CSC 2429 – Approaches to the P vs. NP Question and Related Complexity Questions
Lecture 2: Switching Lemma, AC^0 Circuit Lower Bounds

Lecturer: Toniann Pitassi
Scribe: Robert Robere

Winter 2014

1 Switching Lemmas

In this lecture we discussed a further application of random restrictions: namely, the switching lemma and strong lower bounds for bounded-depth circuits. The switching lemma is a tool (introduced in it's modern form by Håstad [5]) for transforming CNFs into DNFs and vice versa, possibly after applying a random restriction. We will state the switching lemma in terms of *decision trees*:

Definition 1.1. Let X be a set of variables. A *decision tree* T on X is a model of computation defined as follows. There is an underlying full binary tree T, in which each internal vertex v is labeled with variable $x_v \in X$ and each leaf is labeled with 0 or 1. Given an input x (equivalently, an assignment to each of the variables in X) we take a walk on the tree as follows: we start at the root, and if at any point we are standing at a vertex v, we step to the left child of v if $x_v = 0$ and to the right child of v if $x_v = 1$. The output of T on x is label of the leaf reached on this walk, which we denote as $T(x)$. We say that the decision tree T computes a boolean function f if $T(x) = f(x)$ for all inputs x. The *depth* of a decision tree is the length of the longest path from the root to a leaf.

Decision tree depth is an important measure of the complexity of a function. Recall that a k-DNF on a set of variables X is a "disjunction of conjunctions" of positive or negated literals in X in which each conjunct has at most k literals. We call the conjuncts in a DNF *terms*. A k-CNF is defined similarly as a conjunction of disjunctions where each disjunct has at most k literals. For the sake of distinguishing them from DNFs, we call the disjuncts *clauses*. It turns out that the decision tree depth and CNF/DNF size are closely related.

Proposition 1.2. *If a function f has a decision tree of depth t then it is representable as both a t-DNF and a t-CNF.*

Proof. We construct the DNFs and CNFs from the decision tree by constructing the clauses out of the paths in the decision tree which reach the 0s and the 1s of the functions, respectively. We show how to construct a t-DNF, and the construction for the t-CNF is analogous. Each clause in the t-DNF will correspond to a 1-path in the decision tree, and so the t-DNF will be true if and only if a 1-path can be followed in the decision tree. Let T be a depth-t decision tree computing f, and let P be any path from the root of T to a leaf labelled with 1. Let $x_1, x_2, \ldots, x_\ell$ be the literals appearing on the path P, where the literal x_i is negated if we moved left at the vertex and it is positive otherwise. For each such path P, we add a term $x_1 \wedge x_2 \wedge \dots x_\ell$ to our t-DNF. If an input x is accepted by our decision tree, then the term corresponding to the accepting path in the tree will be accepted by the DNF, and the converse direction holds as well. Clearly $\ell \leq t$ for each path P in the tree, and so the resulting DNF has at most t literals per term.

The construction for t-CNFs proceeds analogously, except we instead consider the paths to the 0s of the decision tree, and add a clause with the *negations* of the literals followed on the paths. Intuitively, this t-CNF is true if and only if none of the 0 paths are followed in the decision tree. □

Conversely, given a t-DNF (or t-CNF) F we can construct a decision tree computing the same boolean function as F called the *canonical decision tree of* F, which we denote by $T(F)$. Suppose that F is an r-DNF defined on a variable set X. A *restriction* of F is a function $\rho : X \to \{0, 1, *\}$ which is interpreted as partial assignment to the variables in F, i.e. if a variable $x \in X$ has $\rho(x) = *$ then that variable is considered to be unset. For a restriction ρ we denote by $F|_\rho$ the DNF obtained by simplifying the DNF F using the assignment to variables in ρ. To construct $T(F)$ choose a term in our DNF F and let $y_1, y_2, \dots, y_r$ be an ordered list of literals appearing in the term. Create a complete binary tree of depth r, where each node at depth i is labeled with y_i, and note that each path in the tree leading from the root to a leaf ℓ corresponds to a restriction ρ_ℓ on X where we set all of the variables according to the path. If this restriction sets the term to 1, label the leaf with a 1. Otherwise, choose another term and simplify it using ρ_ℓ, and apply the same reasoning again.

