CSE 531: Computational Complexity I **Winter 2016**

Lecture 5: Circuit Lower bounds and Hierarchy Theorems

January 20, 2016

Lecturer: Paul Beame *Scribe: Paul Beame*

We have many uniform complexity classes $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$, $\mathsf{DTIME}(\mathsf{T}(\mathsf{n}))$ and $\mathsf{NTIME}(\mathsf{T}(\mathsf{n}))$ for all bounds $T(n)$. We have non-uniform complexity classes $\mathsf{SIZE}(\mathsf{S}(\mathsf{n}))$ for all bounds $S(n)$. However, it is consistent with what we have seen so far that all the uniform complexity classes are the same, as are all the non-uniform complexity classes.

We now show that additional resources allow us to compute new things. We begin with circuit complexity since it is a bit simpler to analyze for this question.

1 Circuit Lower Bounds

We previously saw that there are DNF circuit families of size $O(n2^n)$ for every Boolean function. On your homework you are showing that for general circuits $O(2^n/n)$ suffices. We first show that this is asymptotically optimal.

Theorem 1.1. *Almost all (a $1 - o(1)$ fraction of) functions $f_n : \{0,1\}^n \to \{0,1\}$ require circuit size $2^n/n - o(2^n/n)$.*

Proof. The basic idea is to count the number of functions of circuit size at most S and compare it to the total number of functions on n bits.

We begin by counting the number of circuits of size at most S:

We can assume without loss of generality that $S \geq n + 2$. A circuit of size at most S has n inputs $x_1, \ldots, x_n$ as well as the constants 0 and 1. Each of its S gates has its inputs from among the other gates, as well as these $n + 2$ other values.

A gate can be described by its two inputs and its gate type from among $\{\wedge, \vee, \neg\}$. There are at most $3(S + n + 2)^2$ possibilities per gate. The circuit can then be described by its S gates and the name of the output, for which there are $S + n + 2$ possibilities. (Note that if the circuit size is less than S, we simply add dummy gates that are not connected to the output.)

Therefore there are at most $3^S(S + n + 2)^{2S+1}$ different circuits in total.

However, this over-counts the number of different functions that these circuits can compute. The function stays the same even if we permute the gate numbers, which changes the circuit description.

Therefore the number of different functions computed by circuits of size at most S is at most

$$\begin{aligned} 3^S(S+n+2)^{2S+1}/S! &\leq 3^S(2S)^{2S+1}/S! && \text{since } S \geq n+2 \\ &\leq 3^S(2S)^{2S+1}e^S/S^S && \text{using the fact that } S! \geq (S/e)^S \text{ for all } S \geq 1 \\ &\leq c^S S^{S+1} \end{aligned}$$

for some constant c that is roughly $6e$.

Now the number of functions $f : \{0,1\}^n \to \{0,1\}$ is the same as the number of truth tables, binary vectors of length 2^n; i.e., 2^{2^n}.

$2^{2^n}/n$ is only a $o(1)$ fraction of 2^{2^n}. So, if $c^S S^{S+1} < 2^{2^n}/n$ then circuits of size at most S can only compute a $o(1)$ fraction of all functions. Therefore, almost all functions require circuit size S with

$$c^S S^{S+1} \geq 2^{2^n}/n.$$

Taking logarithms of both sides we have

$$S \log_2 c + (S+1) \log_2 S \geq 2^n - \log_2 n.$$

Now since we can assume that $S \leq 2^n$ and hence $\log_2 S \leq n$, we obtain

$$S \log_2 c + (S+1)n \geq 2^n - \log_2 n.$$

Hence

$$S \geq \frac{2^n - n - \log_2 n}{n + \log_2 c}.$$

This implies that $S \geq 2^n/n - o(2^n/n)$ which is what we wanted to show. □

From this we derive a hierarchy for circuit complexity based on your homework solution.

Corollary 1.2. *Let c be the constant such all functions $f : \{0,1\}^n \to \{0,1\}$ have complexity at most $c2^n/n$. If $S : \mathbb{N} \to \mathbb{N}$ and S is $o(2^n/n)$ then* $\mathsf{SIZE(S(n))} \subsetneq \mathsf{SIZE(4cS(n))}$.

Proof. Let $m \leq n$ be the smallest integer such that $2^{m-1}/m - 1 \leq 2S(n) < 2^m/m$. We consider the set of functions that depend only on the first m bits. By definition $2^m/m \leq 4S(n)$ and hence by the upper bound, every function on the first m bits can be computed in size at most $4cS(n)$. However, by Theorem 1.1, size $S(n) \leq 2^{m-1}/m$ is not large enough to compute all such functions. □

2 Hierarchy Theorems for DTIME and NTIME

Theorem 2.1. *Let $f, g : \mathbb{N} \to \mathbb{N}$. If g is time-constructible and $f(n)log_2 f(n)$ is $o(g(n))$ then*

$$\mathsf{DTIME(f(n))} \subsetneq \mathsf{DTIME(g(n))}.$$

Proof. The general idea of the proof follows by a variant of the diagonalization that is used to prove the undecidability of the halting problem. That argument uses a listing of all Turing machines $M_1, M_2, \ldots$, associates a distinct input (either i or $[M_i]$) with each Turing machine and creates a "diagonal" language that differs from the result of M_i on input i (or $[M_i]$).

For the present argument we need a listing that captures all $O(f(n))$ time-bounded Turing machines but does not include all Turing machines running in time $O(g(n))$.

