CS 710: Complexity Theory 10/11/2011

Lecture 10: Circuit Lower Bounds

Instructor: Dieter van Melkebeek Scribe: Li-Hsiang Kuo

Last time we introduced nonuniform computations, and nonuniform models such as Boolean circuits, branching programs, and uniform models with advice. Today we are mainly focus on Boolean circuits. We will address the complexity of some problems we are interested in.

1 Boolean Circuits

Definition 1. $C_L(n) =$ *smallest size of a Boolean circuit deciding* L *on* $\{0,1\}^n$

Theorem 1. $C_L(n) = O(\frac{2^n}{n})$

Theorem 2. $(\exists L) C_L(n) = \Omega(\frac{2^n}{n})$

Proof. Let s denote the number of binary gates. For each of the s gates, we can pick a variable or some other gate as input, and each gate has at most 2 inputs. So, the number of circuits of size at most s is at most $(c(s+n)^2)^s$, where c is some constant that takes care of different type of gates. Since we can map circuits to Boolean functions, this is also the maximum number of Boolean functions computable with at most s gates. We know that 2^{2^n} is the number of Boolean functions on n variables, and $(c(s+n)^2)^s = 2^{O(s \log s)}$. By setting $s = d\frac{2^n}{n}$ for a sufficiently small constant d, $(c(s+n)^2)^s = 2^{O(s \log s)} = 2^{O(d\frac{2^n}{n} \cdot (n \log d - \log n))} << 2^{2^n}$. The claim then follows. □

Both theorems also holds for branching programs. For Theorem 2 the situation is similar since Boolean circuits can easily simulate branching programs. For Theorem 1 this takes extra work.

Theorem 3. $BP_L(n) = O(\frac{2^n}{n})$

2 Problems in NP

- Upper bounds: No better results than general.
- Lower bounds:
  - $C_L(n) = \omega(n)$ is open for any $L \in$ NP.
  - $BP_L(n) = \Omega(\frac{n^2}{\log^2 n})$ is best known for $L \in$ NP.

The above are unconditional results. However, we can show that if NP has polynomial size circuits, then PH collapses to the second level.

Theorem 4. NP $\subseteq$ P/poly $\Rightarrow \Sigma_2^p = \Pi_2^p$

Proof. For a language L in Π_2^p, we have

$$x \in L \iff (\forall y \in \{0,1\}^{|x|^c})(\exists z \in \{0,1\}^{|x|^c})R(x,y,z)$$

where R is a predicate that can be decided deterministically in time, say, linear in its combined input.

Since $(\exists z \in \Sigma^{|x|^c})R(x,y,z)$ is a Σ_1^p predicate on input $< x, y >$, we can reduce it to a SAT instance and by the hypothesis, there exists a circuit C_{SAT} that is of size polynomial in the running time to decide R, and so of size polynomial in the size of x. Letting f denote the reduction, we can replace the Σ_1^p predicate with $C_{SAT}(f(x,y))$. We would like to rephrase the formula above roughly as follows: Does there exist a circuit solving SAT such that for all strings y, the circuit accepts $f(x,y)$?

Let V be the following recursive predicate:

(1) **if** φ has at least 1 variable **then** $C_{SAT}(\varphi) \iff C_{SAT}(\varphi|_{x_1 \leftarrow 0}) \vee C_{SAT}(\varphi|_{x_1 \leftarrow 1})$
(2) **else** $C_{SAT}(\varphi) \iff \varphi$ *is true*

where x_1 is the first unset variable of φ.

We can transform the right-hand side of the above to:

$$(\exists C_{SAT})(\forall y \in \Sigma^{|x|^c})[C_{SAT}(f(x,y)) \wedge (\forall \varphi \; of \; size \leq n^c)[V(C_{SAT}, \varphi)]]$$

where C_{SAT} is a circuit of polynomial size with n^c inputs.

Essentially, we guess a circuit C_{SAT} and the combined predicate checks if C_{SAT} accepts $f(x,y)$ and whether or not C_{SAT} is a valid circuit solving SAT.

Since we are evaluating polynomial size circuits, and we evaluate n times, V takes polynomial time to check. The second universal quantifier above can be merged with the first, and then we have a single polynomial time verifiable predicate. So, we now have a Σ_2^p formula. Note that our hypothesis is crucial in that if NP does not have polynomial size circuits, then V will always fail. □

We would expect that at least for complicated functions like those that capture NP-complete problems, we can prove non-trivial lower bounds. The following shows that if we go up a little bit higher in the hierarchy, there is some non-trivial lower bounds.

Theorem 5. $(\forall c > 0)(\exists L \in \Sigma_2^p \cap \Pi_2^p)$ *with* $C_L(n) = \Omega(n^c)$

Proof. We claim that $\exists y \in \{0,1\}^{n^d+1}$, where y encodes the inclusion or exclusion of the first $n^d + 1$ strings of a language Y, such that none of the circuits is consistent with it.

Intuitively, with each bit in y, we can "kill" half of the remaining consistent circuits. That is, we choose each bit in y by doing the opposite of the majority vote of the circuits that are consistent with the previous bits of y. The number of circuits of size $\leq n^c$ is $\leq (n^c + n)^{2n^c} \leq 2^{n^d}$ for some $d \geq 0$. By doing against the majority vote of the circuits, in each bit in y there is at most half of the circuits remains. n^d bits leaves ≤ 1 consistent circuit and $n^d + 1$ bits leaves no consistent circuit. By counting, there exist a setting of the "Lexicographically first" string in $\{0,1\}^{n^{d+1}}$ such that if we interpret this as a language, this language cannot be decided by any circuits of size $\leq n^c$.

