CS 6810: Theory of Computing **Spring 2026**

Lecture 11: March 3, 2026

Lecturer: Eshan Chattopadhyay *Scribe: Zachary Cheslock*

1 Overview

Recall from last lecture that we defined P/poly as all languages computable by polysized circuits, and proved that $P \subseteq \text{P/poly}$. We will build off this today, constructing P/poly from a non-uniform model of turing machines. Then, we will move on to examine the most promising direction for proving $\mathbf{NP} \not\subseteq \mathbf{P}$ by examining AC^0 circuits, a subclass of circuits where we have been able to prove some lower bounds.

2 Non-Uniform Turing Machines

We seek to construct the class P/poly from a non-uniform model of Turing Machines. To do this, we consider multi-tape Turing Machines with an extra advice tape. In addition to the input x on the input tape, this TM also has advice α on the advice tape in its initial state. Importantly, this advice α can only depend on the *input length*, not the specific input itself.

Definition 2.1. *For any language $\mathcal{L} \subset \{0,1\}^*$, $\mathcal{L} \in \text{DTIME}(t(n))/a(n)$ if there exists a multi-tape Turing Machine M with an advice tape and advice $\alpha_1, \alpha_2, \ldots$ with α_i of length $a(i)$, such that on input x (of length n) M with advice α_n runs for at most $O(t(n))$ steps and accepts iff $x \in \mathcal{L}$.*

This gives us our desired construction of P/poly, as the union of all Turing Machines with polynomial-size advice that run in polynomial time.

Remark 2.2. *Note that there need not be a TM that is able to compute α_n on input 1^n. The advice strings are simply assumed to be given.*

Lemma 2.3.

$$\text{P/poly} = \bigcup_{c_1, c_2 \geq 0} \text{DTIME}(n^{c_1})/n^{c_2}$$

We will sketch the main ideas of a proof.

Proof. First, suppose that $\mathcal{L} \in \text{DTIME}(n^{c_1})/n_2^c$. We want to show that there exists a polysize circuit family $\{C_n\}$ that computes $\mathcal{L}$. The argument is very similar to the proof that $P \subseteq \text{P/poly}$ from last lecture, and we will outline the alteration.

From $P \subseteq \text{P/poly}$, we know that a TM with no advice can be simulated by a polysize circuit. Now, we also need the circuit to depend on the advice α_n. Since the circuit family $\{C_n\}$ has a different circuit for each input length, we can hardcode the advice α_n into C_n for each n. Since $\alpha_n \leq n^{c_2}$, adding this advice to the input maintains the polynomial size of the circuit. Then, the circuit can simulate the TM identically to the $P \subseteq \text{P/poly}$ proof, where we have both the input x and the advice α_n as inputs to C_n.

In the other direction, we suppose that $\mathcal{L} \in \text{P/poly}$, so there exists a circuit family $\{C_n\}$ with $\text{size}(C_n) \leq n^c$ that computes $\mathcal{L}$. We want to show there exists a polytime TM with polynomial advice for $\mathcal{L}$.

We define the advice α_n to be a description of the circuit C_n, which is polynomial length since C_n is polynomial size. Note that the circuit C_n only depends on the input length, so the same is true of the advice. Then, we define a TM M which on input $x \in \{0,1\}^n$, uses the advice α_n to simulate C_n. □

Remark 2.4. *Recall that circuits are a non-uniform model of computation. Standard TMs are a uniform model of computation, but we design a non-uniform model of TMs here. One could also consider a uniform model of circuit families; a "uniform family of circuits" is often defined as a circuit family $\{C_n\}$ where the circuit descriptions α_n are computable by an efficient TM. Essentially, choosing a different circuit for each input length does not provide any computational power since an efficient TM could efficiently compute the circuit given the input (or its length).*

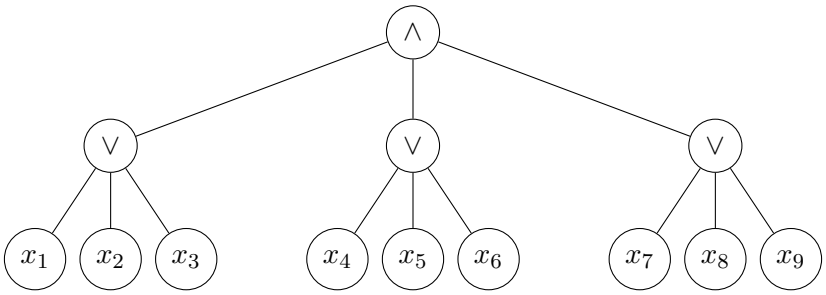
3 AC^0 circuits

Recall from last lecture that we want to show that 3-SAT has no polynomial-size circuit, as it would imply $\mathbf{P} \subsetneq \mathbf{NP}$. However, it is very hard to prove lower bounds on arbitrary polynomial-size circuits, which motivates studying the following subclass of circuits.

Definition 3.1. AC^0 *is the set of all constant depth, polynomial-size circuits with unbounded fan-in.*

We note that allowing unbounded fan-in is crucial for constant depth circuits; otherwise, the output of a depth c circuit could depend on at most 2^c bits of the input.

Example 3.2. *Consider a depth 2 circuit.*



This is a DNF (disjunctive normal form); we could similarly have made a CNF. However, note that having an AND gate as the parent of another AND gate is redundant as they could be combined (and similarly with ORs). Hence, all depth 2 circuits correspond to CNFs or DNFs; more generally, circuits typically have this alternating structure of ANDs/ORs layers.

We now show an important theorem regarding lower bounds on AC^0 circuits

Theorem 3.3 (Razborov-Smolensky)**.**

$$\text{MAJORITY} \notin AC^0$$

where MAJORITY is the function that outputs 1 if at least half the input bits are 1s, and 0 otherwise.

