CMPT 407: Computational Complexity **Summer 2017**

Lecture 4: Randomized Computation (6 - 9 June)

Lecturer: Valentine Kabanets *Scribe: Lily Li*

4.1 Review

Theorem 4.1 $\mathsf{EXP} \subset \mathsf{PolySize} \implies \mathsf{EXP} = \Sigma_2^p$.

Proof: For all L in EXP there exists a TM M, which runs in time $2^{n^c} = t$ for an input x of size n. Imagine a $t \times t$ grid which describes the operation of M, where each row is a configuration of M. This transcript is valid if and only if all windows (three consecutive cells in row i and the associated cell in row $i+1$) are consistent. Consider the function $T : [t] \times [t] \to \Sigma^*$ where $T(i,j) =$ cell j at time i. Since we assumed $\mathsf{EXP} \subset \mathsf{PolySize}$ all we need to do is show that $T \in \mathsf{EXP}$ (due to the Church-Turing thesis). Well, that's pretty obvious, simply execute the TM M. Now show that $T \in \Sigma_2$ as follows: $\exists C, \forall i, j$: window (i,j) (in the tableau) is consistent and the tableau ends in an accepting state. ■

Note: (by IKW) it is possible to generalize this implication for NEXP, namely $\mathsf{NEXP} \subset \mathsf{PolySize} \implies \mathsf{NEXP} = \Sigma_2$. Proving this is quite a bit more difficult and requires more tool.

4.2 Circuits

Let us consider the set of inclusion of circuit complexity:

$$\mathsf{AC}_0 \subset \mathsf{TC}_0 \subset \mathsf{NC}_1 \subset \mathsf{PolySize}$$

Where

$$\mathsf{NC}^i = \{L : L \text{ is decidable by a family of } \mathsf{PolySize} \text{ circuits of depth } O(\log^i n) \text{ with gates } \wedge, \vee, \neg\}$$

$$\mathsf{TC}^i = \{L : L \text{ is decidable by a family of depth } O(\log^i n) \text{ circuits of using unbound fan-in } Maj, \neg \text{ gates}\}$$

$$\mathsf{AC}^i = \{L : L \text{ is decidable by a family of depth } O(\log^i n) \text{ circuits using unbounded fan-in } \wedge, \vee, \neg \text{ gates}\}$$

4.2.1 NC_1

Here we will consider mainly the class of NC_1, that is all languages with logarithmic depths boolean circuits. As it turns out polysize circuits of log-depth are the same as formulas of log-depth. The reasoning is not trivial but quite simple nonetheless. A boolean formula is almost the same as a binary tree except for loops where one node is the input of several nodes below it in the graph. This does not cause a problem, because even if you duplicate the lower nodes to remove loops you will get at worst a complete log-depth binary tree, which is polynomial in size.

Another equivalence of note is: languages computable by polysize formulas is the same as languages computable by logdepth circuits.

Claim 4.2 $\mathsf{NC}_1 = \mathsf{PolySize}$ *formula.*

Proof: $\mathsf{NC}_1 \subseteq \mathsf{PolySize}$ formula is trivial. Now lets attempt to show th other direction, $\mathsf{PolySize}$ formula $\subseteq$ NEXP. Now a normal expansion of a formula F in $x_1, ..., x_n$ might be of depth $O(n)$. But if you think about it you realize that there are not a lot of "stuff" so a long path can be restructured to be made shorter and wider. In particular cut F into two pieces F_1 and F_2 each of depth approximately half. Let F_2 be a subformula of F_1. If we substitute F_2 with a variable z in F_1 as $F_1(x_1, ..., x_n, z)$ then there is a logical equivalent between:

$$F = (F_1(x_1, ..., x_n, 0) \wedge F_2(x_1, ..., x_n)) \vee (F_1(x_1, ..., x_n, 0) \wedge \neg F_2(1, x_2, ..., x_n))$$

(this should remind you of the way of polytime conversion from CNF to DNF). By repeated applications you can turn the originally $\mathsf{PolySize}$ formula into a logdepth circuit. ■

4.2.2 Valiant's Challenge

Find an explicit function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $O(n)$ and depth $O(\log n)$. That is, it is still an open problem to find a problem in PH with super linear circuit complexity.

Consider some examples of problems in NC_1.

