ITCS:CCT09 : Computational Complexity Theory Mar 14 & 16, 2009

Lecture 5 & 6

Lecturer: Jayalal Sarma M.N. *Scribe: Hongyu Liang*

In the previous lecture we introduced the Boolean Circuits, a non-uniform computational model, together with some related circuit classes such as AC^0(polynomial-size, constant depth and unbounded fanin), ACC^0(AC^0 augmented with the MOD gates) and NC^1(polynomial-size, $O(\log n)$ depth and constant fanin). We showed that $\mathsf{AC}^0 \subseteq \mathsf{ACC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{L}$. Today we will introduce a new circuit class, named TC^0, which is obtained by allowing the use of threshold gates Th_k^n in AC^0. We will show that $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$. Furthermore, we will prove that for a small threshold parameter k, namely $k = \log^{O(1)} n$, the threshold function Th_k^n can actually be computed in AC^0. We will prove it by means of hash families.

1 Threshold Functions and TC^0

First we give the definition of threshold functions as follows.

Definition 1 (Threshold Functions) *Given $n, k (k \leq n)$, taking n bits $x_0, x_1, \ldots, x_n$ as inputs, the threshold function Th_k^n is defined as:*

$$Th_k^n(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \sum_{i=1}^n x_i \geq k, \\ 0 & otherwise. \end{cases}$$

From now on we will omit the inputs of Th_k^n if they are clear and unambiguous. We can regard the threshold functions as a kind of gates used in circuits, which we call THRESHOLD gate. For the sake of simplicity, we also use Th_k^n to denote the n-fanin 1-fanout gate which outputs $Th_k^n(x_1, x_2, \ldots, x_n)$ when taking $x_1, x_2, \ldots, x_n$ as inputs.

Now we are able to give the definition of the circuit class TC^0.

Definition 2 TC^0 *is the class of all languages which are decidable by boolean circuits with constant depth, polynomial size, containing only unbounded-fanin* AND *gates,* OR *gates and* THRESHOLD *gates.*

The majority gate is defined as $\textsc{Maj}(x_1, x_2, \ldots, x_n) = Th_{n/2}^n(x_1, x_2, \ldots, x_n)$. It's easy to see that threshold gates can be simulated by majority gates because we can rewrite Th_k^n as:

$$Th_k^n(x_1, x_2, \ldots, x_n) = \begin{cases} \textsc{Maj}(x_1, x_2, \ldots, x_n, 0, \ldots, 0), & n < 2k \\ \textsc{Maj}(x_1, x_2, \ldots, x_n, 1, \ldots, 1), & n \geq 2k \end{cases}$$

where in both cases there are $|n-2k|$ addition input bits. So we can also define TC^0 using MAJ gates instead of THRESHOLD gates.

Obviously $\mathsf{AC}^0 \subseteq \mathsf{TC}^0$. We can further prove that TC^0 is contained in NC^1. A first idea is motivated by the observation that $Th_k^n = 1$ if and only if $\sum_{i=1}^n x_i \geq k$. Thus we can compute the sum of all n input bits and compare the result with k. However, a more detailed analysis reveals that this idea is not good enough. Remember that the AC^0 circuit for adding two n-bit numbers, which we introduced formerly, requires $O(n)$-$fanin$ gates. Since the sum of n 1-bit numbers can have $\Omega(\log n)$ bits, we can only construct a circuit for Th_k^n with polynomial size, $O(\log n)$ depth and $O(\log n)$-fanin gates. Therefore converting this circuit to the one with bounded fanin will make its depth become $O(\log n \log\log n)$, which is not allowed in NC^1.

To get rid of this extra $O(\log\log n)$ factor we need to use another idea which is essnetially used to prove the folllowing theorem which we aimed for.

Lemma 3 *Adding n n-bit numbers can be done in NC^1.*

Proof First we consider only 3 n-bit numbers. We wish to find a circuit in AC^0 computing the sum of them with only bounded-fanin gates, so that we can build a recursive circuit to add n n-bit numbers. But this is unable to be done even if there are only two addends. Now a natural question appears: What can we gain when restricting AC^0 to contain only bounded-fanin gates?

