6.841: Advanced Complexity Theory Fall 2012

Lecture 3: Sep 13, 2012

Prof. Dana Moshkovitz *Scribe: Vincent Liew*

1 Overview

In the last lecture we examined relativization. We found that a common method (diagonalization) for proving non-inclusions between classes of problems was, alone, insufficient to settle the P versus NP problem. This was due to the existence of two orcales: one for which $P = NP$ and another for which $P \neq NP$. Therefore, in our search for a solution, we must look into the inner workings of computation.

In this lecture we will examine the role of Boolean Circuits in complexity theory. We will define the class $P_{/poly}$ based on the circuit model of computation. We will go on to give a weak circuit lower bound and prove the Karp-Lipton Theorem.

2 Boolean Circuits

Turing Machines are, strange as it may sound, a relatively high level programming language. Because we found that diagonalization alone is not sufficient to settle the P versus NP problem, we want to look at a lower level model of computation. It may, for example, be easier to work at the bit level. To this end, we examine boolean circuits. Note that any function $f : \{0,1\}^n \to \{0,1\}$ is expressible as a boolean circuit.

2.1 Definitions

Definition 1. *Let $T : \mathbb{N} \to \mathbb{N}$. A T-**size circuit family** is a sequence $\{C_n\}$ such that for all n, C_n is a boolean circuit of size (in, say, gates) at most $T(n)$.*

$SIZE(T)$ is the class of languages with T-size circuits.

Definition 2. *$P_{/poly}$ is the class of languages decided by polynomial size circuits: $\bigcup_k SIZE(n^k)$*

2.2 Remarks

These are classes of **non-uniform computation**: for different input lengths, the computation is different. In this case, we have a different circuit for each input length. This is in contrast to **uniform computation**, for example, having a fixed Turing Machine acting all inputs regarless of length.

This type of computation can be funny. For example, any unary language (strings all of the form 1^n) is contained in $P_{/poly}$: since there is only one string per input length, if an input is in the

language we may assign it a circuit which always outputs 1 and otherwise assign a circuit which always outputs 0. But the class of unary languages includes undecidable languages!

2.3 An Equivalent Definition for $P_{/poly}$

There is a way to make the circuit model of computation a uniform one: we restrict the families of circuits to those which can be output by a Turing Machine. So we obtain an alternative characterization of P:

Definition 3. $P = \{$*Languages for which there exists a Turing Machine which, when given the input* 1^n*, outputs a circuit* C_n *in polynomial time, where* $\{C_n\}$ *is a polynomially sized circuit family for the language.*$\}$

We can also make the usual Turing Machine model of computation non-uniform. To do this, we give it a polynomially sized advice string which depends only on the length of the input. There is no constraint on what kind of function this advice can be; if we wish, it could even be undecidable.

Claim 4. *The class of languages decided by Turing Machines equipped with an advice string is equivalent to* $P_{/poly}$*.*

Proof. We simulate $P_{/poly}$ by giving our Turing Machine advice describing the circuit which computes the correct decision function for the given input length. The Turing Machine can simulate this circuit in polynomial time.

Conversely, we know that given a computation, there exists a boolean circuit which can simulate it. Since the advice given to the Turing Machine is a fixed string, we simply give a circuit simulating the Turing Machine computation with the advice string hardwired into the circuit. ■

Now that we have established this equivalence, we can see why we named our class $P_{/poly}$. The subscript denotes the type of advice (in this case, polynomially sized) that we give to our computer (in this case, a polynomial time Turing Machine).

3 Circuit Lower Bounds and the Karp-Lipton Theorem

In 1980, Karp and Lipton published a paper which essentially stated that instead of asking if $NP \nsubseteq P$, we should instead ask if $NP \nsubseteq P_{/poly}$. There were two main reasons for this:

- Computers are, at the hardware level, more similar to the circuit model than the Turing Machine model
- $NP \nsubseteq P_{/poly} \rightarrow NP \nsubseteq P$, since $P \subseteq P_{/poly}$

In the paper, they proved that the showing $NP \nsubseteq P_{/poly}$ is not much harder than showing $NP \nsubseteq P$.

3.1 An Easier Version of $NP \not\subseteq P_{/poly}$

Oftentimes when faced with a difficult problem, we can instead look to solve an easier version. In this case, we can examine a subclass of problems in $P_{/poly}$ and a larger class than NP.

