INFR11102: Computational Complexity 29/10/2019

Lecture 13: More on circuit models; Randomised Computation

Lecturer: Heng Guo

1 TM taking advices

An alternative way to characterize $\mathtt{P}_{/\mathtt{poly}}$ is via TMs that take advices.

Definition 1. *For functions $F : \mathbb{N} \to \mathbb{N}$ and $A : \mathbb{N} \to \mathbb{N}$, the complexity class $\mathtt{DTime}[F]_{/A}$ consists of languages L such that there exist a TM with time bound $F(n)$ and a sequence $\{a_n\}_{n\in\mathbb{N}}$ of "advices" satisfying:*

- *$|a_n| \le A(n)$;*
- *for $|x| = n$, $x \in L$ if and only if $M(x, a_n) = 1$.*

The following theorem explains the notation $\mathtt{P}_{/\mathtt{poly}}$, namely "polynomial-time with polynomial advice".

Theorem 1. $\mathtt{P}_{/\mathtt{poly}} = \bigcup_{c,d\in\mathbb{N}} \mathtt{DTime}[n^c]_{/n^d}$.

Proof. If $L \in \mathtt{P}_{/\mathtt{poly}}$, then it can be computed by a family $\mathcal{C} = \{C_1, C_2, \cdots\}$ of Boolean circuits. Let a_n be the description of C_n, and the polynomial time machine M just reads this description and simulates it. Hence $L \in \bigcup_{c,d\in\mathbb{N}} \mathtt{DTime}[n^c]_{/n^d}$.

For the other direction, if a language L can be computed in polynomial-time with polynomial advice, say by TM M with advices $\{a_n\}$, then we can construct circuits $\{D_n\}$ to simulate M, as in the theorem $\mathtt{P} \subset \mathtt{P}_{/\mathtt{poly}}$ in the last lecture. Hence, $D_n(x, a_n) = 1$ if and only if $x \in L$. The final circuit C_n just does exactly what D_n does, except that C_n "hardwires" the advice a_n. Namely, $C_n(x) := D_n(x, a_n)$. Hence, $L \in \mathtt{P}_{/\mathtt{poly}}$. □

2 Karp-Lipton Theorem

Dick Karp and Dick Lipton showed that $\mathtt{NP}$ is unlikely to be contained in $\mathtt{P}_{/\mathtt{poly}}$ [KL80]. Since $\mathtt{P} \subset \mathtt{P}_{/\mathtt{poly}}$, if we could rule out the possibility that $\mathtt{NP} \subseteq \mathtt{P}_{/\mathtt{poly}}$, then we would have separated $\mathtt{P}$ and $\mathtt{NP}$. Karp-Lipton theorem stirred a lot of effort trying to show circuit lower bounds. Indeed many successes followed, but unfortunately, we hit some barriers later on.

Theorem 2. *If $\mathtt{NP} \subseteq \mathtt{P}_{/\mathtt{poly}}$, then $\mathtt{PH} = \Sigma_2^p$.*

Proof. As shown before, we just need to show that $\Pi_2^p \subseteq \Sigma_2^p$ to conclude $\mathtt{PH} = \Sigma_2^p$. Let $L \in \Pi_2^p$. Then, there exists a poly-time TM M such that

$$x \in L \Leftrightarrow \forall y \exists z,\ M(x,y,z) = 1, \text{ and } y \text{ and } z \text{ are poly-size.}$$

All strings appear in this proof actually have polynomial length, and thus we use notations $\exists^p$ and $\forall^p$ to denote this. Due to the Cook-Levin theorem, the expression $\exists z,\ M(x,y,z) = 1$ can be converted in polynomial-time to a CNF formula φ such that

$$x \in L \Leftrightarrow \forall^p y,\ \varphi(x,y) \in \textsc{Sat}. \tag{1}$$

To show $L \in \Sigma_2^p$, we need to somehow change the quantifier in (1).

Our assumption is that $\mathtt{NP} \in \mathtt{P}_{/\mathtt{poly}}$, namely $\textsc{Sat}$ has a polynomial-size circuit family $\mathcal{C} = \{C_1, C_2, \cdots\}$.

For an input x, $|x| = n$, we give an Σ_2^p algorithm to decide whether $x \in L$. We first guess a circuit C for $\textsc{Sat}$. It may have two types of error: $\varphi \in \textsc{Sat}$ but $C(\varphi) = 0$ or $\varphi \notin \textsc{Sat}$ but $C(\varphi) = 1$. There is at least one correct guess, namely $C_{p(n)}$ where $p(n) = |\varphi(x,y)|$, that works correctly on $\varphi(x,y)$ by assumption.