On input x of length n, the $O(g(n))$ time-bounded algorithm begins by computing $g(n)$ using the time-constructibility of g and then computing $h(n) = g(n)/\log_2 g(n)$. By assumption that $f(n)\log_2 f(n)$ is $o(g(n))$, $f(n)$ is $o(h(n))$. That is, $h(n)$ is eventually larger than $cf(n)$ for any constant c. That is, for all $c > 0$, there is an n_c such that $n \geq n_c$ implies that $h(n) \geq cf(n)$.

The fact that we do not know *a priori* what size input will allow $h(n)$ to be larger than $cf(n)$ makes it necessary to dedicate an infinite set of input strings to each possible Turing machine in order to ensure that the input length on which we make the diagonal language different is large enough.

The diagonal language D we define is

$$D = \{[M]01^k \mid k \geq 0 \text{ and } M \text{ does } \textbf{not} \text{ accept } x \text{ in at most } h(|[M]01^k|) \text{ steps}\}.$$

We first show that $D \notin \mathrm{DTIME}(f(n))$: Let M be any Turing machine that runs in time at most $cf(n)$ for some constant c. Let n_c be the constant such that $h(n) \geq cf(n)$ for all $n \geq n_c$. Because $x = [M]01^{n_c}$ is long enough, the time bound $h(|x|)$ is no constraint on M's behavior on input x. Therefore $x \in D$ if and only if M does not accept x. Therefore D is not the language decided by M.

We now need to show that $D \in \mathsf{DTIME(g(n))}$. Here is the description of the algorithm: On input x first compute $h(|x|)$ as above on an extra "clock" tape using time $O(g(|x|))$. Then reject if x is not of the form $[M]01^k$, extract $[M]$ from the input, and then run the universal Turing machine $\mathcal{U}$ on input x and $[M]$, while subtracting 1 from the clock tape for each simulated step of M. The computation stops if the clock tape reaches 0. The simulation takes $O(h(|x|)\log h(|x|))$ steps, which is $O(g(|x|))$. If M accepts during this time, the machine for D rejects; otherwise, it accepts. □

Corollary 2.2. $\mathsf{P} \neq \mathsf{EXP}$.

There is a similar hierarchy for nondeterministic computation. It is sharper for small time bounds but degrades noticeably as the functions grow.

Theorem 2.3. *Let $f, g : \mathbb{N} \to \mathbb{N}$. If g is time-constructible and $f(n+1)$ is $o(g(n))$ then*

$$\mathsf{NTIME(f(n))} \subsetneq \mathsf{NTIME(g(n))}.$$

Proof. We describe the differences between this argument and that for the deterministic time hierarchy. In this case, the universal Turing machine is a nondeterministic machine $\mathcal{U}'$ that on input x, NTM $[M]$ and time bound T simulates M on input x for at most T steps. As you have shown on your homework, $\mathcal{U}'$ takes time $O(T)$. This eliminates the extra $\log_2 f(n)$ factor allows the simulation to use the function $g(n)$ instead of the function $h(n)$ used for the deterministic hierarchy theorem.

However, because M is an NTM, we cannot simply have the simulation flip the answer along each path. (If there are both accepting and rejecting computation paths, then flipping the answers will merely leave the input as accepted.) Simulating and flipping the answer for an input x of length n in general requires $2^{O(f(n))}$ time which is at most time $2^{g(n)}$ for sufficiently large n.

The trick to making this all work is a form a "lazy" diagonalization that only flips the answer by using exponentially larger inputs. The analog of each input of the form $[M]01^k$ in the diagonal language D will use a consecutive sequence of input lengths from n to $2^{g(n)}$. (We can easily assume that the length n (or indeed any input with length in the range from n to $2^{g(n)}$ is uniquely decodable as a pair $[M]$ and k. All inputs in the range we consider will be associated with fooling M.) For convenience, the only inputs in this range that we consider for membership in D will be $1^n, 1^{n+1}, \ldots, 1^{2^{g(n)}}$.

For m in the range $n \leq m \leq 2^{g(n)}$, the NTM unique decodes input 1^m to find n and the NTM $[M]$. It then computes $g(m)$ on $O(g(m))$ steps using the time-constructibility of g.

If $n \leq m < 2^{g(n)}$ the NTM defining D computes $g(m)$ in $O(g(m))$ steps using the time-constructibility of g. It simulates NTM $[M]$ on input 1^{m+1} for $g(m)$ steps and does exactly what M does. (It does **not** flip the answer.) Note that since $f(n+1)$ is $o(g(n))$, for large enough m, $cf(m+1) \leq g(m)$ and so the time bound does not restrict the computation. However, when $m = 2^{g(n)}$, the NTM for D computes $g(n)$ and does a deterministic simulation of NTM M on input 1^n for $g(n)$ steps and flips the answer.

We now argue that if n is large enough so that $g(n) \geq cf(n)$ where the running time of NTM M on is at most $cf(n)$, then M and D disagree about some input in the range $1^n, \ldots, 1^{2^{g(n)}}$: Suppose that M and D agree on all strings 1^m with $n \leq m < 2^{g(n)}$. By construction $D(1^m) = M(1^{m+1})$ for all m in this range. In addition, since D and M agree, then $D(1^m) = M(1^m)$ for all m with $n \leq m < 2^{g(n)}$; in particular, $M(1^n) = M(1^{2^{g(n)}})$. However, by construction, $D(1^{2^{g(n)}}) \neq M(1^n)$ and so $D(1^{2^{g(n)}}) \neq M(1^{2^{g(n)}})$ which implies that D is not decided by M with running time $g(n)$ (and hence not with time bound $O(f(n+1))$ either).

The rest of the proof is analogous to the deterministic case. □

Corollary 2.4. $\mathsf{NP} \neq \mathsf{NEXP}$.