CAS CS 535: Complexity Theory

Lecturer: Mark Bun Fall 2026

Lecture Notes 13:

Karp-Lipton, Circuit Lower Bounds, Restricted Circuit Classes

Reading.

- Arora-Barak § 6.1-6.7

Last time: Intro to Circuits

A circuit family $C = \{C_n\}_{n=1}^{\infty}$ is an infinite sequence of circuits, where each $C_n : \{0,1\}^n \to \{0,1\}$ is built from $\wedge, \vee, \neg$ gates.

The class $\mathbf{P}_{/\mathbf{poly}}$ consists of languages that are decided by poly-size circuit families.

1 Circuits vs. Turing Machines

We saw last time that circuit families are at least as powerful as TMs because TMs can be compiled into circuits with low overhead in the compilation.

Theorem 1. *Let M be a TM running in time $T(n)$ where $\log T(n)$ is a space-constructible function. Then there exists a circuit family $C = \{C_n\}_{n=1}^{\infty}$ such that:*

1. *For every $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, we have $C_n(x) = M(x)$.*
2. *For every n, we have $|C_n| \leq \mathrm{poly}(T(n))$.*
3. *Uniformity: There exists a TM N running in time $\mathrm{poly}(T(n))$ and space $O(\log T(n))$ that prints descriptions of the circuits, i.e., $N(1^n) = \lfloor C_n \rfloor$.*

Corollary 2. $\mathbf{P} \subseteq \mathbf{P}_{/\mathbf{poly}}$.

How about the reverse containment? Is $\mathbf{P}_{/\mathbf{poly}}$ contained in $\mathbf{P}$? The answer is no, since (awkward terminology) $\mathbf{P}_{/\mathbf{poly}}$ can decide undecidable languages.

Fact 3. *Every unary language $L \subseteq \{1\}^*$ is in $\mathbf{P}_{/\mathbf{poly}}$.*

Proof. Each circuit C_n is just a constant indicating whether 1^n is in L. □

Fact 4. *There exists an undecidable unary language.*

Proof. Take your favorite undecidable language, e.g., the Halting problem, and encode it in unary. □

1.1 Uniformity and Non-uniformity

The key conceptual difference between Turing machines and circuits is that the former is a uniform model of computation (i.e., the same algorithm is used on inputs of every length), whereas the latter is a non-uniform model (i.e., a different algorithm is used on each input length.) In a sense, this is the only difference, as efficient TMs are equivalent to uniformly-generated circuit families, while arbitrary circuit families are equivalent to TMs that take "non-uniform advice."

Theorem 5. *A language $L \in \mathbf{P}$ if and only if there exists a logspace uniform circuit family computing L. (That is, there exists a logspace TM that on input 1^n outputs the description of the n'th circuit in the family.)*

Proof. The "only if" direction follows immediately from Theorem 1. For the "if" direction, suppose L is decidable by a circuit family generated by logspace TM N. Then the following poly-time TM decides L:

On input x:

1. Generate circuit $C_n = N(1^{|x|})$
2. Evaluate $C_n(x)$.

Logspace uniformity guarantees that C_n takes logspace, and hence polynomial time, to construct. Moreover, since the circuit has polynomial size, it takes polynomial time to evaluate. □

Now let's see how to "non-uniformize" Turing machines to get an equivalent characterization of $\mathbf{P}_{/\mathbf{poly}}$.

Theorem 6. *A language $L \in \mathbf{P}_{/\mathbf{poly}}$ if and only if L is decided in poly-time by a TM taking "polynomial advice." That is, there exists a TM M, a polynomial p, and an infinite sequence of strings $\{\alpha_n\}_{n=1}^{\infty}$, with $\alpha_n \in \{0,1\}^{p(n)}$ such that $x \in L \iff M(x, \alpha_{|x|}) = 1$.*

Proof. For the "only if" direction, let $L \in \mathbf{P}_{/\mathbf{poly}}$ be computed by a poly-size circuit family $\{C_n\}$. Define an advice sequence by $\alpha_n = \lfloor C_n \rfloor$. Then the following TM M with advice $\{\alpha_n\}$ decides L in poly-time:

On input x, α_n:

1. Construct circuit C_n corresponding to α_n
2. Evaluate $C_n(x)$.

For the "if" direction, let L be decided by a poly-time TM M using advice $\{\alpha_n\}$. For each n, use the transformation underlying Theorem 1 on $M(\cdot, \alpha_n)$ to obtain a poly-size circuit family. □

2 Karp-Lipton Theorem

Non-uniform advice bears a tantalizing similarity to the witnesses/certificates used to characterize $\mathbf{NP}$ languages. But the order of quantifiers makes a big difference! Specifically, a machine M decides a language L in each model if:

Non-uniform advice: $\exists \{\alpha_n\}_{n=1}^{\infty} \forall x \in \{0,1\}^* \qquad x \in L \iff M(x, \alpha_n) = 1$

Certificates: $\forall x \in \{0,1\}^* \qquad x \in L \iff \exists w\, M(x,w) = 1.$

That is, with non-uniform advice, the same advice string α_n has to work for all strings x of length n, whereas an NP certificate is allowed to depend on the particular instance x. At first glance, this may make non-uniform advice appear less powerful than NP certificates. But the upshot of the quantifier switch is that the same advice string equally helps certify non-membership in L. So the resources seem incomparable.

