Notes on Complexity Theory Last updated: October, 2015

Lecture 10

Jonathan Katz

1 Non-Uniform Complexity

1.1 The Power of Circuits

Last time we saw that every function $f : \{0,1\}^n \to \{0,1\}$ could be computed by a circuit family of size $n \cdot 2^n$, and noted that this could be improved to $(1+\varepsilon) \cdot 2^n/n$ for every $\varepsilon > 0$. We now show that this is essentially tight.

Theorem 1 *Let $\varepsilon > 0$ and $q(n) = (1-\varepsilon)\frac{2^n}{n}$. Then for n large enough there exists a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size at most $q(n)$.*

Proof In fact, the fraction of functions $f : \{0,1\}^n \to \{0,1\}$ that can be computed by circuits of size at most $q(n)$ approaches 0 as n approaches infinity; this easily follows from the proof below.

Let $q = q(n)$. The proof is by a counting argument. We count the number of circuits of size q (note that if a function can be computed by a circuit of size at most q, then by adding useless gates it can be computed by a circuit of size exactly q) and show that this is less than the number of n-ary functions. A circuit having q internal gates is defined by (1) specifying, for each internal gate, its type and its two predecessor gates, and (2) specifying the output gate. We may assume without loss of generality that each gate of the circuit computes a different function — otherwise, we can simply remove all but one copy of the gate (and rewire the circuit appropriately). Under this assumption, permuting the numbering of the internal gates does not affect the function computed by the circuit. Thus, the number of circuits with q internal gates is at most:

$$\frac{\left(3 \cdot (q+n)^2\right)^q \cdot q}{q!} \leq \frac{\left(12 \cdot q^2\right)^q \cdot q}{q!},$$

using the fact that $q \geq n$. (In fact, we are over-counting since some of these are not valid circuits, e.g., they are cyclic.) We have:

$$\begin{aligned} \frac{\left(12 \cdot q^2\right)^q \cdot q}{q!} &\leq q \cdot (36)^q \cdot \frac{q^{2q}}{q^q} \\ &= q \cdot (36 \cdot q)^q \\ &\leq (36 \cdot q)^{q+1} \\ &\leq (2^n)^{(1-\varepsilon)2^n/n\,+\,1} \;=\; 2^{(1-\varepsilon)2^n+n}, \end{aligned}$$

for n sufficiently large, using Stirling's bound $q! \geq q^q/e^q \geq q^q/3^q$ for the first inequality. But this is less than 2^{2^n} (the number of n-ary boolean functions) for n large enough. ■

We saw that any function can be computed by a circuit family of depth $n + \lceil \log n \rceil \leq (1+\varepsilon) \cdot n$ for any $\varepsilon > 0$ (for n large enough). This, too, is essentially tight (see [1, Sect. 2.12]):

Theorem 2 *Let $\varepsilon > 0$ and $d(n) = (1-\varepsilon) \cdot n$. Then for n large enough there exists a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of depth at most $d(n)$.*

1.2 Polynomial-Size Circuits

As with our interest in polynomial-time algorithms, we are also interested in polynomial-size circuits. Define $\mathcal{P}_{/\mathsf{poly}} \stackrel{\text{def}}{=} \bigcup_c \text{SIZE}(n^c)$. A second characterization of $\mathcal{P}_{/\mathsf{poly}}$ is:

Definition 1 *$L \in \mathcal{P}_{/\mathsf{poly}}$ iff there exists a Turing machine M running in time polynomial in its first input, and a sequence of "advice strings" $\{z_n\}_{n\in\mathbb{N}}$ such that $x \in L$ iff $M(x, z_n) = 1$.*

A proof that this is an equivalent definition uses the proof that follows, and is left as an exercise.

Perhaps not surprisingly, we have

Theorem 3 *$\mathcal{P} \subseteq \mathcal{P}_{/\mathsf{poly}}$.*

Proof We prove a stronger result: for any $t(n) \geq n$ we have $\text{TIME}(t(n)) \subseteq \text{SIZE}(t(n) \cdot \log t(n))$. Given a language $L \in \text{TIME}(t(n))$, we first construct an *oblivious* Turing machine M deciding L in time $t'(n) = O(t(n) \cdot \log t(n))$. For simplicity, assume M has a single work tape (in addition to its input tape); the proof is similar if M has k work tapes. Considering the tableau of the computation of $M(x)$ with $x \in \{0,1\}^n$, we see that $M(x)$ passes through (at most) $t'(n)+1$ configurations and each configuration can be represented using $O(t'(n))$ bits. We will imagine associating a set of wires with each configuration of $M(x)$. Note, however, that in passing from one configuration to the next most of the wire values will remain unchanged and, because of the obliviousness of M, we know in advance which wires can possibly change. Wires that do not change can simply be "extended" from one configuration to the next, without using any gates.