For any such guess C, we construct a TM M' on input (C, φ) so that its error is "one-sided". It relies on the fact that $\textsc{Sat}$ is *self-reducible*. Namely given an oracle to the decision version, we can efficiently find a solution. M' takes C as a subroutine, and if C answers "Yes", then M' proceeds to find a satisfying assignment by repeatedly asking C. If there is any inconsistency from C (namely C says that a formula is satisfiable but both assignment of some variable x make it unsatisfiable), or the final assignment is not satisfying, then M' rejects. Otherwise M' accepts if C accepts. This machine M' never accepts a formula $\varphi \notin \textsc{Sat}$, but it may reject $\varphi \in \textsc{Sat}$ if C is an incorrect guess for $\textsc{Sat}$.

Now, we claim the following:

$$x \in L \Leftrightarrow \exists^p C \forall^p y,\ M'(C, \varphi(x,y)) = 1. \tag{2}$$

Comparing to (1), we have changed the order of quantifiers, and the right hand side of (2) can be easily converted to a Σ_2^p expression (namely compute $\varphi(x,y)$ first). What we are left to do is to verify (2).

- If $x \in L$, then there is a correct guess $C_{p(n)}$ making the right hand side of (2) true.
- If $x \notin L$, then by (1), any y_0 will make $\varphi(x, y_0) \notin \textsc{Sat}$. Recall that M', whatever the guess C is, never accepts $\varphi \notin \textsc{Sat}$. Hence, the right hand side of (2) is false. □

Why doesn't the proof above work if we don't construct M' and simply use C?

Remark (Bibliographic). Karp and Lipton first proved a weaker result than Theorem 2, namely collapsing to the third level of $\mathtt{PH}$. In their paper [KL80] they attribute this stronger version to Mike Sipser. The proof given here uses an idea first noted by John Hopcroft.

Relevant chapters are [AB09, Chapter 6.4] and [Pap94, Chapter 17].

3 Randomised computation

A great discovery in the theory of computation is the power of randomness. There are a lot of surprising randomised algorithms discovered since the 80s. Examples include, primality testing [Mil76, Rab80], polynomial identity testing [Zip79, Sch80], volume computation [DFK91], and many more.

To formalize the idea of randomised computation, we need the notion of a probabilistic Turing Machine (PTM).

A *probabilistic Turing Machine* is a TM with two transition functions δ_0 and δ_1. At every step of an execution with input x, we apply δ_0 with probability $1/2$, and δ_1 otherwise. For a PTM M, its output $M(x)$ is now a random variable. We say M runs in time $T(n)$ if for any input of length n, M halts within $T(n)$ time regardless of the random choices made.

The standard class for efficient randomised computation is called $\mathtt{BPP}$ (bounded-error probabilistic polynomial-time).

Definition 2. *A language L is in $\mathtt{BPP}$ if there exists a PTM P and a polynomial $p(\cdot)$, such that P runs in time $p(n)$ and*

- *if $x \in L$, then $\Pr[P(x) = 1] \geq 3/4$;*
- *if $x \notin L$, then $\Pr[P(x) = 1] \leq 1/4$.*

In other words, the requirement is that $\Pr[P(x) = L(x)] \geq 3/4$. The constant $3/4$ in Definition 2 is not essential, as long as it is strictly larger than $1/2$. This is captured by a notion called "amplification". On the other hand, if it is $1/2$, then the class defined would be (under standard assumptions) more powerful. That class is called $\mathtt{PP}$, probabilistic polynomial-time. Historically $\mathtt{PP}$ is defined earlier than $\mathtt{BPP}$, and as it turns out, $\mathtt{PP}$ is the *wrong* definition for efficient randomised computation.

We also have the "one-sided" error version of $\mathtt{BPP}$, called $\mathtt{RP}$ (randomised polynomial-time).

Definition 3. *A language L is in $\mathtt{RP}$ if there exists a PTM P and a polynomial $p(\cdot)$, such that P runs in time $p(n)$ and*

- *if $x \in L$, then $\Pr[P(x) = 1] \geq 3/4$;*
- *if $x \notin L$, then $\Pr[P(x) = 1] = 0$.*

In other words, an $\mathtt{RP}$ algorithm never errs if $x \notin L$, but it may reject some $x \in L$.

Remark (Bibliographic). Relevant chapters are [AB09, Chapter 7.3, 7.4] and [Pap94, Chapter 11].

References

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

[DFK91] Martin E. Dyer, Alan M. Frieze, and Ravi Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991.

[KL80] Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *STOC*, pages 302–309. ACM, 1980.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13(3):300–317, 1976.

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

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12(1):128 – 138, 1980.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, volume 72 of *LNCS*, pages 216–226. Springer, 1979.