E0 224 Computational Complexity Theory

Lecture 16

Lecturer: Chandan Saha *Scribe: Sandip Sinha*

October 8, 2014

Some remarks on boolean circuits:

1. Boolean circuits provide a finer level view of computation performed by Turing Machines.
2. Strong lower bounds for circuits imply lower bounds for TMs.
   (a) $\mathbf{NP} \nsubseteq \mathbf{P}_{/\mathbf{poly}} \Rightarrow \mathbf{P} \neq \mathbf{NP}$
   This is clear since we know that $\mathbf{P} \subset \mathbf{P}_{/\mathbf{poly}}$.
   Remark: If we assume $\mathbf{P} \neq \mathbf{NP}$, it is not clear if $\mathbf{NP} \nsubseteq \mathbf{P}_{/\mathbf{poly}}$ follows. An intuitive explanation for this is based on the non-uniformity of boolean circuits. For a language L to be in $\mathbf{P}$, there must exist a *single* poly-time TM which correctly decides membership of strings *of all sizes* in L. On the other hand, a language is in $\mathbf{P}_{/\mathbf{poly}}$ if there exists a family of circuits $\{C_n\}$ such that C_n decides membership of strings of length n in L. For instance, for the SAT problem, it might be the case that there is no single polynomial-time algorithm that decides SAT formulae of all sizes. However, for each natural number n, there may exist a poly-size (in n) circuit C_n which decides SAT formulae of size n. Then SAT is in $\mathbf{P}_{/\mathbf{poly}}$ - a possibility we do not know how to rule out.
   (b) It seems unlikely that $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ because of the Karp-Lipton Theorem:
   If $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ then $\mathbf{PH} = \Sigma_2^p$ (equivalently, $\mathbf{PH}$ collapses to level 2).

   Since circuits provide a finer view of computation performed by TMs, the hope is that one might be able to prove strong lower bounds for circuits.
3. Simple counting argument shows that there are "plenty" of boolean functions that are not in $\mathbf{P}_{/\mathbf{poly}}$.
   Question: Is there such a function in $\mathbf{NP}$? Finding such a function would imply $\mathbf{NP} \nsubseteq \mathbf{P}_{/\mathbf{poly}}$, from which it would follow that $\mathbf{P} \neq \mathbf{NP}$. However, such efforts have failed till date. The best known circuit lower bound for a language in $\mathbf{NP}$ is only $(5 - o(1))n$.
   (Iwana, Lachish, Morizumi and Raz (2005)).
4. Naturally, people started looking at special (but interesting) classes of circuits.
5. Classes $\mathbf{NC}$ and $\mathbf{AC}$:

Definition (NC): For every i, a language L is in $\mathbf{NC}^i$ if L can be decided by a family of circuits $\{C_n\}$ where C_n has poly(n) size and depth $O(\log^i n)$. The class $\mathbf{NC}$ is $\bigcup_{i\geq 0} \mathbf{NC}^i$.

Definition (AC): The definition of class $\mathbf{AC}^i$ is quite similar to the above definition of $\mathbf{NC}^i$, except that gates are allowed to have unbounded fan-in. The class $\mathbf{AC}$ is $\bigcup_{i\geq 0} \mathbf{AC}^i$.

Remark: $\mathbf{NC}^0 \subseteq \mathbf{AC}^0 \subseteq \mathbf{NC}^1 \subseteq \mathbf{AC}^1 \subseteq \mathbf{NC}^2 \subseteq ...$
It is obvious that $\mathbf{NC}^i \subseteq \mathbf{AC}^i$ for all i. To show that $\mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$, we observe that a gate with unbounded (but poly(n)) fan-in can be replaced by an equivalent tree of gates with fan-in k (using a complete k-ary tree structure) thus incurring only a logarithmic increase in the depth of gates (since log (poly(n)) = $O(\log n)$).

It is known that $\mathbf{NC}^0 \subsetneq \mathbf{AC}^0 \subsetneq \mathbf{NC}^1$. The first inclusion is obvious since $\mathbf{NC}^0$ cannot even capture the AND operation of n boolean variables, where n is a parameter. The second inclusion is a non-trivial result which was proved by showing that PARITY is not in (non-uniform) $\mathbf{AC}^0$. PARITY is clearly in $\mathbf{NC}^1$, since we can design a circuit in the form of a binary tree, which recursively computes the parity at a node by computing the parity of its left and right sub-trees and then computes the parity of these two bits. The circuit has $O(\log n)$ depth, and can also be shown to be logspace uniform (we define logspace uniform circuit families later).