In a bit more detail, we show how to compute one configuration from the next using only a constant number of gates. To go from configuration i to configuration $i+1$, we construct a boolean sub-circuit that takes as input (1) the state of M at time i, (2) the wires representing the cell scanned by M at time i (here we use the assumption that M is oblivious, so we know in advance where the work-tape head of M will be at every time step); and (3) the position of the input scanned by M at time i (once again using obliviousness). The output of this sub-circuit is (1) the updated state of M and (2) an updated value for what is written on the cell scanned by M at the previous time step. Note that the number of inputs to this sub-circuit is constant, so by what we have shown previously we can compute the transition function using a constant number of gates. Using similar reasoning, we can also add a final sub-circuit that tests whether the final state of M is accepting or not. We thus have $O(t'(n))$ constant-size sub-circuits, for a circuit of total size $O(t'(n))$. ■

The above proof can also be used to show that the following problem is $\mathcal{NP}$-complete:

$$\mathsf{CKT\text{-}SAT} \stackrel{\text{def}}{=} \{C \mid C \text{ is a circuit, and } \exists x \text{ s.t. } C(x) = 1\}.$$

Could it be that $\mathcal{P} = \mathcal{P}_{/\mathsf{poly}}$? Actually, here we know that the inclusion is strict, since $\mathcal{P}_{/\mathsf{poly}}$ is still powerful enough to contain undecidable languages. To see this, let $L \subseteq 1^*$ be an undecidable unary language, and note that any unary language is trivially in $\mathcal{P}_{/\mathsf{poly}}$.

Could it be that $\mathcal{NP} \subseteq \mathcal{P}_{/\mathsf{poly}}$? This would be less surprising than $\mathcal{P} = \mathcal{NP}$, and would not necessarily have any practical significance (frustratingly, $\mathcal{NP} \subseteq \mathcal{P}_{/\mathsf{poly}}$ but $\mathcal{P} \neq \mathcal{NP}$ would mean that efficient algorithms for $\mathcal{NP}$ *exist*, but can't be found efficiently). Nevertheless, the following result suggests that $\mathcal{NP} \not\subseteq \mathcal{P}_{/\mathsf{poly}}$:

Theorem 4 (Karp-Lipton) *If $\mathcal{NP} \subseteq \mathcal{P}_{/\mathsf{poly}}$ then $\Sigma_2 = \Pi_2$ (and hence $\mathsf{PH} = \Sigma_2$).*

Proof We begin with a claim that can be proved easily given our earlier work on self-reducibility of SAT: if $\mathsf{SAT} \in \mathcal{P}_{/\mathsf{poly}}$ then there exists a polynomial-size circuit family $\{C_n\}$ such that $C_{|\phi|}(\phi)$ outputs a satisfying assignment for ϕ if ϕ is satisfiable. That is, if SAT can be *decided* by polynomial-size circuits, then SAT can be *solved* by polynomial-size circuits.

We use this claim to prove that $\Pi_2 \subseteq \Sigma_2$ (from which the theorem follows). Let $L \in \Pi_2$. This means there is a Turing machine M running in time polynomial in its first input such that[1]

$$x \in L \Leftrightarrow \forall y \exists z : M(x, y, z) = 1.$$

Define $L' = \{(x, y) \mid \exists z : M(x, y, z) = 1\}$. Note that $L' \in \mathcal{NP}$, and so there is a Karp reduction f from L' to SAT. (The function f can be computed in time polynomial in $|(x, y)|$, but since $|y| = \mathsf{poly}(|x|)$ this means it can be computed in time polynomial in $|x|$.) We may thus express membership in L as follows:

$$x \in L \Leftrightarrow \forall y : f(x, y) \in \mathsf{SAT}. \tag{1}$$

But we then have

$$x \in L \Leftrightarrow \exists C \forall y : C(f(x, y)) \text{ is a satisfying assignment of } f(x, y),$$

where C is interpreted as a circuit, and is chosen from strings of (large enough) polynomial length. Thus, $L \in \Sigma_2$. ■

1.3 Small Depth Circuits and Parallel Computation

Circuit depth corresponds to the time required for the circuit to be evaluated; this is also evidenced by the proof of Theorem 3. Moreover, a circuit of size s and depth d for some problem can readily be turned into a parallel algorithm for the problem using s processors and running in "wall clock" time d. Thus, it is interesting to understand when low-depth circuits for problems exist. From a different point of view, we might expect that *lower bounds* would be easier to prove for low-depth circuits. These considerations motivate the following definitions.

Definition 2 *Let $i \geq 0$. Then*

- $L \in \mathsf{NC}^i$ *if L is decided by a circuit family $\{C_n\}$ of polynomial size and $O(\log^i n)$ depth over the basis $\mathcal{B}_0$.*
- $L \in \mathsf{AC}^i$ *if L is decided by a circuit family $\{C_n\}$ of polynomial size and $O(\log^i n)$ depth over the basis $\mathcal{B}_1$.*

$\mathsf{NC} = \bigcup_i \mathsf{NC}^i$ *and* $\mathsf{AC} = \bigcup_i \mathsf{AC}^i$.

Note $\mathsf{NC}^i \subseteq \mathsf{AC}^i \subseteq \mathsf{NC}^{i+1}$. Also, NC^0 is not a very interesting class since the function computed by a constant-depth circuit over $\mathcal{B}_0$ can only depend on a constant number of bits of the input.

Bibliographic Notes

Theorem 1 is due to Shannon; the proof here is adapted from Vollmer [2].

[1] By convention, quantification is done over strings of length some (appropriate) fixed polynomial in $|x|$.

References

[1] J.E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley, 1998.

[2] H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.