CS 810: Complexity Theory	2/7/2007

Lecture 8: Constant-Depth Circuits

Instructor: Dieter van Melkebeek — Scribe: Seeun William Umboh

DRAFT

In the last lecture we applied alternations to prove some time-space lower bound results for SAT. We also introduced the notion of nonuniform computation, and nonuniform models such as Boolean circuits, branching programs and uniform machines with advice.

Today, we begin with a theorem that suggests that SAT does not have small circuits. Then, we investigate constant-depth circuits.

1 If NP has small circuits, then PH collapses

Theorem 1. *If* $\mathrm{NP} \subseteq \mathrm{P/poly}$*, then* $\mathrm{PH} = \Sigma_2^p$*.*

Note that we only need to show that we can simulate a Π_2^p computation with a Σ_2^p computation: since $\Pi_2^p = \mathrm{co}\Sigma_2^p$, if $\Pi_2^p \subseteq \Sigma_2^p$ then $\Pi_2^p = \Sigma_2^p$.

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

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

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 $f(x,y)$[1]?

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

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

where V is the following recursive predicate:

$$\begin{array}{ll} (1) & \textbf{if } \varphi \text{ has at least 1 variable } \textbf{then } C_{SAT}(\varphi) \iff C_{SAT}(\varphi|_{x_1 \leftarrow 0}) \vee C_{SAT}(\varphi|_{x_1 \leftarrow 1}) \\ (2) & \qquad\qquad\qquad\qquad\qquad \textbf{else } C_{SAT}(\varphi) \iff \varphi \textit{ is true} \end{array}$$

where x_1 is the first unset variable of φ.

[1] One slight detail here: the circuit C_{SAT} takes inputs of fixed input size only, but $f(x,y)$ is of varying size. However, we can simply pad $f(x,y)$ to an equivalent instance of the required size.

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. $\square$

Since we do not believe that the polynomial-time hierarchy collapses, this is taken to be evidence suggesting that NP does not have polynomial-size circuits.

The proofs of the following are similar, so we leave them as exercises.

Exercise 1. *If PSPACE* $\subseteq \mathrm{P}/\mathrm{poly}$ *then PSPACE* $= \Sigma_2^p$.

Exercise 2. *If EXP* $\subseteq \mathrm{P}/\mathrm{poly}$ *then EXP* $= \Sigma_2^p$.

2 Nonuniform Lower Bounds for NP

In the previous lecture, we stated that one application of nonuniform models of computation is in attempts to prove lower bounds for computing certain functions (in particular NP-complete problems). In fact, there has been little progress in proving lower bounds for NP-complete problems. We survey results in this section. Nontrivial lower bounds have been proven for restricted models of computation - these are discussed in the next section.

2.1 Boolean Circuits

We only know the following facts in this area:

Theorem 2. $C(f) = O(\frac{2^n}{n})$ *for any Boolean function* $f : \{0,1\}^n \to \{0,1\}$

Using the naive encoding of the truth table into a DNF, we can get a $O(n2^n)$-size circuit. A better analysis gives the better bound.

Theorem 3. $C(f) = \Omega(\frac{2^n}{n})$ *for most Boolean functions* f*. That is, if we pick* f *uniformly at random from the set of Boolean functions on* n *variables, the probability that* $C(f) = \Omega(\frac{2^n}{n})$ *converges to 1 as* n *grows.*

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 also takes care of the possibility that the input is negated. 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. $\square$

This shows that most Boolean functions require circuits of the maximum circuit size, up to a constant factor. So we would expect that at least for complicated functions like those that capture NP-complete problems, we can prove non-trivial lower bounds. However, that is not the case. The best known lower bound for any $L \in \mathrm{NP}$ is:

Proposition 1. $\exists L \in \mathrm{NP}$ *with* $C_L(n) = \Omega(n)$

2.2 Branching Programs

For branching programs, we can show that $BP(f) = \Theta(\frac{2^n}{n})$ for most functions just as in the case of Boolean circuits, and the best known lower bound is roughly quadratic. Even though the lower bound is better than in the case for Boolean circuits, it is still a rather trivial result. Recall from the previous lecture that the corresponding Turing machine uses $O(\log v)$ space where v is the size of the branching program.

We can hope to prove a lower bound on branching program size by restricting our attention to layered branching programs of constant width. However, these are more powerful than they might seem at first sight, and we cannot prove substantially better lower bounds than for arbitrary branching programs.

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. We first give some basic facts about constant-depth circuits, and then prove that they require exponential size to even compute the parity function.

Definition 1 (constant-depth circuit)**.** *A constant-depth circuit is a Boolean circuit with unbounded fan-in but whose depth is bounded by a constant.*

This model might seem too restricted, but we have already mentioned that any function can be computed by a depth 2 CNF or DNF. However, such a circuit is in general of exponential size, and we would like to know if we can do better. We will see that even for the PARITY function, the size required is exponential.

