University of California, Los Angeles
CS 281 Computability and Complexity

Instructor: Alexander Sherstov
Scribe: Glenn Sun
Date: November 5, 2020

LECTURE
11

Oracle Characterization of Σ_k and Π_k and Introduction to Randomness

This lecture begins by wrapping up our discussion of the polynomial hierarchy by giving an alternative way to understand the classes Σ_k and Π_k through the lens of oracle-aided computation. Then, we begin our discussion of randomized algorithms. Because this unit will require more mathematical background than previous units, we first quickly review some mathematics, then discuss our first example of a randomized algorithm.

11.1 Oracle characterization of Σ_k and Π_k

Recall that we used notations such as P^O (or NP^O) to mean the class of languages computable with a polynomial time deterministic (resp. non-deterministic) Turing machine with the assistance of an oracle that decides the language O. When O is a complete problem for some complexity class, for example NP-complete, we may also write P^O and NP^O as P^{NP} and $\mathrm{NP}^{\mathrm{NP}}$, respectively. This notation is well-defined, because for any two NP-complete languages O and O', we may convert between instances of either problem in deterministic polynomial time, so $\mathrm{P}^O = \mathrm{P}^{O'}$ (resp. for NP).

In Stockmeyer's original paper definining the polynomial hierarchy [2], the oracle characterization of Σ_k and Π_k were actually used as the definition. For us, it is just an alternative way to understand the polynomial hierarchy. We begin by showing a characterization of Σ_2, and then we will generalize to the rest of the polynomial hierarchy.

Proposition 11.1. $\Sigma_2 = \mathrm{NP}^{\mathrm{NP}}$.

Proof. ($\subset$) Let $L \in \Sigma_2$, so that by definition, there exists a polynomial time Turing machine m such that

$$x \in L \iff \exists u_1\, \forall u_2\, M(x, u_1, u_2) = 1.$$

Note that we may replace $\forall u_2\, M(x, u_1, u_2) = 1$ with $\neg\exists u_2\, M(x, u_1, u_2) = 0$. Hence it suffices to consider u_1 as the NP certificate, and ask the NP oracle for the answer to $\exists u_2\, M(x, u_1, u_2) = 0$. We simply output the opposite.

($\supset$) This direction is a little trickier, since the oracles are computationally unbounded. The best time bound we have for a SAT-solver is still EXP, so the trivial simulation clearly is not possible. Instead, fix a language $L \in \mathrm{NP}^{\mathrm{NP}}$ with a Turing machine M that decides it appropriately, and consider the following:

This image has been redacted for copyright.

That is, we observe the communication transcript between the Turing machine M and the SAT oracle. The machine M first asks a question only dependent on x and u, and receives a single bit a_1 back from the oracle. Then the next question asked by M is a function of x, u, and a_1, and the process repeats polynomially many times. Instead of simulating the oracle itself, we will try to simply verify the communication between M and the oracle instead.

We encode this idea into a Σ_2 formula as follows. Note that the output of M is 1 (equivalently, $x \in L$) if and only if there exists a certificate u, SAT queries $\varphi_1, \dots \varphi_{|x|^c}$, and answers $a_1, \dots, a_{|x|^c}$ (i.e. there exists a transcript) such that for all i, the following is true:

1. The SAT query φ_i is indeed the query posed by M given the transcript up to i.

2. The query φ_i is in the SAT language if and only if $a_i = 1$.

Note that the "for all i" quantification is not really a quantifier, since there are only polynomially many i, so we can just check them all in polynomial time. So we have only used one quantifier for the transcript so far.

Clearly, (1) can be computed in polynomial time by simulating the Turing machine M. However, (2) requires additional quantification, since it asks us to solve both SAT and the complement problem UNSAT. This uses an existential quantifier and a universal quantifier. But since the expressions within these quantifiers don't depend on each other, we may move them to the outside, so that the existential merges with the other existential quantifier, and we have a Σ_2 formula that is true if and only if $x \in L$. □

As a side note, note that as an oracle, any class has the same power as its complement class, since we may simply ask the oracle for the answer and negate it. Hence, we also have $\Sigma_2 = \mathrm{NP}^{\mathrm{co\,NP}}$. And, we also have $\Pi_2 = (\mathrm{co\,NP})^{\mathrm{NP}} = (\mathrm{co\,NP})^{\mathrm{co\,NP}}$, since $\mathrm{co}\,\Sigma_2 = \Pi_2$ and $\mathrm{co}(\mathrm{NP}^{\mathrm{NP}}) = (\mathrm{co\,NP})^{\mathrm{NP}}$.

Also, to get a similar conclusion for higher levels of the polynomial hierarchy, it can be easily seen that the argument remains essentially the same, just with more quantifiers. Hence, we have the following characterization of the polynomial hierarchy.

THEOREM 11.2. $\Sigma_k = \mathrm{NP}^{\Sigma_{k-1}}$ *and* $\Pi_k = \mathrm{co}\,\mathrm{NP}^{\Sigma_{k-1}}$.

