ITCS:CCT09 : Computational Complexity Theory Mar 2, 2009

Lecture 2

Lecturer: Jayalal Sarma M.N. *Scribe: Jayalal Sarma M.N.*

In the previous lecture we saw that constant factors don't matter when define class of functions accepted by a Turing machine in time t or space s. Hence the classes defined as invariant under O notation. We can define the following complexity classes based on this.

Definition 1

$$\mathsf{P} = \bigcup_{k \geq 0} \mathsf{DTIME}(n^k)$$
$$\mathsf{PSPACE} = \bigcup_{k \geq 0} \mathsf{DSPACE}(n^k)$$
$$\mathsf{L} = \mathsf{DSPACE}(\log n)$$

From the simulations we already saw it is clear that $\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE}$.

1 Diagonalisation to prove Heiraracy Theorems

In this lecture we want to introduce one of the important technique to prove lower bounds that has been used in the classical complexity theory. The method is diagonalisation and is a simple adaptation of the method you have seen in previous courses. We will do a warm up proof to review diagonalisation.

A standard example is the claim : the set of functions from $\{0,1\}^* \to \{0,1\}$ is not countable. Diagonalisation is used to prove this by contradiction. Suppose they are countable via an enumeration of the functions $f_1, f_2, \ldots$. It is easy to construct a function f which on input i is the exact negation of f_i. This function is clearly from $\{0,1\}^* \to \{0,1\}$ and is different from each function in the enumeration. Hence the contradiction. The proof we want present uses exactly the same idea, but over an enumeration of Turing machines.

We need to fix a technical notion before we state the theorem. This is called constructibility and is a sort of "niceness" for the resource function that we consider. A function $f : \mathbb{N} \to \mathbb{N}$ is called a fully space-constructible function if there exists a DTM that on any input x of length n halts visiting exactly $s(n)$ squares of the work-tape. For any fully space constructable function s we can construct a 2-tape machine that on input x marks #-symbols on exactly $s(n)$ squares on the work-tape. We will call this the *space-marking machine.* Most of the common functions, like $\lceil \log n \rceil$, cn, n^c are all space constructible functions.

Theorem 2 (Space Hierarchy Theorem) *Let $s_2(n)$ be a fully constructible function. Suppose $s_1(n) = o(s_2(n))$ then,*

$$\mathsf{DSPACE}(s_1(n)) \subsetneq \mathsf{DSPACE}(s_2(n))$$

Proof We are going to demonstrate a language L that is accepted by a Turing machine M which runs in space $s_2(n)$ and differs from every language in $\mathsf{DSPACE}(s_1(n))$. We are going to define L by giving the description for the machine M. We use the standard enumeration of one-tape Turing machines $M_1, M_2, \ldots$. This is sufficient because Tape reduction (we proved this in last lecture) does not increase space beyond a constant factor. In fact the machine M is going to be a 2-tape machine.

Given input x the machine M is set to do the following:

1. *Initialisation:* M marks off $s_2(n)$ cells on the firs tape. This is possible by the assumption that $s_2(n)$ is *space constructable*. M also writes the integer $t_2(n) = 2^{s_2(n)}$. This is just writing off $s_2(n)$ zeros and a 1 on the left. Again, space constructability of $s_2(n)$ is being used. This could be thought of as a counter for the simulation process.
2. Now M processes input x as follows : It simulates M_x in the enumeration on input x with its first tape as M_x's work-tape. If at any point of the simulation, a cell which is not marked is touched, M_x aborts the simulation and rejects. Otherwise it decrements the counter and proceeds to the next step of M_x on x. If the counter reaches down to zero, M_x halts and rejects.
3. If M_x on input x halts before counter being zero, and without touching any unmarked cells of tape 1, then this means M was able to correctly simulate M, and it needs to differ on this input. M accepts on x if and only if M_x rejects on x.

Space bound for the simulation is clearly $O(s_2(n))$. Hence the language L accepted by M is inside $\mathsf{DSPACE}(s_2(n))$.

Suppose language accepted by M is in $\mathsf{DSPACE}(s_1(n))$. Since $s_1(n) = o(s_2(n))$, there is an input x, $L(M_x) = L(M)$. Let us see how M simulates M_x on input x. Let $n = |x|$ and $t_1(n)$ be the number of possible configuration of M_x. We can also assume that $t_1(n) \leq t_2(n)$ by choice of x (we skip this calculation).

If M_x halts in time $t_2(n)$, then there is a contradiction in step 3, because M does just opposite of what M_x does on input x. If M_x does not halt in time $t_2(n)$. But by choice of x there can't be more than $t_2(n)$ configurations. Hence M_x must be running into an infinite loop where as M never does. Thus a contradiction.

This proves the theorem. ■

Note that the above proof diagonalised against an enumeration of only one-tape Turing machines. In addition M was a 2-tape Turing machine. But in the case of space as the resource, this did not make a difference (up to constant factors). However, we saw that we will have to lose a quadratic factor in terms of time as the resource. Essentially the same proof works to prove a weaker form heirarchy theorem for time as well. We will skip the details.