In the switching lemma, we will be interested in converting DNFs to CNFs (and vice-versa) of roughly the same size, possibly after restricting the values of some variables. To this end, we will introduce some more notation. If we have a fixed set of n variables X, we let $\mathcal{R}^\ell$ denote the set of all restrictions of X which leave exactly ℓ variables unset. It's not hard to see that

$$|\mathcal{R}^\ell| = \binom{n}{\ell} 2^{n-\ell},$$

since we must choose the ℓ variables to leave unset and then choose a boolean assignment for the rest of them.

Finally, we present Håstad's Switching Lemma.

Lemma 1.3 (Håstad's Switching Lemma)**.** *Let s be a positive integer, let F be an r-DNF and let $\rho \sim \mathcal{R}^\ell$ denote a restriction chosen uniformly at random from $\mathcal{R}^\ell$, with $\ell < n/2$. Then*

$$\Pr_{\rho \sim \mathcal{R}^\ell}[T(F|_\rho) \textit{ has height at least } s] \leq \left(\frac{8\ell r}{n}\right)^s.$$

Proof. We prove the lemma using the combinatorial argument originally due to Razborov [6]. Let B be the set of bad restrictions in $\mathcal{R}^\ell$. We will give a one-to-one map f from each restriction $\rho \in B$ to a tuple $(\rho\pi, x)$ where $\rho\pi$ is a restriction on $\ell - s$ variables and x is a bitstring containing $s(\log r + 1)$ bits. First we show that if such a map f exists then the lemma holds. There are $|\mathcal{R}^\ell| = \binom{n}{\ell} 2^{n-\ell}$ restrictions on ℓ variables, and so (using our assumed injection f) the probability of choosing a bad restriction uniformly at random will be

$$\alpha = \frac{|B|}{|\mathcal{R}^\ell|} \leq \frac{|\mathcal{R}^{\ell-s}| 2^{s(\log r + 1)}}{|\mathcal{R}|^\ell} = \frac{\binom{n}{\ell-s} 2^{n-(\ell-s)} (2r)^s}{\binom{n}{\ell} 2^{n-\ell}}.$$

We can upper bound the fraction of binomial coefficients like so. Expanding and cancelling the $n!$ term leaves

$$\frac{\binom{n}{\ell-s}}{\binom{n}{\ell}} = \frac{\ell!(n-\ell)!}{(\ell-s)!(n-(\ell-s)!)},$$

and using the inequalities $\ell!/(\ell - s)! \leq \ell^s$ and $(n-\ell)!/(n-(\ell-s)!) \leq 1/(n-\ell)^s$ yields an upper bound of $(\ell/(n-\ell))^s$. Substituting this back in to α and using $\ell \leq n/2$ to simplify we get

$$\alpha \leq \left(\frac{4\ell r}{n-\ell}\right)^s = \left(\frac{8\ell r}{n}\right)^s.$$

Now, we will construct the injection f. The canonical decision tree for $F|_\rho$ has depth at least s, so choose an assignment π to the remaining literals of $F|_\rho$ so that $F|_{\rho\pi}$ is the constant 0 function. The assignment π induces a path from root to a 0 leaf in $F|_\rho$ of length at least s. If necessary, trim some of the last assignments so that π assigns values to exactly s variables.

Let $t_1, t_2, \ldots, t_k, \ldots$ be the terms remaining in $F|_\rho$, and we will break π up into k sub-restrictions $\pi_1\pi_2\cdots\pi_k$ (we define k in a moment). For $i < k$, the restriction π_i will be obtained by following the path π from the root of the decision tree to the beginning of the subtree where we assign all of the variables in the term t_i. We then define π_i to be the assignment obtained from following the path π through the portion of the subtree that assigns values to all of the variables remaining in t_i. In other words, π_i assigns values to all of the variables in the term $t_i|_{\pi_1\pi_2\cdots\pi_{i-1}}$, leaving the term with value 0. Note that there is a unique assignment σ_i which assigns values to exactly the same variables as π_i, except the term $t_i|_{\pi_1\pi_2\cdots\pi_{i-1}\sigma_i}$ will have value 1. For $i = k$, which will be the last term assigned variables by π, we will obtain the restriction π_k in the same way, except now it may not set all of the remaining variables in the term t_k. This will not bother us – we define σ_k to assign values to the same variables as π_k, except that σ_k will agree with the unique 1 assignment to t_k.