This concludes our discussion of the polynomial hierarchy, but we will continue to see how the polynomial hierarchy relates to other complexity classes as we define various other complexity classes in future lectures.

11.2 Mathematical preliminaries to randomness

As we move into more modern and more advanced topics in complexity theory, we will need to use more advanced mathematics. This section contains a terse list of mathematical definitions and results that will be important for our study of randomness and beyond. Much of it may be review, but some results are likely new.

11.2.1 Probability

We will not need measure theoretic probability (or even continuous probability) because for a fixed input, algorithms cannot use more than a finite number of random bits.

DEFINITION 11.3 (finite probability space). A finite probability space is a tuple $(\Omega, \mathbb{P})$ where Ω is a finite set and $\mathbb{P} : 2^\Omega \to [0, \infty)$ is a function satisfying:

1. $\mathbb{P}(\Omega) = 1$
2. If $E_1, E_2 \subset \Omega$ are disjoint, then $\mathbb{P}(E_1) + \mathbb{P}(E_2) = \mathbb{P}(E_1 \cup E_2)$.

For the rest of this section, let $(\Omega, \mathbb{P})$ be a finite probability space.

PROPOSITION 11.4 (Union bound). *Suppose* $E_1, \ldots, E_n \subset \Omega$. *Then*

$$\mathbb{P}\left(\bigcup_{k=1}^{n} E_k\right) \leq \sum_{k=1}^{n} \mathbb{P}(E_k).$$

Proof. The proof follows by induction and the fact that one may write $A \cup B$ as the union of disjoint events A and $B \setminus A$, and the fact that one may write B as the union of disjoint events $B \setminus A$ and $B \cap A$. ∎

DEFINITION 11.5 (random variable). A random variable is a function $X : \Omega \to \mathbb{R}$. We often abuse notation and write $X > k$ to mean $\{x \in \Omega : X(x) > k\}$ (likewise $\geq$, $<$, $\leq$, $=$).

DEFINITION 11.6 (expectation). The expectation of a random variable X is the real number

$$\mathbb{E}(X) = \sum_{k \in \mathbb{R}} k \cdot \mathbb{P}(X = k).$$

The notation is well-defined because there are finitely many k (in particular, at most $|\Omega|$) such that $\mathbb{P}(X = k)$ is non-zero.

THEOREM 11.7 (Markov's inequality). *For any non-negative random variable X, we have*

$$\mathbb{P}(X \geq k\mathbb{E}(X)) \leq \frac{1}{k}.$$

Proof. Note that $\mathbb{E}(X) \geq a\mathbb{P}(X \geq a)$ for any $a \geq 0$ by the definition. Hence, setting $a = k\mathbb{E}(X)$, we get $\mathbb{E}(X) \geq k\mathbb{E}(X)\mathbb{P}(X \geq k\mathbb{E}(X))$, and dividing by $k\mathbb{E}(X)$ yields the result that we want. ∎

THEOREM 11.8 (Chernoff bound). *Let $X_1, \ldots, X_n$ be independent indicator random variables with probability of success p. Then,*

$$\Pr\left(\left|\frac{X_1 + \cdots + X_n}{n} - p\right| \geq \delta\right) \leq 2e^{-2\delta^2 n}$$

Proof. This proof is rather long and tedious, so it is deferred to [3]. ∎

Markov's inequality and the Chernoff bound will be crucial tools to help us bound the probability of tail events. In particular, Markov's inequality is a general but weak tool that says the probability of exceeding the expectation by a certain multiplicative factor decreases linearly by that factor. In contrast, the Chernoff bound makes the strong assumption that the random variable is a sum of identical indicator random variables, and concludes that the chance of deviating from its expectation decreases *exponentially* in both the error and the number of indicator random variables.

11.2.2 Algebra

DEFINITION 11.9 (field). A field is a tuple $(F, +, \cdot)$ such that the operations $+ : F \times F \to F$ and $\cdot : F \times F \to F$ respect the usual associative, commutative, identity, inverse, and distributive properties. A finite field is a field where F is a finite set.

PROPOSITION 11.10. *For any prime p, $(\mathbb{F}_p, +, \cdot)$ is a field, where $\mathbb{F}_p = \{0, 1, \ldots, p-1\}$ and the operations are the normal operations mod p.*

Proof. The only non-trivial part is the existence of multiplicative inverses. Let $a \in \mathbb{F}_p$. Note that $\gcd(a, p) = 1$, so by the Euclidean algorithm there exists $n, m \in \mathbb{Z}$ such that $an + pm = 1$. But $an + pm \equiv an \pmod{p}$, so we may simply take $a^{-1} = n$. ∎

DEFINITION 11.11 (polynomial). A polynomial in n variables $x_1, \ldots, x_n$ over a field F is sum of terms of the form $ax_1^{d_1} \cdots x_n^{d_n}$, where $a \in F$ and $d_1, \ldots, d_n \in \mathbb{N}$. We often denote polynomials by $p(x)$ or $p(x_1, \ldots, x_n)$, and the set of all n variable polynomials over F by $F[x_1, \ldots, x_n]$. The degree of a polynomial is the maximum value of $d_1 + \cdots + d_n$ over all of its terms. Polynomials may be evaluated at points $(a_1, \ldots, a_n) \in F^n$ by substituting the point in for $x_1, \ldots, x_n$ and evaluating the expression.

