INFR11102: Computational Complexity 26/09/2019

Lecture 4: Hierarchy theorems; Robust Complexity Classes

Lecturer: Heng Guo

1 Delayed Diagonalisation

Recall the time hierarchy theorem.

Theorem 1. *Let $T_1(n)$ and $T_2(n)$ be two functions such that T_2 is time-constructible. If $\lim_{n\to\infty} \frac{T_1(n)^2}{T_2(n)} = 0$ (or equivalently, $T_1(n)^2 = o(T_2(n))$), then there exists a language $L \in \mathtt{DTime}[T_2(n)] - \mathtt{DTime}[T_1(n)]$.*

Last time we talked about why a naive diagonalisation argument does not work. We will then use a technique called *delayed diagonalisation*.

We construct a TM A' in Algorithm 1.

Algorithm 1 A'

Input: $x \in \{0,1\}^*$.
Simulate $M_x(x)$, where M_x is the TM encoded by x.
On a separate tape, run the clock of $T_2(|x|)$.
if $T_2(|x|)$ time is reached, **then**
  return 1.
end if
return $1 - M_x(x)$.

From the description above, it is easy to see that $L(A') \in \mathtt{DTime}[T_2(n)]$ because of the clock. Now the main task is to show that $L = L(A') \notin \mathtt{DTime}[T_1(n)]$.

Suppose otherwise, and $L \in \mathtt{DTime}[T_1(n)]$. Then there is a constant C such that L can be computed by some TM M (not necessarily A') with time $CT_1(n)$. Let x_0 be an encoding of M such that $n = |x_0|$ be sufficiently large that $T_2(n) \geq C_M C T_1^2(n)$, where C_M is the constant in the simulation complexity of UTM. Although this constant C_M does depend on the encoding length of M (the number of states, transitions, etc. in M), it does not depend on the input x. The important thing here is that padding does not change the behaviour of M and thus does not increase C_M, so our assumption on the existence of n is valid. Hence, the UTM simulation of $M_{x_0}(x_0)$ takes time at most $C_M C T_1^2(n) \leq T_2(n)$. Namely, the "if" line in the description of A' doesn't execute on the input x_0.

As before, we now ask what is the value of $A'(x_0)$, or equivalently whether $x_0 \in L$?

- If $x_0 \in L$, then A' returns 1 in the end, implying $M_{x_0}(x_0) = 0$, which in turn implies $x_0 \notin L$. Contradiction!

- If $x_0 \notin L$, then A' returns 0 in the end, implying $M_{x_0}(x_0) = 1$, which in turn implies $x_0 \in L$. Contradiction!

This completes the proof.

This argument is called a delayed diagonalisation, because the diagonalisation action may be taken later than the first assumed encoding.

Remark. It is easy to see that the condition $\lim_{n\to\infty} \frac{T_1(n)^2}{T_2(n)} = 0$ is only used for the simulation step. If we can simulate TMs more efficiently, then this condition can be made stronger. Indeed, any k-tape TM with time complexity $T(n)$ can be simulated by a 2-tape TM in time $O(T(n) \log T(n))$ (shown by Hennie and Stearns [HS66], details can be found in [AB09, Theorem 1.9], or the supplementary note on the course website). This simulation leads to a better assumption $\lim_{n\to\infty} \frac{T_1(n) \log T_1(n)}{T_2(n)} = 0$. We will not show it here.

Completely analogously, we can show a space hierarchy theorem. Recall that in the UTM simulation, we only need to use $S(|x|) + C_M$ space, where $S(\cdot)$ is the space complexity of M, and C_M is a constant only depending on M. Hence, the space hierarchy is slightly tighter than the time one.

Theorem 2. *Let $S_1(n)$ and $S_2(n)$ be two functions such that $S_2(n) \geq \log n$ is space-constructible. If $\lim_{n\to\infty} \frac{S_1(n)}{S_2(n)} = 0$, then there exists a language $L \in \texttt{DSpace}[S_2(n)] - \texttt{DSpace}[S_1(n)]$.*

Similar to time-constructibility, a function $s(n)$ is *space-constructible* if there is a TM M such that on input of length n (like $1^n = 111\cdots1$ with n 1's), M halts while using exactly $s(n)$ cells on the working tape.

The proof of Theorem 2 is completely analogous to that of Theorem 1, with a couple of subtle twists. It will be left as an exercise.

Note that without the condition $S_2(n) \geq \log n$, the conclusion of Theorem 2 is actually not true. However we do not need to assume $T_2 \geq n$ in Theorem 1. This is implicitly implied by the fact that T_2 is time-constructible (otherwise there is not enough time to read the whole input). However Theorem 1 is still true if $T_2(n) = O(n)$. In this case $T_1 = o(n)$, and thus within $T_1(n)$ time, the algorithm cannot even read, say, the $T_2(n)$th bit. Therefore it is easy to design a language that is in $\texttt{DTime}[T_2]$ but not $\texttt{DTime}[T_1]$ (e.g. simply output the $T_2(n)$th bit).