References: (PARITY is not in non-uniform $\mathbf{AC}^0$)
Furst, Saxe, Sipser (1981)
Ajtai (1983)
Håstad (1986) (Håstad's Switching Lemma)
Note: The result was proved by Furst, Saxe, Sipser (1981) (see also Ajtai (1983), Yao (1985)). However, Håstad's Switching Lemma can be used to give optimal parameters for the result.
Håstad (2014) - "On the Correlation of Parity and Small-Depth Circuits" - This paper shows that PARITY is not only hard to solve, but also hard to approximate with circuits with constant depth and unbounded fan-in, since the output of such a circuit differs from the PARITY function on a significant proportion of inputs. The precise result shown is the following:
"The correlation of a depth-d unbounded fanin circuit of size S with parity of n variables is at most $2^{-\Omega(n/\log S)^{d-1}}$."

6. Without any uniformity constraint, even $\mathbf{AC}^0$ would include undecidable languages. Any unary language would be in $\mathbf{AC}^0$. To see this, let L be a unary language. Then for each $n \in \mathbb{N}$, if $1^n \in L$, take C_n to be the $\wedge$ gate with all the n bits as input, and if $1^n \notin L$, take C_n to be a trivial circuit which outputs 0. Then the family of circuits $\{C_n\}$ decides L. In particular, the following undecidable language UHALT (described in Arora and Barak's Book - "Computational Complexity: A Modern Approach"), which is the unary encoding of the halting problem, would be in $\mathbf{AC}^0$:
UHALT = $\{1^n : n$'s binary expansion encodes a pair $< M, x >$ such that M halts on input $x.\}$

Therefore, it is clear that some uniformity restriction must be imposed on these classes to make them comparable to the classes we are familiar with, such as **P** and **L**.

7. What kind of uniformity restrictions should be imposed?

   One option is to impose poly-time uniformity restriction, i.e. it should be possible to compute the circuit in poly-time. Another option is log-space uniformity (defined below). It turns out from the analysis of the Cook-Levin Theorem that a language is in **P** if and only if it has log-space uniform circuits of polynomial size.

   Definition (logspace-uniform circuit families): A circuit family $\{C_n\}$ is logspace uniform if there is an implicitly logspace computable function mapping 1^n to the description of the circuit C_n.

   We impose the restriction that circuit families for languages in **NC** and **AC** are log-space uniform. Subsequently, by **NC** (or **AC**), we will always refer to log-space uniform **NC** (or **AC**).

8. This naturally leads us to the notion of **P**-completeness.

   Definition: A language $L \subseteq \{0,1\}^*$ in **P** is said to be **P**-complete if for every other language $L' \in \mathbf{P}$, L' is log-space reducible to L (denoted $L' \leqslant_\ell L$).

9. Why do we use log-space reductions to define **P**-completeness? This is because the complexity questions we have in mind while defining **P**-completeness are:
   - $\mathbf{L} = \mathbf{P}$?
   - $\mathbf{NC} = \mathbf{P}$?

   It seems intuitively clear that the first complexity question is captured by implicit log-space reduction. However, it is not immediately obvious why log-space reduction should capture the second question as well. This is discussed below.

10. Suppose $L \subseteq \{0,1\}^*$ is **P**-complete. Then

    (a) $\mathbf{P} = \mathbf{L}$ iff $L \in \mathbf{L}$

    Proof. We know $\mathbf{L} \subseteq \mathbf{P}$. Assume $L \in \mathbf{L}$, and let L' be a language in **P**. Since L is **P**-complete, there is an implicit log-space reduction f from L' to L. To decide L', we can use the reduction f to L and then run the log-space TM for L. Clearly, this can be done in log-space, and so $L' \in \mathbf{L}$. Thus $\mathbf{P} \subseteq \mathbf{L}$, which implies $\mathbf{P} = \mathbf{L}$.
    For the converse, assume $\mathbf{P} = \mathbf{L}$. Since L is **P**-complete, it is in **P**, and hence in **L**. □

