CS 2401 - Introduction to Complexity Theory

Lecture #7: Fall, 2015

Lecturer: Toniann Pitassi

Scribe Notes by: Eric Bannatyne

1 Circuit Complexity

In this lecture we began discussing a number of topics related to circuit complexity. Boolean circuit families are a natural example of a *nonuniform* model of computation. In contrast with our usual definition of a Turing machine, which is a uniform model of computation in which a machine uses the same algorithm on all input lengths, a boolean circuit family computing a function includes a different circuit for every possible input size.

Recall that a boolean circuit C_n is a directed acyclic graph that has input nodes labeled with the literals $x_1, \ldots, x_n$ and $\neg x_1, \ldots, \neg x_n$, as well as internal nodes, or gates, labeled either $\wedge$ or $\vee$, and one output node. We can define what it means for a family of circuits to compute a boolean function f.

Definition Let $f : \{0,1\}^* \to \{0,1\}$ be a boolean function. A circuit family $\mathcal{C}_f = \{C_n\}_{n\in\mathbb{N}}$ computes f if for every $n \in \mathbb{N}$ and $x \in \{0,1\}^n$ we have $C_n(x) = f(x)$.

We say that f has polynomial sized circuits, or $f \in \mathbf{P/poly}$, if there is some constant c such that $|C_n| \leq n^c$ for all sufficiently large n, where $|C_n|$ denotes the number of nodes in C_n.

We also introduce the parity function, which will be an important example of a problem that occurs in circuit complexity.

Definition Let $x \in \{0,1\}^n$, then $\mathrm{PAR}_n(x) = 1$ if and only if $x_1 \oplus_2 \cdots \oplus_2 x_n \equiv 1 \pmod 2$, where $\oplus_2$ denotes exclusive or. Equivalently, $\mathrm{PAR}_n(x) = 1$ if and only if x has an odd number of ones.

1.1 Common Circuit Classes, $\mathbf{AC}^0$ and NC

In addition to $\mathbf{P/poly}$, we can define a number of related circuit complexity classes by restricting the types of circuits we can use to compute a given boolean function.

Definition A boolean *formula* is a boolean circuit in which every gate has fanout 1.

1. **DNF/CNF** consists of boolean functions that can be computed with polynomial-size, depth 2, unbounded fanin formulas, which correspond to traditional DNF and CNF boolean formulas.

2. $\mathbf{AC}^0$ is the class of all boolean functions that can be computed with polynomial-size, unbounded fanin, constant-depth circuits. $\mathbf{AC}^0_d$ consists of all those functions in $\mathbf{AC}^0$ that can be computed with circuits of depth d.

3. $\mathbf{NC}^i$ is the class of all boolean functions that can be computed with polynomial-size, fanin 2, depth $O(\log^i n)$ circuits. The class $\mathbf{NC}$ is the union of $\mathbf{NC}^i$ over all $i \in \mathbb{N}$. Functions in $\mathbf{NC}$ can also be viewed as those functions which can be computed efficiently in parallel.

The following chain of inclusions is known:

$$\mathbf{DNF/CNF} \subseteq \mathbf{AC}^0 \subseteq \mathbf{NC}^1 \subseteq \mathbf{NC}^2 \subseteq \cdots \subseteq \mathbf{NC} \subseteq \mathbf{P/poly} \tag{1}$$

It is conjectured that problems in $\mathbf{NP}$ do not have polynomial-sized circuit families, that is, that $\mathbf{NP} \not\subseteq \mathbf{P/poly}$. In particular, since $\mathbf{P} \subseteq \mathbf{P/poly}$, a proof of this conjecture would imply that $\mathbf{P} \neq \mathbf{NP}$. A primary aim of proving circuit lower bounds is to find some explicit function, ideally in $\mathbf{NP}$ (or at least $\mathbf{NEXP}$) that cannot be computed in one of the above circuit complexity classes.

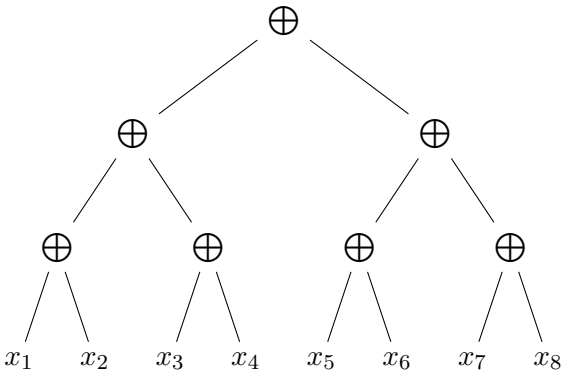
We can show, by a simple counting argument, that almost all functions on n bits *do not* have polynomial size circuits. Firstly, there are 2^{2^n} different boolean functions on n bits. Second, a circuit of size s can be represented as a string encoding an adjacency list in which 2 bits specify the type of each gate, and for each wire, $2 \log s$ bits specify the endpoints of the wire, requiring $s(2 + 2 \log s)$ bits in total. Thus there are at most $s2^{s(2+2\log s)}$ circuits of size at most s. Taking $s = 2^n/(10n)$, there are at most $2^{0.4^n+2n+2}$ circuits of size s, which is much smaller than 2^{2^n} for sufficiently large n. In particular, the value of the ratio

$$\frac{2^{0.4^n+2n+2}}{2^{2^n}}$$

decreases quickly to zero as n increases.