Now, we must show that there is such a language Y in NP. The explicit construction of this language that we have described is not necessary in NP, it requires the majority vote counts on exponentially many circuits.

So, we can only assume that this language Y exists. And, in PH we have the power of guessing. However, this Y may not be unique, to make it unique, we can encode this as:

$\exists y \in \{0,1\}^{n^d+1}$

(1) ($\forall C$ of size $\leq n^c$)C and y disagree on one of the first n^d+1 strings

(2) ($\forall z < y$) (1) fails for z

(3) x is one of the first n^d+1 strings and the xth bit of y is set

This accepts precisely the language L described above. But, the negation of (2) means that we end up with Σ_3^p instead of Σ_2^p. However, using today's previous result we have two cases, either NP $\subseteq$ P/ poly in which case $\Sigma_3^p = \Sigma_2^p$, or NP $\nsubseteq$ P/ poly in which case their is some other language $L \in$ NP with $C_L(n) = \Omega(n^c)$ for all c. □

3 Constant-Depth Circuits

Because we have been unable to prove lower bounds for NP-complete problems in the general setting, we focus our attention on a restricted model - namely constant-depth circuits. Typically we talk about constant-depth circuit we do not count the exponential fan-in. Today we only focus on the class of languages decidable by constant-depth circuits of polynomial size.

Definition 2. *Binary addition:* $L = \{< x, y, z > | x + y = z\}$

Proposition 1. *The decision variant of binary addition has constant-depth circuits of polynomial size.*

Proof. In this proof, all strings are indexed from the right. To determine the ith bit of the sum, we only need to look at the ith bits of the summands and determine if there is a carry from the bits in position $(i-1)$. We first introduce some notation. We will label each column from 1 up to $i-1$ depending on if it either: *g*enerates a carry bit, *t*ransmits a carry bit, or *s*tops a carry bit. If both input bits in the column are 1, the column is labeled g; if only one bit is 1, it is labeled t; and if both bits are 0, the column is labeled s.

For there to be a carry from the $(i-1)^{st}$ column, there must be a g at some point followed by zero or more t columns. In other words, to determine if there is a carry into the ith position, our job is reduced to detecting if the above string is of the form $t^*g\{s,g,t\}^*$. First of all, we use an OR to guess the length of t^*g, and the number of possible lengths are at most linear in the input size. For each length j, we need an AND to determine if the $(i-j+1)$th symbol on the string is a g, and XORs to ensure that the symbols after it are ts, and then we do a big AND over the XORs and the AND. So, at the first level, we have an OR, at the second we have ANDs, and at the third we have ANDs and XORs. Since XORs can be implemented in constant depth using ANDs, ORs and NOTs, the overall circuit has constant depth of polynomial size. □

We now sketch a proof that PARITY requires exponential size to be computed by constant-depth circuits.

Definition 3. *PARITY* $= \{x :$ *x has an odd number of 1s*$\}$

We also denote PARITY on n variables as $\bigoplus_n$.

Exercise 1. *The decision variant of binary multiplication is equivalent to PARITY.*

Theorem 6. $C_d(\bigoplus_n) = 2^{\Omega(n^{\frac{1}{d-1}})}$

The proof we present uses the following tool.

Definition 4 (Random Restrictions)**.** *A p-random restriction on n variables is a random function $\rho : \{x_1, \ldots, x_n\} \to \{*, 0, 1\}$, such that for each i, independently, $Pr[\rho(x_i) = *] = p$ and $Pr[\rho(x_i) = 1] = \frac{1-p}{2} = Pr[\rho(x_i) = 0]$. If $\rho(x_i) = *$ then we leave x_i as a variable. Otherwise we set it to the result of $\rho(x_i)$.*

Proof sketch for Theorem 6. We can apply p-random restriction with $p = \frac{n^{\frac{1}{d-1}}}{n}$. With high probability it can reduce the depth of the circuit by one at a time. Repeat this method we turn depth-d circuits into depth-2 circuits. Then apply depth-2 lower bound. □

Proposition 2. $C_2(\bigoplus_n) = \Theta(n \cdot 2^{n-1})$.

Proof. For depth 2, we can easily prove an exponential lower bound. By assumption, the circuit is either a DNF or a CNF. Let us assume that it is a DNF. Each of the AND terms check for a setting of variables such that $\bigoplus_n = 1$. Thus they must contain all n variables. Otherwise, we can flip a variable and at least one AND will not be able to detect the difference. There are 2^n possible n-variable AND terms in a DNF formula, and we only need half of them as only half of them check for $\bigoplus_n = 1$. Since each AND must be of size n, and there must be 2^{n-1} of them, we get the bound as stated. □

4 Next Time

We will continue by giving an alternate proof on PARITY using polynomial approximations. The bound that we will get is $C_d(\bigoplus_n) = 2^{\Omega(n^{\frac{1}{2d}})}$, which is not as tight as the previous result, but the advantage is that it applies to circuits with gates other than AND, OR, NOT.

Acknowledgements

In writing the notes for this lecture, I perused the notes by Chris Hopman for lecture 8 from the Spring 2010 offering of CS 710 and the notes by Seeun William Umboh for lecture 8 from the Spring 2007 offering of CS 810.