Example 4.3 *Consider boolean matrix multiplication where we are given two $n \times n$ matrices A and B and told to compute their product $AB = C$. Each entry $C[i,j] = \bigvee_{k=1}^{n} A[i,k] \wedge B[k,j]$. Since we are or-ing together n ands, we can construct a $\log n$ circuit for each cell.*

Example 4.4 *Let $a_1, a_2, ..., a_n$ be n, n digit numbers. We want to show that this problem in the domain of Valiant's Challenge. So we need to demonstrate a $O(\log n)$ circuit of $O(n)$ size to solve this problem. The algorithm here requires a trick (actually it is the same trick as was used in prune and search).*

If you as two numbers as usual, each addition requires $\log n$ depth circuitry so in aggregate adding n numbers takes $\log^2 n$ depth. Instead we will do the 3 for 2 trick. That is instead of adding two numbers to get one number, we will add three numbers to get two numbers. In particular the two outputs will be the usual addition ignoring carries and the carry as a number. Each layer will only need constant layers. And each layer has only a fraction, namely $\frac{2}{3}$rd, of the numbers in the layer before. At most $O(\log n)$ layers required.

4.2.3 TC_0

Definition 4.5 *A majority gate is defined as follows: $Maj_n : \{0,1\}^n \to \{0,1\}$ where*

$$Maj_n(x_1, ..., x_n) = \sum x_i \geq \frac{n}{2}$$

Theorem 4.6 *Finding the parity of n numbers is in TC_0.*

Proof: First we need to construct a threshold function. Given n input bits we want to know if the sum is greater than or equal to a specified k. Formally define $Thr_{n,k}(x_1, ..., x_n) = 1$ if and only if $\sum_{i=1}^{n} x_i \geq k$. We can do this using dummy variables as follows: for example if we wanted a $Thr_{3,1}$ we would use $Maj_4(1, x_1, x_2, x_3)$. This way if any $x_i = 1$ then the sum, including the dummy variable $x_0 = 1$ adds to 2. Using two threshold gates, we can specify the "exactly k gate" as: $Exc_{n,k}(x_1, ..., x_n) = 1$ if and only if $\sum_{i=1}^{n} x_i = k$. Thus, using constant depth we can check if $n = 1, 3, 5, ...$ to see if n is odd or even. ■

4.3 AC_0

Theorem 4.7 *The addition of two n bit numbers is in* AC_0.

Proof: Let the input to the circuit be $a = a_{n-1} \cdots a_0$ and $b = b_{n-1} \cdots b_0$ and the output be $c = c_n \cdots c_0$. Note that $c_i = a_i + b_i + carry_i$ where $carry_i$ is the carry from position $i-1$ into position i. $carry_i = 1$ if there exists some $j < i$ such that $a_i = b_i = 1$ and for all $j \leq k < i$ either $a_k = 1$ or $b_k = 1$. The constant dept circuit is constructed from the formula $c_i = \exists j, 0 \leq j < i, \forall k, i > k > j : (a_j \wedge b_j) \wedge (a_k \vee b_k)$. Since we have unbounded fan-in $\wedge$ and $\vee$ gates, we can use these for $\forall$ and $\exists$ respectively. ■

Note: as it turns out the class AC_0 is exactly the set of all first order formulas!

Theorem 4.8 $\mathsf{Parity}_n \notin \mathsf{AC}_0$.

Proof: We want to find a distinguishing property which separates problems in AC_0 and Parity. The property we want is: given $f(x_1, ..., x_n)$, f can be make constant by fixing k variables where $k < n$. Note that Parity does not have this property. We will show that every problem in AC_0 have this property. But in order to do this we need the probabilistic method.

Define a probability distribution over partial assignments. Argue that there exists a random partial assignment will make the AC_0 function constant.

Consider function $f(x_1, x_2, ..., x_n)$ and take a parameter $p \in [0, 1]$. The random p-restriction ρ is as follows: $\forall 1 \leq i \leq n$, assigned independently at random,

$$x_i = \begin{cases} \star & \text{with probability } p \\ 0 & \text{with probability } \frac{1-p}{2} \\ 1 & \text{with probability } \frac{1-p}{2} \end{cases}$$

Lemma 4.9 *(Hastad's Switching Lemma) Let f be any k-CNF on $x_1, ..., x_n$. For any k, s, p and with ρ a random p restriction:*

$$\mathsf{Pr}_\rho[f|\rho \text{ is not a } s - DNF] \leq (5pk)^s$$

Proof:

■

Basically, this lemma says that for most k-CNF under p restriction they have an equivalent s-DNF. The same is true if you switch CNF and DNF as you would just have to take the dual. Observe also that the probability is independent of the size of f. Here is how we will use the lemma:

1. For the bottom two levels we will use
2.
3.
4.
5.

The analysis of our ■