We have encountered constant-depth circuits before in the lecture on alternation, and we showed that we can simulate alternation with constant-depth circuits of exponential fan-in. Today, we will look at such circuits that are of polynomial size. In particular, we look at the following family of classes:

Definition 2 (AC^k)**.** $AC^k = \{L | C_{O(\log^k n)}(L_n)$ *is polynomial*$\}$, *where* $C_{O(\log^k n)}(L_n)$ *denotes the complexity of circuits with unbounded fan-in, depth* $O(\log^k n)$*, and deciding the restriction of* L *to length* n*.*

For now, we are interested in AC^0, the class of languages decidable by constant-depth circuits of polynomial size. Let us now look at examples of languages in and not in AC^0.

Proposition 2. *The decision variant of binary addition is in* AC^0*.*

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. $\square$

We will discuss problems that can be computed in various AC^k in a future lecture. 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 $\bigoplus_n$.

Theorem 4. *PARITY is not in* AC^0*.*

By proving this result, we will have also shown that PARITY cannot be computed by polynomial size circuits with bounded fan-in and log-depth. This follows from Theorem 4 by using a divide-and-conquer strategy.

In fact, the proof we outline today shows that $C_d(\bigoplus_n) = 2^{\Omega(n^{\frac{1}{d-1}})}$. As PARITY can be computed by circuits of size $2^{O(n^{\frac{1}{d-1}})}$, this gives an exact characterization of the size of constant-depth circuits required to compute parity.

The proof we present today 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 $\rho(x_i)$.*

Note that if we apply a random restriction to a parity function, we get a parity function or its complement on those bits set to $*$.

Proof sketch of Theorem 4. Let us start with some AC^0 circuit C. WLOG, we assume that for each level of C, there are only ANDs or ORs, and that the circuit alternates between these. We also assume that the inputs to the circuit are the variables and their negations, allowing us to ignore NOT gates for the most part.

The main ingredients of the proof are:

Proposition 3. $C_2(\bigoplus_n) = \Theta(n2^{n-1})$.

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. $\square$

Lemma 1 (Switching Lemma)**.** *Given a CNF with small bottom fan-in. Then we can apply a random restriction that does not set to many variables so that with high probability the resulting function can be written as a DNF with small bottom fan-in.*

Note that the statement is trivial if the restriction can set all variables – the "not setting too many variables" is important. Also, all of the qualifications like "not too many" and "small" need to be quantified appropriately for the statement to hold but we keep the exposition of this approach at a qualitative level.

Proof Idea. Consider an AND of ORs, where each OR has size at most k. Notice that a random restriction is not very likely to set an OR to 0 since all literals involved need to be set to 0 for that to happen. But if k is small, there is a nontrivial probability that this happens. There are two cases:

1. There are a large number of pairwise disjoint ORs. In that case, there are many independent events that can set the AND gate to 0, namely each of those pairwise disjoint ORs being set to 0. Since each of those events happens with a nontrivial probability, the odds are that the random restriction will set the AND gate to 0, in which case it can trivially be written as a DNF will small bottom fan-in.

2. There is not a large number of pairwise disjoint ORs. Let V be a minimal set of variables such that each OR queries at least one variable from V. Since there is a lot of overlap among the ORs, V is small. If we query all the variables in V, then we have essentially reduced our problem to a simpler one of the same type, namely the transformation of a CNF with bottom fan-in at most $k-1$. This is because each of the ORs contains at least one literal in V. We then repeat the case distinction to that simpler problem, depending on the setting of the variables in V.

Along every branch of this process, we will eventually end up in case 1. Since there are at most k steps and each step involves querying a small number of variables, we end up with a decision tree of small depth that represents the given CNF under a random restriction with high probability. A decision tree of small depth can be turned into a DNF with small bottom fan-in by writing down an OR over all paths in the decision tree that lead to acceptance of the AND of all the conditions that define the path. $\square$

Note that we can also switch from a DNF to a CNF by considering the negation of the circuit.

We use the Switching Lemma to reduce the depth of the circuit by 1 at a time until we are left with a circuit of depth 2. Suppose that the bottom gates are ANDs. To apply the switching lemma, we need to ensure the gates at the bottom of the circuit have small fan-in. To ensure this, we insert dummy OR gates below the AND gates. Namely, for each input x to the AND gate, we replace that with x OR x. Now, we apply the switching lemma to the AND of ORs we have created. With high probability, each application is successful in creating an OR of ANDs with small bottom fan-in and without setting too many variables. Now the second bottom-most and third bottom-most levels are both ORs and can be merged. This reduces the depth of the circuit by 1 (back down to d since we added a level initially).

Now the circuit still has small bottom fan-in, so we can apply the switching lemma again. We repeat this process until we get a circuit C' of depth 2. At this point, if C computed $\bigoplus_n$, then C'

computes $\bigoplus_m$ on some m-subset of the variables (those that were unset by the random restrictions). At this point we use Proposition 3 to derive a lower bound on the size of the remaining circuit (and thus also of the original circuit). If m is still relatively large, this gives an exponential lower bound on the size of the original circuit. Further, there must be a positive probability that each application of the switching lemma was indeed successful. $\square$

Next lecture, we give an alternate proof that PARITY requires exponential size constant-depth circuits. This proof will use low-degree polynomial approximations rather than random restrictions. The bound that we will get, $C_d(\bigoplus_n) = 2^{\Omega(n^{\frac{1}{2d}})}$ is not as tight as the previous result, but the advantage is that it applies to circuits with gates other than AND, OR, NOT.