is smaller than $\lg(2^k)\lg\lg(2^k) = k \lg k$, and thus its running time will not be truncated.

It remains to note that the Turing machine M, defined by the diagonalization method, is computable in $TIME(n^3)$.

The main question to be asked is:

Can diagonalization prove NP $\neq$ P*?*

In the 70's, Baker, Gill and Solovay gave negative evidence for this question. They introduced the notion of *relativization*, and used this notion to prove that, in some sense, "diagonalization cannot prove NP $\neq$ P."

3 Relativization

We use the following notations

1. P^O is the set of all languages accepted by deterministic polynomial-time oracle Turing machines with oracle access to O.

2. NP^O is the set of all languages accepted by non-deterministic polynomial-time oracle Turing machines with oracle access to O.

Baker, Gill and Solovay noticed the following observation.

Observation: If one can use the diagonalization technique to show that NP $\not\subseteq$ P, then one can also use the diagonalization technique to show that for every language O, $\mathrm{NP}^O \not\subseteq \mathrm{P}^O$.

The idea behind this observation is the following. If we can prove NP $\not\subseteq$ P using diagonalization, then it means that we can find a non-deterministic polynomial-time Turing machine N that can simulate every polynomial-time Turing machine M and negate the answer. We can augment these machines into oracle Turing machines and obtain similar results. In other words, N^O can simulate every polynomial-time oracle Turing machine M^O and negate the answer. Thus, a similar argument shows that $\mathrm{NP}^O \not\subseteq \mathrm{P}^O$.

Theorem 5 (BGS) *There exist oracles A and B such that*

1. $\mathrm{P}^A = \mathrm{NP}^A$*, and*

2. $\mathrm{P}^B \neq \mathrm{NP}^B$.

Note that the first part of the BGS Theorem together with the above observation imply that it is impossible to prove P $\neq$ NP using diagonalization. For completeness we prove both parts of the BGS Theorem.

Proof-Sketch The first part is straightforward. The idea is to take some language that is sufficiently powerful e.g. PSPACE-complete. Let A be TQBF. Clearly $\mathrm{P}^A \subseteq \mathrm{NP}^A$. For the other direction, as TQBF is PSPACE-complete, we have

$$\mathrm{NP}^A \subseteq \mathrm{NPSPACE} = \mathrm{PSPACE} \subseteq \mathrm{P}^A.$$

The middle step follows from Savitch's theorem. For the second part, the idea is that non-determinism gives us more powerful access to the oracle, allowing us to ask exponentially more questions than allowed by a deterministic Turing machine. For any language B, let

$$L(B) = \{x | \exists w : |x| = |w| \text{ and } B(w) = 1\}$$

One can easily observe that for every language B, $L(B) \in \mathrm{NP}^B$, because we can guess w the same length as x, and check if $B(w) = 1$. We want to show that there exists a langauge B for which $L(B) \notin \mathrm{P}^B$. This will be done by enumerating all Turing machines $M_1, M_2, \ldots$ and showing that there exists a language B such that for every k, $L(B) \neq L(M_k^B)$. The language B will be defined as follows. We denote by $M^?$ the Turing machine M with oracle access to an undetermined oracle. For every $k = 1, 2, \ldots$ simulate $M_k^?(0^k)$. Reply to all oracle questions with 0, unless a question has already been set. In this case answer according to the previous setting.

1. If $M_k^?(0^k) = YES$ the set $B \cap \{0,1\}^k = \emptyset$.
2. If $M_k^?(0^k) = NO$ then choose a string $y \in \{0,1\}^k$ that has not yet been set and set $y \in B$. Intuitively, the reason that there exists such a string $y \in \{0,1\}^k$ is that for every $i = 1, \ldots, k-1$, $M_i^?$ runs in polynomial-time and thus at most $poly(k)$ many strings in $\{0,1\}^k$ are set.

In both cases, $L(M_k^B) \neq L(B)$, because $M_k^B(0^n) = YES$ if and only if $0^n \notin L(B)$.

Note that the technique, which was used to prove that diagonalization doesn't work, is diagonalization. One can similarly prove that there exists an oracle B such that $\mathrm{NP}^B \neq \mathrm{coNP}^B$.

The diagonalization method was also useful in other contexts. It was used, for example, in Rice's Theorem and Ladner's Theorem.

Theorem 6 (Rice) *If* NP $\neq$ P *then determining whether* $L \in$ NP *is also in* P *is undecidable.*

Theorem 7 (Ladner, 1973) *If* P $\neq$ NP, *then for all* $k \geq 1$, *there are languages* $L_1, \ldots, L_k \in$ NP *such that*

$$L <_{\mathrm{P}} L_1 <_{\mathrm{P}} \cdots <_{\mathrm{P}} L_k <_{\mathrm{P}} L'$$

where $L \in$ P *and* L' *is* NPcomplete.

4 Introduction to Alterations

Relativization, which was used in the BGS Theorem, turned out to be a very interesting concept. Consider the following question: why do we use $TQBF$ (a PSPACE-complete) problem for the first part of BGS Theorem? Isn't an NP-complete problem sufficiently powerful? That is, isn't $\mathrm{NP}^{SAT} = \mathrm{P}^{SAT} = \mathrm{NP}$? To answer this question, we should answer if $\mathrm{NP}^{\mathrm{NP}} = \mathrm{NP}$? Note that if $\mathrm{NP} = \mathrm{NP}^{\mathrm{NP}}$ then it follows that $\mathrm{NP} = co\mathrm{NP}$. The reason is that $co\mathrm{NP} \subseteq \mathrm{NP}^{\mathrm{NP}}$. Thus, we tend to believe that $\mathrm{NP} \neq \mathrm{NP}^{\mathrm{NP}}$. Consider for example the following language:

MinDNF $= \{(\phi, k)$: where ϕ is a DNF formula and $\exists$ DNF formula ψ s.t. $|\psi| \leq k$ and ψ is equivalent to $\phi\}$.

This language captures what can be done in $\mathrm{NP}^{\mathrm{NP}}$.

Proposition 8 MinDNF *is in* $\mathrm{NP}^{\mathrm{NP}}$.

Proof We can construct a non-deterministic oracle Turing machine, which uses a SAT oracle, to solve MinDNF.

1. First guess ψ of length $\leq k$.
2. Ask SAT oracle if there exists an assignment x such that $\psi(x) \neq \phi(x)$.
3. Accept if oracle says No, otherwise reject.

■

This motivated Meyer and Stockmeyer to define a hierarchy of complexity classes starting with

$$\Sigma_2^P = \mathrm{NP}^{\mathrm{NP}}.$$

We will elaborate on this hierarchy in the next lecture.