Theorem 3 (Time Hierarchy Theorem) *Let $t_2(n)$ be a fully constructible function. Suppose $(t_1(n))^2 = o(t_2(n))$ then,*

$$\mathsf{DTIME}(t_1(n)) \subsetneq \mathsf{DTIME}(t_2(n))$$

We did not prove this in complete detail in class. However, the details are similar to the Space Hierarchy Theorem. A stronger form of this theorem can in fact be proved by the stronger version of tape reduction that we stated in the last lecture. This version essentially says that if $t_1(n) \log t_1(n) = o(t_2(n))$, then $\mathsf{DTIME}(t_1(n)) \subsetneq \mathsf{DTIME}(t_2(n))$.

There are many corollaries to these two fundamental heirarchy theorems. However, most of them follow from the definition itself.

Corollary 4
- $\mathsf{L} \subsetneq \mathsf{PSPACE}$.
- *If* $\mathsf{L} \subseteq \mathsf{DTIME}(n^k)$ *then* $\mathsf{L} \neq \mathsf{P}$.

We did not state the second corollary above explicitly in class. But it may be interesting to note that it says, if the number of configurations of any log-space bounded TM M is bounded by $2^{c.s(n)}$ for an absolute constant c(independent of M) then, $\mathsf{L} \neq \mathsf{P}$.

2 Nondeterminism

Nondeterministic Turing machines are deterministic Turing machines with an additional *guessing power*. That is at each configuration of the computation, there are multiple values for the successor function, and the Turing machine tries to simultaneously compute in each of those choices, and accepts if atleast one of the choices accept. To make it a little more formal, the non-deterministic Turing machine has an additional *guess bit* in each state which gets associated with each configuration, and there are at most two (without loss of generality) outgoing transitions from each state for a given input and worktape contents. If the guess bit 1, the Turing machine takes one transition and if not the other. Finally, if for any input x, there is atleast one choice of the guess bits such that the computation accepts, then the machine is said to accept input x.

Similar to the deterministic complexity classes we can define their non-deterministic counterparts too.

$$\mathsf{NP} = \bigcup_{k\geq 0} \mathsf{NTIME}(n^k)$$
$$\mathsf{NPSPACE} = \bigcup_{k\geq 0} \mathsf{NSPACE}(n^k)$$
$$\mathsf{NL} = \mathsf{NSPACE}(\log n)$$

2.1 Simulating Non-determinism with Determinism

Now we will see some deterministic simulations of Non-deterministic Complexity classes. The most important one of this is the Savitch's theorem and we did this in class. However we will list down other ones too. A simple simulation of the nondeterministic machine by a deterministic Turing machine explores all possibilities of the guess bit one-by-one and checks if the given non-deterministic Turing machine accepts with that guess bits. This gives $\mathsf{NTIME}(t) \subseteq \mathsf{DSPACE}(t)$ and $\mathsf{NSPACE}(t) \subseteq \cup_{c>0}\mathsf{DSPACE}(2^{ct})$. The second inclusion also uses the elementary theorem about configurations that we proved in the first lecture. This gives $\mathsf{NL} \subseteq \mathsf{P}$ and $\mathsf{NP} \subseteq \mathsf{PSPACE}$.

Now we can prove the third main theorem of this lecture, which is the simulation due to Savitch.

Theorem 5 (Savitch's Theorem) *If $s(n)$ is space constractible, then,*

$$\mathsf{NSPACE}(s(n)) \subseteq \mathsf{DSPACE}((s(n))^2)$$

Proof To prove the theorem, let $L \in \mathsf{NSPACE}(s(n))$ via a Turing machine M. Since the space used is bounded by $s(n)$ there can be atmost $2^{s(n)}$ configurations in the configuration graph of M. M accepts if and only if there is atleast one path from the start configuration to the accepting configuration (we can ensure that there is at most one accepting configuration).

Now we define the following predicate for each configuration α, β, $\mathsf{reach}(\alpha, \beta, k)$ is 1 if and only if there is a computation path starting from configuration α to configuration β of length at most k.

Basically we will give a way to test if $\mathsf{reach}(\alpha, \beta, 2^{O(s(n))})$. We will keep this $2^{O(s(n))}$ implicitly so that we will not need to find the explicit constant. The basic idea is a simple search for a middle node. That is,

$$\mathsf{reach}(\alpha, \beta, k) \iff \exists\gamma \left(\mathsf{reach}\left(\alpha, \gamma, \left\lceil\frac{k}{2}\right\rceil\right) \wedge \mathsf{reach}\left(\gamma, \beta, \left\lfloor\frac{k}{2}\right\rfloor\right)\right)$$

This naturally gives rise to a deterministic recursive algorithm which runs over all possible γ that could be the middle node. It is clear that the depth of the recursion is bounded by $\log k$. At each step of the recursion, the argument includes γ which is a $\log(2^{(O(s(n)))}) = O(s(n))$ sized number. Thus the above recursive algorithm can be simulated by a stack which uses atmost $\log k.O(s(n))$ space. This gives the $O((s(n))^2)$ space bound. ■

An easy corollary of Savitch's theorem is that for PSPACE non-determinism does not really help. That is PSPACE = NPSPACE.

There is a version of heirarchy theorem for non-deterministic Turing machines. We will leave some of it to the excerices.