1.2 Circuit Complexity of Parity and Other Problems

We can show that n-bit parity is in $\mathbf{NC}^1$ by constructing, for each $n \in \mathbb{N}$, a balanced boolean circuit (which will have depth $O(\log n)$) to compute $\mathrm{PAR}_n(x)$ for every $x \in \{0,1\}^n$. Below is an example for $n = 8$.



In the above circuit, each XOR gate $x \oplus y$ can be rewritten as $(x \vee y) \wedge (\overline{x} \wedge \overline{y})$.

We can also show that any DNF formula computing PAR_n must have size exponential in n.

Theorem 1 *Any DNF formula computing* PAR_n *has size at least* $2^n/2$.

Proof Suppose f is some DNF formula that computes PAR_n on n bits. We can observe the following facts:

1. Every term in f must have size exactly n. This is because if a variable is missing from some term in f, then the final value of $\text{PAR}_n(x)$ will depend on that variable, since the value of that variable will determine whether the number of ones in x is even or odd.
2. Each term is satisfied by only one assignment to the bits of x.

Therefore the total number of terms in f must be greater than or equal to the number of inputs α such that $\text{PAR}_n(\alpha) = 1$, which is equal to $2^n/2$.

A similar argument also shows that parity requires exponential size CNF formulas. A harder result shows that PAR_n is not in $\mathbf{AC}^0$. Another major result by Ryan Williams shows that there is a function in **NEXP** that is not in **ACC**. Here, the class **ACC** can be thought of as "$\mathbf{AC}^0$ with counting," in which we also allow circuits to include gates that count modulo a fixed integer.

It's also possible to show that basic arithmetic operations, such as binary addition and multiplication, are in $\mathbf{NC}^1$. In the case of addition, the standard gradeschool algorithm requires us to maintain a carry bit at each step. This method requires a circuit of depth that is linear in the number of bits in the input to compute the carry bits. However, this can be made more efficient by using *carry lookahead*. For instance, if we want to add the following binary numbers:

$$1011011$$
$$1101011$$

we can determine whether a carry is necessary at a given position by looking at the pairs of corresponding digits in the inputs. In particular, the pair $\binom{1}{1}$ generates a carry bit, while $\binom{0}{0}$ absorbs a carry bit, and $\binom{1}{0}$ and $\binom{0}{1}$ propagate an existing carry bit. Thus the carry bits can be determined by a shallow circuit, and in particular a circuit of logarithmic depth.

1.3 Monotone Circuit Classes

In addition to our usual circuit complexity classes, we can also define an analogous set of *monotone* circuit classes $\mathbf{mAC}^0, \mathbf{mNC}^1, \mathbf{mNC}^2, \ldots, \mathbf{mNC}$ and $\mathbf{mP/poly}$. The definition of each of these classes is identical to its corresponding complexity class defined above, with the additional restriction that the boolean circuits used cannot include negations. Thus the input gates consist only of positive literals.

We say that a boolean function $f : \{0,1\}^n \to \{0,1\}$ is monotone if for any input $\alpha \in \{0,1\}^n$ such that $f(\alpha) = 1$, then $f(\alpha') = 1$ if $\alpha' \in \{0,1\}^n$ can be obtained from α by flipping some zero bits to be ones.

For example, the function $\text{CLIQUE}_{n,k}$, which takes as input the adjacency matrix of an undirected graph G on n vertices and outputs 1 iff G contains a clique of size k, is a monotone function. This is because if a graph contains a k-clique and we add a new edge (thereby flipping a bit in the adjacency matrix from 0 to 1), the resulting graph will still contain a clique of size k.

Just like with the circuit classes defined above, we have the following chain of inclusions for monotone circuit classes:

$$\mathbf{mAC}^0 \subseteq \mathbf{mNC}^1 \subseteq \mathbf{mNC}^2 \subseteq \cdots \subseteq \mathbf{mNC} \subseteq \mathbf{mP/poly}.$$

However, while it is unknown whether any of the classes in (1) are equal, it is known that each of these inclusions for monotone circuit classes is strict.

Given any boolean function f, we can derive a collection of monotone functions known as the *slice functions* of f.