Now, we define the map $f : \rho \to (\rho\sigma_1\sigma_2\cdots\sigma_k, x)$, where x will be a bitstring that will help to make the function one-to-one. To see how to define x, first suppose we were given the restriction $\rho\sigma_1\sigma_2\cdots\sigma_k$. We will define a process which takes x and replaces the partial restriction σ_i with π_i one by one. After the process we will have the restriction $\rho\pi_1\cdots\pi_k$, along with the name and assignment of every variable in π. We can then use this information to remove the restriction $\pi = \pi_1\pi_2\cdots\pi_k$, leaving us with ρ.

Given $\rho\sigma_1\sigma_2\cdots\sigma_k$, it is easy to determine which term t in F is associated with σ_1: it is simply the first term that is not assigned the value 0, since all of the earlier terms must be assigned 0 by ρ. Using this fact, we will add to the bitstring x the indices of variables in t which are assigned to by σ_1 (and π_1), as well as the bit that π_1 assigns to that variable. This requires at most $|\pi_1|(\log r + 1)$ bits, where $|\pi_1|$ is the number of variables assigned by π_1. Once we determine the term t in F which is assigned to by σ_1, we can use these auxiliary bits to determine the partial assignment π_1 from σ_1. This means that we can "remove" the assignment σ_1 and instead consider the assignment $\rho\pi_1\sigma_2\cdots\sigma_k$, and continue in the same fashion. That is, in the ith step we do the following: we will have the partial assignment $\rho\pi_1\pi_2\cdots\pi_{i-1}\sigma_i\cdots\sigma_k$, we assume that the bitstring x contains the indices of variables in the term t_i assigned to by σ_i, as well as the value of the assignment π_i to those variables, and we use this information to replace σ_i with π_i in the partial assignment $\rho\pi_1\pi_2\cdots\pi_{i-1}\sigma_i\cdots\sigma_k$.

It's easy to see that the length of the bitstring x must be $(|\pi_1|+|\pi_2|+\cdots+|\pi_k|)(\log r+1)) = s(\log r+1)$. Once we perform this process and replace every partial assignment σ_i with the corresponding assignment π_i, we will know the name and assignment of every variable assigned by $\pi_1\pi_2\cdots\pi_k$. Removing these assignments from the partial restriction $\rho\pi_1\pi_2\cdots\pi_k$ will give us ρ, and so the map f must be one-to-one. □

2 AC^0 Circuit Lower Bounds

Now we will move on to one of the primary applications of the switching lemma, which is very strong lower bounds on the size of AC^0 circuits computing the parity function. The complexity class AC^0 is defined to be the set of all decision problems computable by families of constant depth, polynomial size circuit with unbounded fan-in over the basis $\{\wedge, \vee, \neg\}$ (we will refer to such circuits as "AC^0" circuits).

The parity function $\oplus_n : \{0,1\}^n \to \{0,1\}$ takes n bits as input and outputs a 1 iff an odd number of the bits in the input are 1. Equivalently,

$$\oplus_n(x_1, x_2, \cdots, x_n) := (x_1 + x_2 + \cdots + x_n) \mod 2.$$

The next proposition standardizes the form of AC^0 circuits.

Proposition 2.1. *Let C be a depth d circuit with M gates computing some function f. Then there exists a depth $2d$ circuit C' with $(4M)^{2d}$ gates computing f such that the following holds:*

1. *All of the $\neg$ gates are at the input layer*
2. *The $\wedge$ and $\vee$ gates are alternating from layer to layer, and*
3. *The circuit C' is a tree – each gate (except for the input gates) have fanout* 1.