It turns out that although we cannot compute the sum of three n-bit numbers, we can reduce it to the addition of another two n-bit numbers using only bounded-fanin gates. Suppose we want to add $a = \overline{a_{n-1}a_{n-2}\ldots a_0}$, $b = \overline{b_{n-1}b_{n-2}\ldots b_0}$ and $c = \overline{c_{n-1}c_{n-2}\ldots c_0}$. Consider another two $n-$bit numbers d and e defined by:

$$d_i = \begin{cases} a_i \oplus b_i \oplus c_i, & 0 \leq i \leq n-1 \\ 0. & i = n \end{cases}$$

and

$$e_i = \begin{cases} 0, & i = 0 \\ (a_{i-1} \wedge b_{i-1}) \vee (a_{i-1} \wedge c_{i-1}) \vee (b_{i-1} \wedge c_{i-1}). & 1 \leq i \leq n \end{cases}$$

It is easy to verify that $a+b+c = d+e$, and the construction of d and e can be done using polynomial-size, constant-depth circuit with only bounded-fanin gates. In order to calculate the sum of n n-bit numbers, we first divide them into groups each of which contains no more than 3 elements. Then, inside each group, we use the above method to construct the two new addends. We do this recursively until there are only two numbers left, and then use the original AC^0 circuit to compute the sum of them. Now, the recursive circuit we build has polynomial size, $O(\log n)$ depth and contains only bounded-fanin gates except for the last level. For the last level we need only trivially change it into a NC^1 circuit. ■

Theorem 4 $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$.

Proof It follows from Lemma 3 that adding n 1-bit numbers can be done in NC^1. In order to compute Th_k^n, we just need to add all the n input bits and compare the result with k. ■

So far we have proved that $\mathsf{AC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1$. It can be further shown that $\mathsf{ACC}^0 \subseteq \mathsf{TC}^0$. In fact, this will follow from what we show below.

Although we have put TC^0 into its right position, it is useful for us to better understand it. Except for the threshold functions we defined, what other functions can be computed in TC^0? Remember that the threshold function depends only on the number of 1s amongst the inputs. Indeed, this property precisely characterize a large class of functions.

Definition 5 (Symmetric Functions) *A function f of n variables is called* symmetric *if $\forall \sigma \in S_n$, $f(x_1, x_2, \ldots, x_n) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$, where S_n is the collection of all permutations of $[n]$.*

Unless otherwise stated, throughout this note all the inputs and outputs of symmetric functions are from $\{0, 1\}$. Notice that threshold functions are all symmetric. Indeed we have the following theorem:

Theorem 6 TC^0 *contains all symmetric functions.*

Proof Suppose f is an arbitrary symmetric Boolean function of n variables. We know that f only depends on the number of 1s among its input. Hence the number of different possibilities for the input setting of the function is exactly n.

Given x as the input if we are able to detect the number of 1s in the input, the function can be hardwired into the circuit.

■

2 Threshold Functions in AC^0

One of the main interest in circuit complexity is the question of whether one circuit class is strictly contained in another. So we may ask whether $\mathsf{AC}^0 \subsetneq \mathsf{TC}^0$ or $\mathsf{AC}^0 = \mathsf{TC}^0$. The latter will hold if and only if we can simulate threshold gates in AC^0. It is easy to see that $Th_1^n = \vee$ and $Th_n^n = \wedge$, so Th_k^n is in AC^0 if $k = 1$ or n. We may ask that is there any other k such that $Th_k^n \in \mathsf{AC}^0$? It turns out that many k enables this inclusion.