In fact, we already know that non-uniform advice can be more powerful than nondeterminism since the former can be used to solve undecidable problems. Thus, $\mathbf{P}_{/\mathbf{poly}} \not\subseteq \mathbf{NP}$. What about the other direction?

Theorem 7 (Karp-Lipton)**.** *If* $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$*, then* $\mathbf{PH} = \mathbf{\Sigma}_2^p$.

Before proving this, let's mention some interpretations:

- Taking the contrapositive, the Karp-Lipton Theorem says that if the polynomial hierarchy does not collapse, then $\mathbf{NP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$. So this can be taken as evidence that $\mathbf{NP}$ does not have poly-size circuits.
- Combined with the observation that $\mathbf{P}_{/\mathbf{poly}} \not\subseteq \mathbf{NP}$, this suggests that non-uniform advice and nondeterminism are incomparable computational resources.
- Given this evidence that $\mathbf{NP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$, a viable strategy for proving that $\mathbf{NP} \not\subseteq \mathbf{P}$ would be to actually prove the *stronger* statement that, say, SAT does not have poly-size circuits. Since circuits are nice combinatorial objects, this might be easier to prove than trying to reason about the TM complexity of SAT directly.

Proof. Suppose $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$. To show that $\mathbf{PH} = \mathbf{\Sigma}_2^p$, it suffices to show that $\mathbf{\Pi}_2^p \subseteq \mathbf{\Sigma}_2^p$. To establish this, it in turn suffices to show that the $\mathbf{\Pi}_2^p$-complete problem Π_2-$\mathsf{SAT} \in \mathbf{\Sigma}_2^p$. That is, we want to exhibit a Σ_2-type ATM determining the truth of formulas of the form "$\forall u \exists v \varphi(u,v)$."

The idea is as follows. Our assumption that $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ means that for every u, the subproblem $\exists v \varphi(u,v)$ can be computed by a small circuit C. Our Σ_2 machine will *first* existentially guess this circuit, then use universal guessing to check it.

So now let's carry out this idea in detail. We need to do a bit of technical preparation. First, recall the search-to-decision reduction for SAT, which says that if $\mathsf{SAT} \in \mathbf{P}$, then there is a poly-time algorithm that finds a satisfying assignment whenever one exists. This transformation works equally well for circuits. That is, if $\mathsf{SAT} \in \mathbf{P}_{/\mathbf{poly}}$, there exists a poly-size circuit family $\{C_{\mathsf{SAT},m}\}_{m=1}^{\infty}$ such that $C_{\mathsf{SAT},|\psi|}(\psi)$ outputs a satisfying assignment to ψ whenever one exists.

Let's also introduce the following piece of helpful notation. For a formula $\varphi(u,v)$, and a partial assignment a to the variables u, define $\varphi_{u=a}(v)$ to be the formula obtained by setting the variables u to assignment a. Now consider the following Σ_2-type TM:

On input a quantified formula $\forall u \in \{0,1\}^n \exists v \in \{0,1\}^n \varphi(u,v)$:

1. Existentially guess a circuit C (representing a guess for C_{SAT} of appropriate size to handle as input a formula $\varphi_{u=a}$)
2. Universally guess a string $a \in \{0,1\}^n$
3. Construct the formula $\varphi_{u=a}(v)$. Use the circuit to produce a candidate satisfying assignment $b = C(\varphi_{u=a})$.
4. Check that $\varphi(a,b) = 1$.

To see why this is correct, observe that

$$\begin{aligned}\forall u \in \{0,1\}^n \exists v \in \{0,1\}^n \varphi(u,v) &\iff \forall a \in \{0,1\}^n \varphi_{u=a} \in \mathsf{SAT} \\ &\iff \forall a \in \{0,1\}^n \varphi(a, C_{\mathsf{SAT}}(\varphi_{u=a})) = 1 \\ &\iff \exists C \forall a \in \{0,1\}^n \varphi(a, C(\varphi_{u=a})) = 1,\end{aligned}$$

which is the criterion that our ATM checks. □

3 Circuit Lower Bounds

Despite the fact that $\mathbf{P}_{/\mathbf{poly}}$ contains undecidable languages, the Karp-Lipton Theorem says it's reasonable to conjecture that $\mathbf{NP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$. So why might it be easier to prove this statement than to prove that $\mathbf{NP} \not\subseteq \mathbf{P}$ directly? The answer is that we have techniques beyond diagonalization and related arguments for proving circuit lower bounds, in particular, techniques that don't relativize.