We will use the switching lemma to prove that any constant depth circuit computing parity must have exponentially many gates, thus showing parity is not in AC^0. Our strategy is simple: we will take the AC^0 circuit C and apply the last proposition to turn it into a tree. The bottom two layers of this new circuit are a collection of CNFs (or DNFs) on the input variables. The switching lemma says that after applying a random restriction to a k-CNF we can, with high probability, write it as an equivalent k-DNF. A simple union bound shows that we can find a random restriction which works simultaneously for all of the CNFs in the collection, so we can "switch" the bottom two layers of the circuit from a CNF representation to a DNF representation after applying a restriction without blowing up the bottom fanout. Since the AC^0 circuit is alternating, this allows us to reduce the depth of the tree by 1, since we can then merge the $\vee$ gates together. If the circuit has depth d, then applying this switching argument d times will reduce the circuit to computing a constant function, but in doing so we will not restrict all of the variables. Since the parity function is only constant if we restrict *all* of the variables, we get a contradiction.

Theorem 2.2. *Parity is not computable in* AC^0.

Proof. Let C be a depth d AC^0 circuit computing parity on n variables, and let M be the number of gates in C (assume that C has the form in Proposition 2.1). If the gates at depth 1 in C are $\vee$ gates, add "dummy" fan-in $\wedge$ gates with fan-in 1 between the variables and the depth 1 gates (or add dummy $\vee$ gates if the depth 1 gates in C are $\wedge$ gates). This increases the depth of C by 1, but now all of the depth 2 gates in C compute bounded-fanin DNFs or CNFs.

First, suppose w.l.o.g. that the depth-2 gates in C are $\vee$ gates, which now compute "1-DNFs" on the input variables. Let $s = n^{1/d}$. Apply a random restriction on $\ell = n/16$ variables to C; by the switching lemma, the probability that we cannot re-write any DNF rooted at some depth-2 gate g with an s-CNF is at most 2^{-s}. If there are m_1 gates at the second level in C, then the probability that any of the corresponding DNFs cannot be re-written as an s-CNF is at most $m_1 2^{-s}$ by a union bound. Otherwise, we can switch the DNF with an s-CNF and reduce the depth of the circuit by 1 by merging all of the depth-2 gates into the layer above.

Continue in this way $d-1$ times — at each step we have an s-CNF (or s-DNF) at the bottom of the circuit and we apply a random restriction on $\ell = n/16s$ variables. If this is the ith application of the switching lemma and there are m_i gates at depth 2 in the circuit C, then after applying the random restriction we will be able to switch all of the CNFs with DNFs with probability at most $m_i 2^{-s}$.

After we have chosen and applied the $d-1$ random restrictions, by another union bound the probability that at any point we are unable to convert an s-DNF into an s-CNF or vice versa is at most

$$(m_1 + m_2 + \cdots + m_{d-1})2^{-s} \leq M2^{-s}.$$

Suppose by way of contradiction that $M < 2^s$, so that $M2^{-s} < 1$ and so we can find such a sequence of restrictions described above by the probabilistic method. What remains after applying these restrictions is a single conjunct or disjunct F with s literals. There are

$$n/16 + (d-2)n/16s = n/16 + (d-2)n/16n^{1/d} = n/16 + (d-2)n^{1-1/d}/16$$

variables unrestricted, which is less than $n-1$ for any constant d and sufficiently large n. Note that we can make F constant by choosing any literal appearing in F and fixing it appropriately (to 0 if F is a conjunct and 1 if F is a disjunct). However, there are other variables left unset, and since the parity function is not constant under any restriction of less than n variables we have arrived at a contradiction. It follows that $M > 2^s = 2^{n^{1/d}}$, which is clearly superpolynomial even when accounting for the growth from applying Proposition 2.1. $\square$

In fact, a similar argument will prove strong lower bounds for AC^0 circuits computing parity only approximately. For $k \leq 1/2$, say that a circuit C computes a boolean function f with advantage k if

$$\Pr_{x \sim \{0,1\}^n}[C(x) = f(x)] = 1/2 + k.$$

In other words, the circuit C beats randomly guessing the output of the function f on a k-fraction of the inputs. A more careful argument than will establish the following:

Theorem 2.3 (See [5])**.** *Let $d > 0$ be some positive integer constant and $M \leq 2^{s/2}$. Any circuit C with M gates and depth d with unbounded fan-in computes parity with advantage at most $2^{\Omega(-n/s^{k-1})}$.*

3 Applications

There are several other major applications of the switching lemma (and the strong AC^0 lower bounds) in complexity theory.