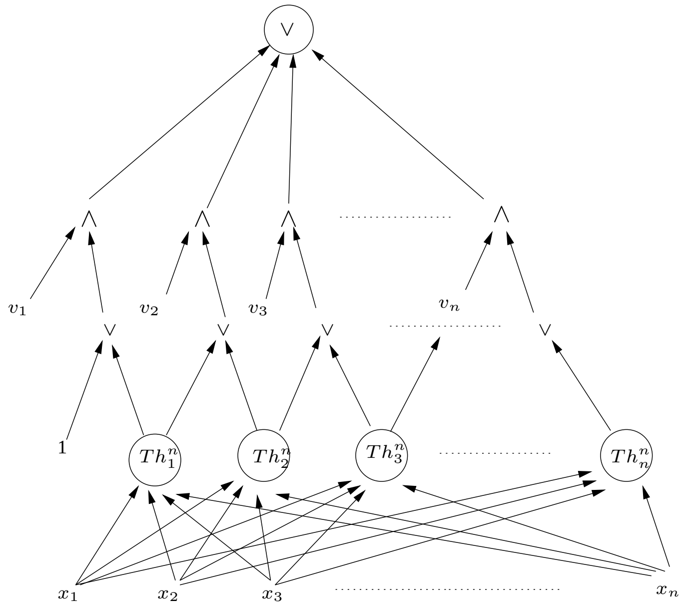

Figure 1: Threshold circuit computing any symmetric function. v_i is the value of the function when the number of 1s in the input is i.

Theorem 7 *$Th_k^n \in \mathsf{AC}^0$ for $k = \log^{O(1)} n$.*

In this class we will only prove a much simpler result as follows. [1]

Corollary 8 *$Th_{\log n}^n \in \mathsf{AC}^0$.*

In order to prove this we need some nontrivial techniques. Let $\{x_i \mid 1 \leq i \leq n\}$ be the input of $Th_{\log n}^n$. Let $k = \log n$ and $S = \{i \mid x_i = 1\}$. What we need is to distinguish between the two cases $|S| < \log n$ and $|S| \geq \log n$. The idea is to construct a hash family such that it has some "good" properties when $|S| < \log n$, and this property can be tested in AC^0.

We need some basic concepts first. A hash family H is a collection of hash functions with the same domain and range. We say H is good for a set S if $\exists f \in H$ such that f_S is a bijection, where f_S is the function f restricting on the input set S. We are interested in the collection of hash functions from $X = [n]$ to a set $T = [t]$, where the value of t will be determined later. More precisely, we expect H to have the following properties:

1. If $|S| < k$, then H is good for S.

[1] The exposition that we present here was communicated to us by Prof. Meena Mahajan.

2. If $|S| > t$, then H is not good for S.
3. If $k \leq |S| \leq t$, then there is no constraint on H.

Notice that the second property is trivial, since all functions in H has the same range $[t]$. The third property is actually redundant, but we still put it here for purpose of clearness. Now we only need to consider the first property.

Lemma 9 *Let $t = \log^2 n$, p be a prime $n \leq p \leq 2n$. Define a hash family H_p as follows:*

$$H_p = \left\{ h_\alpha \;\middle|\; \begin{array}{c} \alpha \in \{1, 2, \ldots, p-1\}, \\ h_\alpha = (\alpha x \mod p) \mod t \end{array} \right\}$$

Then H_p is good for S if $|S| < k$.

Proof Let $W = \{(\alpha, u, v) \mid \alpha \in \{1, 2, \ldots, p-1\},\ u, v \in S,\ h_\alpha(u) = h_\alpha(v)\}$, where h_α is defined in Lemma 9. Suppose there exists $S \subseteq [n]$, $|S| < k$ such that H_p is not good for S, then we have $|W| \geq p - 1$.

Now we fix $u, v \in S, u \neq v$ and let $A_{u,v} = \{\alpha \mid h_\alpha(u) = h_\alpha(v)\}$. Note that $\alpha \in A_{u,v}$ means that $(\alpha u \mod p) \mod t = (\alpha v \mod p) \mod t$, and thus $(\alpha u \mod p - \alpha v \mod p) = kt$, where $k \in \{0, \pm 1, \ldots, \pm \frac{p-1}{t}\}$. Since $p \geq n = |S|$ and $\alpha > 0$, we know that for a fixed $k \neq 0$ there is at most one α satisfying the above equality, and when $k = 0$ such α doesn't exist. So $|A_{u,v}| \leq \frac{2(p-1)}{t}$, and we have $|W| = \sum_{0 \leq u \leq v < n} |A_{u,v}| \leq \frac{2(p-1)}{t}\binom{\log n}{2} < p - 1$ because of our choice $t = \log^2 n$, which contradicts our previous result of $|W| \geq p - 1$. ■