Theorem 5. *$\Sigma_3 \not\subseteq SIZE(n^k)$ for all k.*

Note that on the left, we essentially gave our NP machine additional quantifying power and on the right, we resticted the circuit size to a fixed k. In essence, we have made the competition between the two machines very unfair, and this is what makes this problem easier to solve.

Proof. We first argue that there exist, for each n, functions without $SIZE(n^k)$ circuits. Say we have a function $f : \{0,1\}^n \to \{0,1\}$. The truth table corresponding to this function has size 2^n. We restrict our attention to those functions whose nonzero entries of this table only appear in the first n^{k+1} entries. It is easy to see that there are $2^{n^{k+1}}$ such functions. And given a number s, there are $2^{O(slogs)}$ size s circuits (we can specify a size s circuit with a string of size $O(slogs)$, specifying the wires coming in and out of the s gates).

Setting $s = n^k$ and noting that for large n, $2^{n^{k+1}} > 2^{O(n^k log n^k)}$, it follows that there exists a function family which is not decided by a circuit family in $SIZE(n^k)$. This family of functions describes a language not contained in $SIZE(n^k)$.

Now we show that this language is contained in Σ_3. We will exhibit a Σ_3 machine which (upon a length n input) searches for the lexicographic first function $f : \{0,1\}^n \to \{0,1\}$ which is not decidable by circuits in $SIZE(n^k)$ and mimicks its behavior. To do so, we need only give an appropriate boolean sentence which describes the language given by these functions:

$$x \in L \Leftrightarrow \exists f \forall C_f((\bigvee_{x_0} C_f(x_0) \neq f(x_0)) \wedge (f(x) = 1) \wedge (\forall(f' <_{lex} f)\exists C_{f'} \bigwedge_{x_0} C_{f'}(x_0) = f'(x_0)))$$

Let's parse this sentence. Note that all the quantifiers are legal since they are over polynomially sized sets. The first quantifier asks to find a particular instance of a function that we want. The second is a condition over all size-n circuits. The big-OR is over all possible inputs x_0, and asks that for at least one of them the circuit does not match the function. The middle condition is that the function must take the value 1. The last condition imposes the lexicographic condition: for all the functions lexicographically smaller than f, there exists a circuit which simulates it.

That the above sentence works suffices to show that this language is in Σ_3, and therefore $\Sigma_3 \not\subseteq SIZE(n^k)$.

3.2 The Karp-Lipton Theorem

It would be nice if we could prove that $NP \not\subseteq P \to NP \not\subseteq P_{/poly}$? It turns out that we can prove something similar to this, but weaker.

Theorem 6. *(Karp-Lipton Theorem) $PH \neq \Sigma_2 \to NP \not\subseteq P_{/poly}$*

Proof. Suppose $NP \subseteq P_{/poly}$. We will show that this implies $PH \subseteq \Sigma_2$. Note that $PH \subseteq \Sigma_2$ and $\Pi_2 \subseteq \Sigma_2$ are equivalent; if the latter were true then by swapping quantifier order, we have that $\Sigma_3 = \Sigma_2$, collapsing PH.

By the assumption that $NP \subseteq P_{/poly}$, there exists a polynomially sized circuit family, $\{C_n\}$, which decides SAT. This implies that there exists a polynomial size circuit family $\{C*_n\}$ which, given a boolean formula ϕ, finds a satisfying assignment for it. We construct $\{C*_n\}$ in the following way: we choose a free variable and substitute 1 or 0 for it, then run the decider. If the decider says that one of these boolean formulas has a satisfying assignment, repeat this process on another free variable. When there are no more free variables, we will have substituted a satisfying assignment into ϕ.

let $L \in \Pi_2$. Then $x \in L \leftrightarrow \forall u \exists v \phi(x, u, v) = 0$.

Then $\exists C * \forall u \phi(x, u, C * (x, u)) = 0$. That is, there exists a circuit which outputs an unsatisfying assignment for ϕ for any u.

Then $L \in \Sigma_2$ and the theorem is proven.

References

[1] R. Karp, R. Lipton, *Some Connections Between Nonuniform and Uniform Complexity Classes*, STOC '80 Proceedings of the twelfth annual ACM symposium on Theory of computing, 302-309, 1980.