Pseudorandom generators for AC^0 Pseudorandom number generators are objects central to the question of *derandomization*. The area of derandomization studies the cases in which we can remove a randomized algorithm's reliance on random bits while still maintaining efficiency and correctness. Using the average-case parity lower bound above (Theorem 2.3), Nisan and Wigderson [4] showed that the randomized analogue of AC^0 can be derandomized in polylogarithmic space. That is, there is a *deterministic* algorithm using polylogarithmic space which can "supply" pseudorandom bits to a randomized AC^0 circuit.

Fourier Concentration In a fascinating work, Linial, Mansour and Nisan [3] studied functions computable by AC^0 circuits by using Fourier Analysis. The Fourier expansion of a Boolean function is the representation of the function as a low-degree polynomial. In their paper, Linial et al. showed that for any function in AC^0, the large coefficients in the functions Fourier expansion appear almost entirely on the lower-order coefficients. Among other things, this allows the derivation of a learning algorithm for functions in AC^0 running in quasipolynomial time.

AC^0-SAT and $\#\mathsf{AC}^0$-SAT While the work presented in the lecture here focused on lower bounds, one can also naturally ask about *upper bounds*. One such problem is the following: given an AC^0 circuit C, find an input x on which the circuit C outputs a 1. There is a simple brute-force $\mathsf{poly}(|C|)2^n$ algorithm for solving this problem, but one can also attempt to use the restricted structure of the circuit to try and get a more efficient algorithm. In [2], Impagliazzo et al. used the switching lemma to give such an algorithm for depth d, size cn AC^0 circuits which achieves a run time of $|C|2^{(1-\mu)n}$, where $\mu \geq 1/O(\log c + d \log d)^{d-1}$. Their technique also gives an improved algorithm for the much harder question of counting the number of satisfying assignments to an AC^0 circuit.

Bounded-Depth Frege proof lower bounds *Proof Complexity* is an area of research in computational complexity theory studying the lengths of proofs for propositional tautologies. A proof is formally phrased in terms of a "proof system", which is a prescription for what the lines of the proof look like, and what rules of inference are allowed. Proof complexity was originally introduced as an attack on the NP vs coNP question, which can be rephrased as asking whether there exists a proof system in which every tautology has a proof of polynomial length in the description of the tautology. A modified version of the switching lemma was used by Pitassi, Beame and Impagliazzo [1] to give exponential lower bounds on the size of proofs for the pigeonhole principle in a proof system called *Bounded-Depth Frege*.

References

[1] Toniann Pitassi, Paul Beame, Russell Impagliazzo. Exponential Lower Bounds for the Pigeonhole Principle. Computational Complexity 3:97-140 (1993)

[2] Russell Impagliazzo, William Matthews, Ramamohan Paturi. A satisfiability algorithm for AC^0. In the proceedings of SODA 2012.

[3] Nathan Linial, Yishay Mansour, Noam Nisan. Constant depth circuits, Fourier transform, and learnability. J. ACM 40(3):607-620 (1993)

[4] Noam Nisan and Avi Wigderson. Hardness vs Randomness. Journal of Computer and System Sciences 49(2):149-167 (1994)

[5] Johan Håstad. Computational limits of small depth circuits. Ph.D. Thesis, Massachusetts Institute of Technology, 1987.

[6] A.A. Razborov. An equivalence between second order bounded domain bounded arithmetic and first order bounded arithmetic. In Arithmetic, Proof Theory and Computational Complexity, pp. 247-277. Oxford University Press, 1993.

[7] Paul Beame. A switching lemma primer. Manuscript.