Next we need to show that testing whether H_p is good for a given set S can be done in AC^0. Moreover, if this is the case, we need to find $h \in H_p$ such that h_S is a bijection. Why does it make sense? Let's consider the following process. First, we use an AC^0 circuit to test whether H_p is good for S. If the answer is no, we assert confidently that $|S| > k$ thanks to the three properties of H_p. On the other hand, a "yes" answer implies that $|S| \leq t$ which is not enough for us. But under this case we only need a little more work to show that computing $Th^t_{\log n}$ instead of $Th^n_{\log n}$ is sufficent, which we may expect to have a lower complexity. Now we formalize the ideas.

Lemma 10 *Given a hash family H_p defined in Lemma 9 and a set S, we can decide whether H_p is good for S in AC^0. If so, we can find $h \in H_p$ such that h_S is a bijection.*

Lemma 11 *Given $h \in H_p, S \subseteq [n]$ such that h_S is a bijection, we have that $|S| \geq \log n$ if and only if at least $\log n$ bits of $\{y_j \mid 0 \leq j < t\}$ are 1s, where $y_j = \bigvee_{i \in [n]}(h(i) = j)$.*

Lemma 12 $Th^{\log^2 n}_{\log n} \in AC^0$.

Proof [**Lemma 10**] Let $x = \{x_1, x_2, \dots, x_n\}$. We first define some small circuits as follows.

$$\forall i \in [n], \forall j \in [t],\ B_{\alpha,i,j} = \begin{cases} 1 & \text{if } h_\alpha(i) = j \\ 0 & \text{otherwise.} \end{cases}$$

$$\forall j \in [t], C_{\alpha,j}(x) = \begin{cases} 1 & \text{if there exist } i_1, i_2 \in S \text{ s.t. } h_\alpha(i_1) = h_\alpha(i_2) = j, \\ 0 & \text{otherwise.} \end{cases}$$

$$\forall \alpha \in [p], D_\alpha(x) = \begin{cases} 1 & \text{if } h_\alpha|_S \text{ is 1-1,} \\ 0 & \text{otherwise.} \end{cases}$$

$$E(x) = \begin{cases} 1 & \text{if } H_p \text{ is good for } S, \\ 0 & \text{otherwise.} \end{cases}$$

It suffices to show that there exists an AC^0 circuit computing E. Note that we have the following:

$$E(x) = \bigvee_{\alpha \in [p]} (D_\alpha(x))$$

$$D_\alpha(x) = \bigwedge_{j \in [t]} (\neg(C_{\alpha,j}(x)))$$

$$C_{\alpha,j}(x) = Th^n_2(x_1 \wedge B_{\alpha,1,j}, x_2 \wedge B_{\alpha,2,j}, \dots, x_n \wedge B_{\alpha,n,j})$$

It is easy to verify that this recursive construction of E is in AC^0. A modified version of this circuit will output α such that $D_\alpha(x) = 1$ if $E(x) = 1$, while we omit the details here. ■

Proof [**Lemma 11**] It immediately follows from the fact that h_S is a bijection. ■

To proof Lemma 12 we need another result stated as follows.

Lemma 13 *Adding* $\log n$ n*-bit numbers can be done in* AC^0.

Proof Let $l = \log n$. Suppose we have inputs $a_1, a_2, \dots, a_l$, where $a_i = \sum_{j=0}^{n-1} a_{i,j} 2^j$, and want to compute the sum of them. $\forall j \in \{0, 1, \dots, n-1\}$, we define $S_j = \sum_{i=1}^{l} a_{i,j}$. That is, we use S_j to denote the j^{th} bit sum. Note that $S_j \leq \log n$ so the size of S_j is at most