    (b) $\mathbf{P} = \mathbf{NC}$ iff $L \in \mathbf{NC}$
    Proof idea: Assume $L \in \mathbf{NC}$, and let L' be a language in **P**. Our objective is to show that $L' \in \mathbf{NC}$, i.e. to show that there is a family of circuits $\{D_n\}$ of poly(n) size and polylog(n) depth which decides L', and an implicitly log-space computable function which maps 1^n to the description of D_n.
    Let the circuits $\{C_n\}$ deciding L be of depth $O(\log^i n)$, where n is input length. Since L is **P**-complete, there is an implicit log-space reduction f from L' to L, such that

$x \in L'$ iff $f(x) \in L$. Since the size of $f(x)$ is polynomial in $|x|$, and $f(x)$ is the input string for $C_{|f(x)|}$, the depth of this circuit is $O(log^i|f(x)|) = O(log^i|x|)$. It remains to be shown that the implicit log-space computation of $f(x)$ from x can be captured by a family of circuits satisfying the criteria of class **NC**.

11. It turns out that $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{NC}$.
To prove the second inclusion, it suffices to show that PATH $\in$ **NC** (we will solve this as an assignment problem). Let G be a directed graph with adjacent matrix A.
Observation: Let s, t be vertices in G. The entry (s, t) in A^n gives the number of edge sequences from s to t having atmost n edges.
We can compute A^n from A in $O(\log n)$ steps by repeated squaring (compute A^2, A^4, A^8 and so on). The proof has to be completed by showing that this can actually be done in implicit log-space.

12. Class $\mathbf{ACC^0}$ (or **ACC**):
$\mathbf{ACC^0}$ stands for 'Alternating Circuit with Counters of Constant Depth'.

Definition (Mod_k gate):
On input $x_1, x_2, ..., x_n$, this gate outputs 0 if $\Sigma_{i=1}^n x_i = 0 \bmod k$. Otherwise, it outputs 1.

Definition (Class $\mathbf{ACC}(m_1, m_2, ..., m_l)$ for some fixed $m_1, m_2, ..., m_l$ in $\mathbb{N}$):
We say a language $L \subseteq \{0,1\}^*$ is in $\mathbf{ACC}(m_1, m_2, .., m_l)$ if it can be decided by a family of circuits $\{C_n\}$, where C_n has:

- poly(n) size
- constant depth
- Mod_{m_1}, Mod_{m_2}, ..., Mod_{m_l} gates along with $\vee$, $\wedge$ and $\neg$ gates
- unbounded fan-in

Definition (Class $\mathbf{ACC^0}$):
We say a language $L \subseteq \{0,1\}^*$ is in $\mathbf{ACC^0}$ if $L \in \mathbf{ACC}(m_1, m_2, ..., m_l)$ for some $m_1, m_2, ..., m_l \in \mathbb{N}$.

Result (Razborov and Smolensky (1985)):
For every two distinct primes p and q, $\text{Mod}_p \notin \mathbf{ACC}(q)$.
Since PARITY $\in \mathbf{ACC}(2)$, the above result implies, in particular, that PARITY cannot be decided by an **ACC** circuit having Mod_3 gates.

Theorem. NEXP does not have non-uniform **ACC** circuits of quasi-polynomial size.

Reference: Ryan Williams (2010-11) - "Non-Uniform ACC Circuit Lower Bounds"

13. Monotone Circuits:
Monotone circuits are circuits composed of only $\wedge$ and $\vee$ gates but no $\neg$ gates.

Definition (Monotone function): A function $f : \{0,1\}^n \to \{0,1\}$ is called monotone if $x \preceq y \Rightarrow f(x) \leqslant f(y) \forall x, y \in \{0,1\}^n$
where the notation $x \preceq y$ means $x_i = 1 \Rightarrow y_i = 1$, i.e. if the i^{th} bit of x is 1, then the i^{th} bit of y is 1 (for instance, if $x = 001$ and $y = 101$, then $x \preceq y$).

Fact: Every monotone function can be computed by a monotone circuit (possibly of exponential size). Every monotone circuit computes a monotone function.