2 Robust Complexity Classes

A good complexity class should contain interesting problems, as well as be robust against small tweaks on the model of computation. The ones we see last time, like $\texttt{DTime}[n]$, are not necessarily robust classes. For example, if we consider 1-tape machines instead of k-tape ones, then what $\texttt{DTime}[n]$ contains is actually different.

Also, in practise, it is realized that polynomial-time computation is often feasible whereas exponential time computation is not. This is formulated as *the Cobham-Edmonds thesis*,

in which Cobham [Cob65] and Edmonds [Edm65] have independently proposed polynomial-time as the "correct" notion of efficient computation. However, the importance of "polynomial-time" computation is only realized a few years later.

Let us formally define $\mathtt{P}$.

$$\mathtt{P} := \bigcup_{c\in\mathbb{N}} \mathtt{DTime}[n^c].$$

It is easy to see that $\mathtt{P}$ is closed with respect to some polynomial-time tweaks. For example, $\mathtt{P}$ stands the same for 1-tape and k-tape TMs. It is known as *the extended Church-Turing thesis*, that $\mathtt{P}$ is invariant regardless of the model of computation. This thesis is challenged nowadays by quantum computation, but we still do not know for sure. We will talk about it later if we have time.

Other robust complexity classes are

$$\begin{aligned}
\mathtt{L} &:= \mathtt{DSpace}[\log n];\\
\mathtt{PSpace} &:= \bigcup_{c\in\mathbb{N}} \mathtt{DSpace}[n^c];\\
\mathtt{E} &:= \bigcup_{c\in\mathbb{N}} \mathtt{DTime}[e^{cn}];\\
\mathtt{Exp} &:= \bigcup_{c\in\mathbb{N}} \mathtt{DTime}[e^{n^c}].
\end{aligned}$$

Note the difference between $\mathtt{E}$ and $\mathtt{Exp}$.

It is not hard to establish the following relations among these classes:

$$\mathtt{L} \subseteq \mathtt{P} \subseteq \mathtt{PSpace} \subseteq \mathtt{Exp}.$$

Clearly $\mathtt{P} \subseteq \mathtt{Exp}$, $\mathtt{L} \subseteq \mathtt{PSpace}$, and $\mathtt{P} \subseteq \mathtt{PSpace}$. In fact, by the Hierarchy Theorems, $\mathtt{L} \subsetneq \mathtt{PSpace}$ and $\mathtt{P} \subsetneq \mathtt{Exp}$. But why $\mathtt{L} \subseteq \mathtt{P}$ and $\mathtt{PSpace} \subseteq \mathtt{Exp}$?

We define a (directed) configuration graph $G_{M,x}$ of a TM M with input x that runs in space $S(n) \geq \log n$ where $n = |x|$. Recall that a configuration is a "snapshot" of M during the execution. Each vertex in $G_{M,x}$ is a configuration and (i, j) is an arc if i goes to j. In particular, every vertex has out degree 1 in $G_{M,x}$.

How large is the graph $G_{M,x}$? Note that in n bits, we can count at most 2^n distinct objects. How many bits do we need to use to specify a configuration? We need to specify the input head position ($\log n$ bits), the content of the work tape and the head position ($S(n) + \log S(n)$ bits), and the current state ($\log |Q|$ bits). Hence, in total we need only $\log n + S(n) + \log S(n) + \log |Q| \leq cS(n)$ bits to record a configuration, where c is a constant depending on M but not x. As a consequence, there are only $2^{cS(n)}$ possible vertices in $G_{M,x}$. We may construct this graph in time $O(2^{cS(n)})$ and using the standard BFS to figure out whether the starting state is connected to the accepting state. Thus it implies that

$$\mathtt{DSpace}[S(n)] \subseteq \bigcup_{c\in\mathbb{N}} \mathtt{DTime}[2^{cS(n)}]. \tag{1}$$

In particular, we have that $\mathtt{L} \subseteq \mathtt{P}$ and $\mathtt{PSpace} \subseteq \mathtt{Exp}$.

Also note that if we allow polylog spaces, namely, define

$$\mathtt{PolyLog} := \bigcup_{c \in \mathbb{N}} \mathtt{DSpace}[\log^c n],$$

we still do not know whether $\mathtt{PolyLog} \subseteq P$? Similarly, we do not know whether $\mathtt{PSpace} \subseteq \mathtt{E}$?

Remark (Bibliographic). Relevant chapters are [AB09, Chapter 1] and [Pap94, Chapter 9].

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[Cob65] Alan Cobham. The intrinsic computational difficulty of functions. In *Logic, methodology and philosophy of science, Proceedings of the 1964 International Congress*, pages 24–30, 1965.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17(3):449–467, 1965.

[HS66] F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape Turing machines. *J. ACM*, 13(4):533–546, 1966.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.