There are multiple proof strategies for this, one of which yields the following corollary

Corollary 3.4.

$$\text{PARITY} \notin AC^0$$

where PARITY is the function that outputs the sum of the input bits modulo 2.

This result is notable as it shows that these easy problems (which are even in **P**) cannot be computed by AC^0 circuits. This gives hope that we can eventually show that slightly harder problems, namely those in NP, cannot be computed by any polynomial circuits.

To prove this theorem, we want to find a complexity measure that is guaranteed to be small for AC^0 circuits, but is large for MAJORITY. This is a natural strategy to attempt; the cleverness comes in choosing the *right* complexity measure. The full proof of this theorem will be left for next lecture, but we establish some of the background now.

4 Polynomials

Informally, we will show that AC^0 can be well-approximated by low-degree polynomials (and that MAJORITY cannot be). We will consider polynomials over the finite field $\mathbb{F}_2$, which allows us to add algebraic structure to bitstrings $\{0,1\}^n$. This is useful, as it allows us to reuse functions $f : \{0,1\}^n \to \{0,1\}$ as functions $f : \mathbb{F}_2^n \to \mathbb{F}_2$.

Definition 4.1. *$\mathbb{F}_2$ is a finite field over two elements $\{0,1\}$. Multiplication is standard multiplication of these integers, and addition is standard addition modulo 2.*

We will now begin to formalize this idea.

Definition 4.2. *$\mathcal{P}_{n,d}$ is the set of all n-variate polynomials of degree at most d over $\mathbb{F}_2$*

Observation 4.3. *For $x_i \in \mathbb{F}_2$, $x_i^2 = x_i$. Hence, there is no need to ever have a power greater than one on a single variable within a monomial. In other words, we can assume without loss of generality that each monomial looks like x_1 or $x_1x_3x_4$, never $x_3^2x_4$.*

This allows us to write any polynomial $p \in \mathcal{P}_{n,d}$ as

$$p = \sum_{\substack{S \subseteq [n] \\ |S| \leq d}} C_S x^S$$

where $x^S = \prod_{i \in S} x_i$ and $C_S \in \mathbb{F}_2$.

Example 4.4. *$\mathcal{P}_{n,1}$ is the set of all n-variate polynomials of degree at most one, so for any $p \in \mathcal{P}_{n,1}$ we can write*

$$p = c_0 + \sum_{i \in [n]} c_i x_i$$

but since $c_i \in \mathbb{F}_2$ is either 0 or 1, and ignoring the shift c_0, we have $p = \sum_{i \in S} x_i$ for $S \subseteq [n]$. Thus, p is the sum of a subset of its input bits, and the sum over $\mathbb{F}_2$ is equivalent to the sum modulo 2 Hence, $\mathcal{P}_{n,1}$ is equivalent to PARITY and the negation of PARITY (since we can also have $c_0 = 1$).

Now that we have formalized polynomials, we seek to formalize the notion of well-approximated.

Definition 4.5. *A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is ε-approximated by a function $g : \mathbb{F}_2^n \to \mathbb{F}_2$ if*

$$\Pr_{x \in \mathbb{F}_2^n}[f(x) = g(x)] \geq 1 - \varepsilon$$

Lemma 4.6. *Let C be a size s, depth t AC^0 circuit. Then, there exists $p \in \mathcal{P}_{n,d}$ with $d = O(\log(s/\varepsilon)^t)$ such that p is an ε-approximator for C.*

We note that setting $t = O(1), s = \text{poly}(n)$, and assuming ε is a small constant gives us that $d = \text{poly}(\log n)$.

One may question why we settle for approximating with a polynomial, rather than finding an equivalent polynomial. We claim that this lemma gives an impressive reduction in the required polynomial degree compared to finding an exact polynomial equivalent.

Example 4.7. *Consider the simple OR function on n input bits, and suppose we want to exactly compute it. The natural choice is the polynomial $1 - \prod_{i \in [n]}(1 - x_i)$, which is a degree n polynomial. In fact, this is the lowest degree polynomial that exactly computes OR.*

It turns out that this is the worst case.

Fact 4.8. *Any $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be computed by a degree $\leq n$ $\mathbb{F}_2$ polynomial.*

This can be proven by induction on n. However, we saw that even for simple functions like OR, the worst case of degree n does arrive. Hence, this shows the value in the above lemma, as we get an exponential improvement in degree from n to $\text{poly}(\log n)$.

In the proof next lecture, we will actually prove a slight variation on the above lemma. This requires a couple more definitions.

Definition 4.9. *A probabilistic polynomial is a distribution over polynomials. A probabilistic polynomial has degree d if it is a distribution over $\mathcal{P}_{n,d}$.*

Example 4.10. *Consider the distribution which selects $x_1x_2 + x_3$ with probability 0.1, $x_1x_5x_4 + x_1x_4x_6$ with probability 0.4, and $x_1x_2x_3x_5x_6$ with probability 0.5. This is a probabilistic polynomial with degree 5.*

We define a variation on ε-approximators using these probabilistic polynomials:

Definition 4.11. *A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is ε-pointwise approximated by a probabilistic polynomial P if for all $x \in \mathbb{F}_2^n$,*

$$\Pr[P(x) = f(x)] \geq 1 - \varepsilon$$

or equivalently

$$\Pr_{p \sim P}[p(x) = f(x)] \geq 1 - \varepsilon$$

Next lecture, we will continue by showing that we can construct a ε-pointwise approximator for any AC^0 circuit, and that having a ε-pointwise approximator gives us an ε-approximator.