Proof. Clearly, if f is a function that can be computed by a monotone circuit, it has to be monotone since we cannot invert any of the inputs. So, if $f(x) = 1$ for some $x \in \{0,1\}^n$, then for any $y \in \{0,1\}^n$ such that $x \preceq y$, since $y_i = 1$ for all i for which $x_i = 1$, $f(y) = 1$. For the converse, let $f : \{0,1\}^n \to \{0,1\}$ be a monotone function. If f is constant, it has a trivial monotone circuit - one that outputs 0 or 1 on all inputs. So assume f is not constant. Then $f(0^n) = 0$ and $f(1^n) = 1$. For each path on the boolean hypercube from 0^n to 1^n on the boolean n-dimensional cube which respects the partial order $\preceq$ (i.e. we only move from x to y if $x \preceq y$), consider the first string x such that $f(x) = 1$. The circuit consists of (for each path) an $\wedge$ gate with all the positions of x which are 1 as input, combined with $\vee$ gates over all valid paths. □

Let $\text{CLIQUE}_{k,n} : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ be the function that takes as input the adjacency matrix of an n-vertex graph G and outputs 1 iff G has a clique of size k. Then $\text{CLIQUE}_{k,n}$ is a monotone function since adding more edges to a graph which has a k-vertex clique cannot destroy the clique that is already present.

Theorem. (Razborov 1985):
There exists a constant $\epsilon > 0$ such that $\forall k \leqslant n^{\frac{1}{4}}$, the smallest monotone circuit computing $\text{CLIQUE}_{k,n}$ must have size $2^{\epsilon\sqrt{k}}$.

14. Natural proofs: A hurdle to proving circuit lower bounds?
A natural strategy to prove lower bounds for a function f is to show that there's a predicate P such that
$P(f) = 1$ and
$P(g) = 0$ for all functions g computed by a n^c-size circuit for some constant c.
If a predicate P satisfies the above property then we say P is n^c-useful.

Theorem. (Razborov-Rudich)
Suppose there exist sub-exponentially hard one-way functions. Then there is a constant c for which there is NO natural n^c-useful predicate.

We give definitions of natural n^c-useful predicates and one-way functions below.
Definition (Natural n^c-useful predicate P):
A natural n^c-useful predicate P satisfies the following criteria:

(a) P is n^c-useful.

(b) *Constructiveness*: There is a $2^{O(n)}$-time algorithm that, given $f : \{0,1\}^n \to \{0,1\}$ as input (in truth-table format), outputs $P(f)$. It should be noted that the algorithm is allowed to run for time which is polynomial in the size of the truth table.

(c) *Largeness*: At least $\frac{1}{n}$ fraction of all functions $f : \{0,1\}^n \to \{0,1\}$ satisfy the predicate. This property captures the notion that there is a non-trivial fraction of functions which satisfy the predicate.

Remark: There are proofs of circuit lower bounds which avoid the Constructiveness property of the predicate.

Definition (One-way function):
A poly-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ is one-way if for every (probabilistic) poly-time algorithm A, there is a negligible function $\epsilon(n)$ such that
$Pr_{x \in_R \{0,1\}^n} [A(f(x)) = x' s.t. f(x') = f(x)] \leqslant \epsilon(n)$
The probability in the above definition is over both the random choice of A and that of x.

A function $\epsilon : \mathbb{N} \to \mathbb{R}$ is called negligible if $\epsilon(n) = n^{-\omega(1)}$.
A one-way function f is said to be sub-exponentially hard if there exists $\epsilon > 0$ such that any algorithm which computes the inverse of f has a running time of atleast 2^{n^ϵ}, where n is input size.

Example: Multiplication of integers is a possible candidate for a sub-exponentially hard one-way function since it is easy to multiply integers but extremely hard to factor an integer (into its prime factors). The best known factoring algorithm is the Number Field Sieve Algorithm (a heuristic algorithm) which has a running time of $2^{O\left((\log n)^{\frac{1}{3}} (\log \log n)^{\frac{2}{3}}\right)}$ on input n. It should be kept in mind that the input size is $O(\log n)$.
If there exists $\epsilon, 0 < \epsilon < 1$, such that factoring does not have a $2^{O(\log^\epsilon n)}$-time algorithm, then multiplication is a sub-exponentially hard one-way function.

References

[1] S. Arora and B. Barak, "Computational Complexity: A Mordern Approach", Cambridge University Press, 2009

[2] J. Håstad, "On the Correlation of Parity and Small-Depth Circuits", SIAM Journal on Computing, 2014

[3] V. Arvind, Lecture Notes on "Monotone Circuit Lower Bounds" in the course Complexity Theory 2 (Aug-Dec 2006) scribed by Ramprasad Saptarishi (`http://www.cmi.ac.in/~ramprasad/lecturenotes/raz_clique.pdf`)