Here's a simple (but dramatic) example of a circuit lower bound.

Theorem 8. *For every (sufficiently large) n, there exists a function $f : \{0,1\}^n \to \{0,1\}$ that requires circuit size $\geq 2^n/5n$.*

Note that this is essentially tight. You have a homework problem to show that every function is computed by a circuit of size $O(2^n/n)$. Actually, we have a very sharp understanding of the maximal circuit complexity of n-input functions; it's between $2^n/n$ and $(1+o(1))2^n/n$.

Proof. The proof is by a "counting argument." We're going to count the number of functions $f : \{0,1\}^n \to \{0,1\}$, and count the number of small circuits, and show that the former is larger than the latter.

1. How many n-bit functions $f : \{0,1\}^n \to \{0,1\}$ are there? Answer: 2^{2^n}.
2. How many (fan-in 2) circuits of size s are there? For each vertex $v = 1, \ldots, s$, there are:
   - $\leq s^2$ ways to determine the inputs to v
   - $\leq n + 3$ choices for what type of vertex v is (n inputs, $\wedge$, $\vee$, $\neg$)

   Thus, there are at most $(s^2(n+3))^s \leq 2^{4s \log s}$ circuits of size s.

If $s \leq 2^n/5n$, then we have

$$2^{4s \log s} \leq 2^{4(2^n/5n) \cdot n} \leq 2^{2^n \cdot (4/5)} < 2^{2^n}.$$

□

In fact, note that the ratio of the number of circuits to the number of functions is at most

$$\frac{2^{2^n \cdot (4/5)}}{2^{2^n}} = 2^{-2^n/5} \to 0$$

as $n \to \infty$, so "almost all" functions do not have circuits below this size.

4 Restricted Depth Classes

We know that most functions require huge circuits, but we'd really like to show that functions we care about (e.g., SAT, or any language in $\mathbf{NP}$) do as well.

Sadly, we do not yet have good techniques for proving general circuit lower bounds for "explicit" functions. But we do have techniques for studying circuits of small depth.

Definition 9. For each natural number k, define $\mathbf{NC}^k$ to be the class of languages decidable by circuits of size $\mathrm{poly}(n)$ and depth $O(\log^k n)$.

The class $\mathbf{NC} = \bigcup_{k=1}^{\infty} \mathbf{NC}^k$.

($\mathbf{NC}$ stands for "Nick's Class" named after Nicholas Pippenger.)

Definition 10. For each natural number k, define $\mathbf{AC}^k$ to be the class of languages decidable by circuits of size $\mathrm{poly}(n)$ and depth $O(\log^k n)$ using unbounded fan-in $\wedge$ and $\vee$ gates.

The class $\mathbf{AC} = \bigcup_{k=1}^{\infty} \mathbf{AC}^k$.

($\mathbf{AC}$ stands for "Alternating Class" after the connection to alternating TMs.)

Fact 11. *For every k, we have $\mathbf{NC}^k \subseteq \mathbf{AC}^k \subseteq \mathbf{NC}^{k+1}$.*

Proof. The first inclusion is automatic from the definitions. For the second inclusion, we use the fact that any gate with $\mathrm{poly}(n)$ fan-in can be replaced by an $O(\log n)$-depth tree. □

These classes are already interesting (and test the limits of our understanding) at low levels.

$\mathbf{NC}^0$: This class consists of languages decidable by constant-depth circuits. So observe that each output bit can only depend on a constant number of input bits.

This isn't very interesting for languages, but for multi-bit output functions, there is evidence that this class is powerful enough to do cryptography. (See [Appelbaum-Ishai-Kushilevitz06], "Cryptography in $\mathbf{NC}^0$".)

$\mathbf{AC}^0$: This class generalizes DNF and CNF (depth-2) to circuit families of arbitrary constant depth. Some basic arithmetic (addition, but not multiplication), approximate counting, and searching constant-width mazes are possible in $\mathbf{AC}^0$. Some important examples of functions not computable in $\mathbf{AC}^0$ are parity and majority.

$\mathbf{NC}^1$: This class is equivalent to poly-size formulas (circuits where each gate has fan-out 1). This is because formulas can be "balanced" recursively to bring their depth down to $O(\log(\text{size}))$. Sadly, the best known formula size bound for an explicit function is only about n^3.

4.1 Parallel Computation

Roughly, (uniform) circuits of size s and depth d capture what can be computed by s parallel, random-access processors running in "wall-time" d.

Therefore, (uniform) $\mathbf{NC}$ captures the languages decidable by a parallel computer with polynomially many processors running in polylogarithmic time.