Definition Let $f : \{0,1\}^n \to \{0,1\}$ be a boolean function. For each $i \in \{0,\ldots,n\}$, we can define the ith *slice* of f to be the function

$$f_i(x) = \begin{cases} 1 & \text{if } x \text{ has more than } i \text{ 1's} \\ f(x) & \text{if } x \text{ has exactly } i \text{ 1's} \\ 0 & \text{if } x \text{ has fewer than } i \text{ 1's} \end{cases}$$

for every $x \in \{0,1\}^n$.

In particular, it is easy to see that every slice f_i is monotone, regardless of whether f itself is monotone. One of the main reasons for studying slice functions is related to a result due to Berkowitz, establishing a relationship between the complexity of monotone and non-monotone circuits for slice functions. Namely, that a non-monotone circuit of size s computing a slice function $f : \{0,1\}^n \to \{0,1\}$ can be converted to a monotone circuit of size at most $2s + O(n \log^2 n)$. This is done by introducing the idea of a *pseudo-negation*, which enables us to emulate the behaviour of NOT gates using monotone circuits for slice functions. In particular, this result means that a superpolynomial lower bound for the monotone circuit size of a slice function would also imply the lower bound for non-monotone circuits.

1.4 $\mathbf{NC}^1$ and Polynomial Sized Formulas

It is also possible to view the class $\mathbf{NC}^1$ in terms of polysize formulas, without depth restrictions. This requires a useful lemma concerning binary trees.

Lemma 2 *If T is a binary tree of size $|T|$, then T contains a subtree T' such that*

$$\frac{1}{3}|T| \leq |T'| \leq \frac{2}{3}|T|.$$

Proof If either the left or right subtree of T is within the desired range, then we are done. Otherwise, let $T_1(0) = T_2(0) = T$. Then for each $k \geq 1$, let $T_1(k)$ (resp. $T_2(k)$) be the smaller (resp. larger) of the left and right subtrees of $T_2(k-1)$.

Since $|T_2(1)| > \frac{2}{3}|T|$, there must be some k such that $|T_2(k)| \leq \frac{2}{3}|T|$ but $|T_2(k-1)| > \frac{2}{3}|T|$. This is simply because $T_2(j+1)$ is a proper subtree of $T_2(j)$, and thus has at least one less node, for all j less than the height of T. To conclude the proof, we then have

$$|T_2(k)| = |T_2(k-1)| - |T_1(k)| - 1 > \frac{2}{3}|T| - \frac{1}{3}|T| - 1 = \frac{1}{3}|T| - 1.$$

Since $|T_2(k)|$ is an integer, it follows that $|T_2(k)| \geq \frac{1}{3}|T|$.

Using the above lemma, we can prove the following theorem.

Theorem 3 $\mathbf{NC}^1$ *is equal to the set of all boolean functions that can be computed by polynomial-sized formulas.*

Proof Let f be an $\mathbf{NC}^1$ (polysize, depth $O(\log n)$) circuit. Then we can convert f to a formula by duplicating subcircuits as necessary to obtain a tree that is equivalent to the original circuit f. Since f has depth $O(\log n)$, the resulting formula has size polynomial in n.

To prove the reverse direction, let f be a formula of size polynomial in n. We wish to construct a new circuit f' by "re-balancing" f to have depth $O(\log n)$. Let t_1 be a subformula of f such that $\frac{1}{3}|f| \leq |t_1| \leq \frac{2}{3}|f|$, and let $t_2(x_1, \ldots, x_n, y)$ be the original formula f with the subformula t_1 replaced with a new variable y. We can then rewrite f as

$$f = [t_1 \wedge t_2(1/y)] \vee \left[\overline{t_1} \wedge t_2(0/y)\right],$$

where $t_2(1/y)$ means "t_2 with 1 substituted for y", and similarly for $t_2(0/y)$. We then construct f' by recursively applying this same process on t_1, $t_2(1/y)$ and $t_2(0/y)$. Every application of this process increases the depth of the resulting formula by at most 3. However, since the size of the subtrees that this process gets recursively called on decreases by a factor of $\frac{2}{3}$, the total recursion depth will only be at most $\log_{3/2}|f|$. Since $|f|$ is polynomial in n, the recursion depth is $O(\log n)$, and so the resulting balanced formula will have depth $O(\log n)$.

2 Next Time

During the next lecture, we will show that $\text{PAR}_n \notin \mathbf{AC}^0$. The main idea will be to begin with an alleged small $\mathbf{AC}^0$ circuit for PAR_n, and apply a restriction $\rho : \{x_1, \ldots, x_n\} \to \{0, 1, *\}$, which assigns each variable to be either 0, 1, or unset ($*$). Let $\text{PAR}_n|_\rho$ be the parity function restricted to the inputs set by ρ. We then observe that the problem of computing $\text{PAR}_n|_\rho$ is still the problem of computing the parity function (or its negation) on the unset variables. Turning this argument into an actual lower bound for PAR_n will involve the use of the *switching lemma*.