THEOREM 11.12 (Schwartz–Zippel lemma). *Let $p(x) \in F[x_1, \ldots, x_n]$ be a polynomial in n variables over the field F. Then for any finite subset $S \subset F$, if $r_1, \ldots, r_n$ are sampled from S independently at random, then*

$$\mathbb{P}(p(r_1, \ldots, r_n) = 0) \leq \frac{\deg p}{|S|}.$$

Proof. The proof is by induction on n. For a complete write-up, see [5]. ∎

In particular, if F is a finite field, then it is difficult to hit a root of a polynomial by randomly choosing points in the field. This critically important fact can be viewed as a generalization of the fundamental theorem of algebra, and we will use it extensively in our study of randomized algorithms.

Theorem 11.13 (Bertrand's postulate). *For every positive integer n, there exists a prime in the interval $[n, 2n]$.*

Proof. The proof is elementary but long, see [4] for a proof. $\square$

Bertrand's postulate will allow us to find finite fields of sufficiently large size for our algorithms. In particular, it guarantees that if we just check every number from n to $2n$ by brute force, we will find a prime, and this can trivially be done in time polynomial in n.

11.3 Introduction to randomized algorithms

Randomized algorithms form an exciting field in computer science because it turns out that compared to deterministic algorithms, they are often faster, simpler to implement, more parallelizable, and just as practical. The core idea is that by using randomness in our algorithms, we can get algorithms that are correct not always, but almost always, and that this is sufficient for many purposes. This is the area that we will study for the rest of the lecture and for the next two lectures.

Note that randomness is not the same as nondeterminism. Whereas nondeterminism is not an very practical thing to ask for, since real machines cannot really find the single correct computation path by its own volition, randomness means that most computation paths are correct, and real computers can select one of these correct paths with good probability using thermal noise from the CPU or other sources of randomness.

Let us demonstrate the power of randomness with a quick example. Consider the problem of verifying multiplication of two matrices in $F^{n\times n}$, where F is a field. That is, given $A, B, C \in F^{n\times n}$, does $AB = C$? A deterministic solution to this problem would be to simply perform the multiplication and check it, which would take $O(n^3)$ time with schoolbook matrix multiplication, or around $O(n^{2.37})$ time with the currently best known algorithms [1]. However, with the power of randomness, we have the following:

Proposition 11.14. *Matrix multiplication can be verified in $O(n^2)$ time to an accuracy of $1 - \frac{1}{|F|}$ if F is finite or 99% if F is infinite.*

Proof. Consider the following algorithm: Pick $x \in \mathbb{F}^n$ randomly. Then output whether or not $ABx = Cx$. Since matrix-vector multiplication takes $O(n^2)$ time, and the left hand may be computed as $A(Bx)$, this takes $O(n^2)$ time. For inputs where $AB = C$, the output will always be correct, so it remains to show that for inputs where $AB \neq C$, the probability that $ABx = Cx$ is less than $\frac{1}{|F|}$ or $\frac{1}{\epsilon}$.

In other words, we are considering when $(AB - C)x = 0$. Because $AB \neq C$, some component of $(AB - C)x$ is a non-zero degree 1 polynomial. Then by the Schwarz-Zippel lemma, the probability that we hit a root in that component is at most $\frac{1}{|\mathbb{F}|}$. The probability of $(AB - C)x = 0$ (in all components) is hence also at most $\frac{1}{|\mathbb{F}|}$. If we take the field to be infinite, we may simply choose a subset $S \subset \mathbb{F}$ of size $|S| = 100$ and take $x \in S^n$, and the probability of failure is likewise $\frac{1}{|S|} = \frac{1}{100}$, as we wanted. $\square$

Note that a key property of this algorithm is that the success probability does not depend on the input. In other words, the success probability is at least 99% on *all* inputs! We could have easily also changed the size of S to obtain even better success probabilities. This makes randomized algorithms very practical, since in real life, we may be pretty satisfied with an algorithm that is practically always right, even if it very occasionally is wrong. In the next lecture, we will discuss more examples of randomized algorithms and begin to define and understand the complexity classes that go along with them.

References

[1] J. Alman and V. V. Williams. A refined laser method and faster matrix multiplication, 2020.

[2] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1 – 22, 1976.

[3] Wikipedia contributors. Chernoff bound — Wikipedia, the free encyclopedia, 2020.

[4] Wikipedia contributors. Proof of bertrand's postulate — Wikipedia, the free encyclopedia, 2020. [Online; accessed 10-December-2020].

[5] Wikipedia contributors. Schwartz–zippel lemma — Wikipedia, the free encyclopedia, 2020.