$\log\log n$. Let $t = \log\log n$ and $S_k = \sum_{j=0}^{t-1} S_{k,j} 2^j$. Now we rewrite the sum:

$$
\begin{aligned}
\sum_{i=1}^{l} a_i &= \sum_{k=0}^{n-1} S_k 2^k \\
&= \sum_{k=0}^{n-1} (\sum_{j=0}^{t-1} S_{k,j} 2^j) 2^k \\
&= \sum_{j=0}^{t-1} (\sum_{k=0}^{n-1} S_{k,j} 2^{k+j})
\end{aligned}
$$

Notice that $\sum_{k=0}^{n-1} S_{k,j} 2^{k+j})$ has at most $n + \log\log n$ bits, so we have reduced the sum of $\log n$ n-bit numbers to that of $\log\log n$ numbers of $n + \log\log n$ bits each. If we perform this for i times, we can reduce the original problem to the summation of $\log^{(i+1)} n$ numbers of $n + \sum_{j=2}^{i+1} \log^{(j)} n$ bits each, where $\log^{(k)}$ denotes the log function iterated by k times. We stop until at most two addends are left, that is, after $h = \min\{k \mid \log^{(k+1)} n \leq 2\}$ rounds of reduction. To bound the size of the left number we need the following lemma, although we will not prove it here.

Lemma 14 *Let $h = min\{k \mid \log^{(k+1)} n \leq 2\}$. We have*

$$\sum_{i=2}^{k} \log^{(i)} n = O(\log n)$$

and

$$\prod_{i=2}^{k} \log^{(i)} n = O(\log n)$$

Using this lemma, we know that the reduction stops at adding 2 numbers of $n + O(\log n)$ bits, which can be done in AC^0. We are left to show that the reduction process can also be done using AC^0 circuits. The number of rounds of reductions is $\log^*(n) = \min\{k \mid \log^{(k+1)} n \leq 2\}$. It can be shown that the two addends in the last round only depend on $\log^*(n)$ bits. Combined with Lemma 15 showed later, we know that the reduction can also be done in AC^0. ■

Lemma 15 *Suppose f is a function from $\{0,1\}^m$ to $\{0,1\}$. There exists a const depth, $O(2^m)$ size circuit which computes f.*

Proof Just consider the equivalent DNF of the function. ■

Now we are able to prove Lemma 12.

Proof [**Lemma 12**] We divide the $\log^2 n$ numbers into $\log n$ groups each of which contains $\log n$ numbers. For every group we build an AC^0 circuit computing the sum of the numbers in it, and then we use another AC^0 circuit to add the $\log n$ group sums. ■

Using Lemma 10,11 and 12, it is not hard to prove Corollary 8 and we omit the rigorous proof. The basic idea is that we first use an AC^0 circuit to test if H_p is good for S. If the answer is no, we know that $|S| \geq t$ so $Th_k^n = 1$. If the answer is yes, we can find a function $h \in H_p$ such that h_S is 1-1. Then we use Lemma 11 to reduce the computing of $Th_{\log n}^n$ to $Th_{\log n}^{\log^2 n}$, which can be done in AC^0 thanks to Lemma 12.

Theorem 7 shows that for any k, $Th_{\log^k n}^n \in AC^0$. But what about larger k, say, $k = \Omega(n)$? It still remains open.

3 NC Hierarchy

We can define some hierarchy of circuit classes by relaxing the constraints of some known circuit class like AC^0 and NC^1. We explicitly give the following definitions.

Definition 16 *AC^k is the class of all languages which are decidable by boolean circuits with $O(\log^k n)$ depth, polynomial size, containing only unbounded-fanin AND gates, OR gates and NOT gates.*

Definition 17 *NC^k is the class of all languages which are decidable by boolean circuits with $O(\log^k n)$ depth, polynomial size, containing only bounded-fanin AND gates, OR gates and NOT gates.*

By definition it is easy to prove the following theorem.

Theorem 18 $\forall k \geq 0, \mathsf{NC}^k \subseteq \mathsf{AC}^k \subseteq \mathsf{NC}^{k+1}$.

However, except that $\mathsf{AC}^0 \subsetneq \mathsf{NC}^1$, none of these inclusions are known to be strict.