CS 6810: Theory of Computing **Fall 2021**

Lecture 9: Sept 23, 2021

Lecturer: Eshan Chattopadhyay *Scribe: Matthew Eichhorn*

1 The computational power of P/poly

Given a function $S : \mathbb{N} \to \mathbb{N}$, we define the set $\mathrm{Size}(S(n))$ as the languages which can be computed by circuit families bounded by $S(n)$. That is, a language $L \in \mathrm{Size(S(n))}$, if for each $n \in \mathbb{N}$, there is a boolean circuit with $S(n)$ wires that determines whether each $x \in \{0,1\}^*$ belongs to L.

We define the class of languages P/poly as,

$$\mathsf{P}/\mathrm{poly} := \bigcup_{c \in \mathbb{N}} \mathrm{Size}(n^c).$$

Theorem 1.1. $\mathsf{P} \subsetneq \mathsf{P}/\mathrm{poly}$.

Proof. We can split the theorem into two separate claims.

1. $\mathsf{P} \subseteq \mathsf{P}/\mathrm{poly}$,
2. There is a language $L \in \mathsf{P}/\mathrm{poly}$ which does not belong to P.

For the first claim, the proof is very similar to that of the Cook Levin Theorem. Given a Turing machine M running in $O(n^c)$ time, we can simulate M by an oblivious Turing machine $\hat{M}$ running in $O(n^c \log n^c) = O(n^{c+1})$ time, where the movements of the tape heads does not depend on the tape contents. For a particular $n \in \mathbb{N}$, a transcript of M's execution consists of $T(n)$ snapshots, each encoding the machine's state and the symbols of its heads. Each of these snapshots is a deterministic function of a constant number of previous snapshots, which is also easily computed. Therefore, we can use a constant-sized circuit to construct each successive snapshot. Upon reaching the final snapshot, we can use a final constant-sized circuit to check whether the snapshot represents an accepting state of $\hat{M}$. Wiring these smaller circuits together results in a circuit with size polynomial in n. Putting together these circuits for all $n \in \mathbb{N}$, we have a polynomial-sized circuit family that recognizes the same language as M.

For the second claim, we in fact prove something stronger. We show that there is an undecidable language in P/poly.

A language $L \in \{0,1\}^*$ is unary if $L \subseteq \{1\}^*$. Any unary language can be computed by a $\mathrm{Size}(n+2)$ circuit family: For each $n \in \mathbb{N}$, if $1^n \in L$, then let C_n be the circuit that feeds all n input bits into an n-ary AND gate, and outputs this result. This will accept only the string 1^n. If $1^n \notin L$, then L includes no n-bit strings. In this case, we can take C_n to be the size-3 circuit which outputs $(x_1 \wedge \overline{x_1})$, thereby rejecting every string.

Now, consider a unary encoding of the language of the halting problem, $\mathsf{HALT} = \{\langle M, x\rangle : M \text{ halts on input } x\}$. That is, consider an enumeration of all $\langle$Turing machine, string$\rangle$ pairs, and let L be the unary language where $1^n \in L$ if and only if the n'th pair in the enumeration is in HALT. L is undecidable because it reduces to the halting problem. However, $L \in \mathsf{P}/\mathrm{poly}$ by the previous paragraph. $\square$

Note that the added power of P/poly comes from our ability to specify a different circuit for each input size. The particular families of circuits that we considered were examples of *non-uniform* circuit families: families $\{C_n\}_{n\in\mathbb{N}}$ for which there is no Turing machine that, upon input n, returns a description of circuit C_n.

Next, we state a strongly-believed conjecture that relates P/poly and NP.

Conjecture 1.2. $\mathsf{NP} \not\subseteq \mathsf{P}/\text{poly}$. *That is, there exists some language $L \in \mathsf{NP}$ that is recognized only by super-polynomial sized circuit families.*

If this conjecture is true, then it would imply that $\mathsf{P} \neq \mathsf{NP}$, since $P \subseteq \mathsf{P}/\text{poly}$. Note that this conjecture establishes a "circuit lower bound", as it ensures that a circuit must have a sufficiently large size in order to recognize a language. In the following section, we establish more circuit bounds.

2 Circuit Bounds

2.1 Upper Bound Results

We'll establish three, progressively tighter upper bounds on circuit size. That is, we'll show that there is are $\text{Size}(T(n))$ circuit families that are capable of recognizing *any* language, for three increasingly-tight functions $T : \mathbb{N} \to \mathbb{N}$.

Lemma 2.1. *Any language $L \in \{0,1\}^*$ is in $\text{Size}(O(n \cdot 2^n))$.*

Proof. Fix $n \in \mathbb{N}$, and let $L_n := L \cap \{0,1\}^n$. Let $f_n : \{0,1\}^n \to \{0,1\}$ be the boolean function with $f_n(x) = 1 \iff x \in L_n$. Then, we wish to design a circuit that computes the function f_n.

Consider the truth table for f_n. For example,

x_1	x_2	x_3	$\dots$	x_n	f_n
0	0	0	$\dots$	0	1
0	1	0	$\dots$	0	0
1	1	1	$\dots$	1	1

For each truth table row with $f_n = 1$, we can construct an n-way conjunction that checks whether all input bits agree with that row. That is, in the i'th row of the table, $i = (x_n\ x_{n-1}\ \dots\ x_2\ x_1)_2$ we have the conjunction

$$C_i = \bigwedge_{j:x_j=1} x_j \wedge \bigwedge_{k:x_k=0} \overline{x_k}.$$

Then, f_n can be represented as the disjunction $f_n = \bigvee_i C_i$. This disjunction includes at most 2^n conjunctions, in the case that f_n includes all length-n binary strings. The total number of wires used for this computation is bounded above by $\frac{3}{2} \cdot n \cdot 2^n + 2^n + 1 = O(n \cdot 2^n)$. $\square$

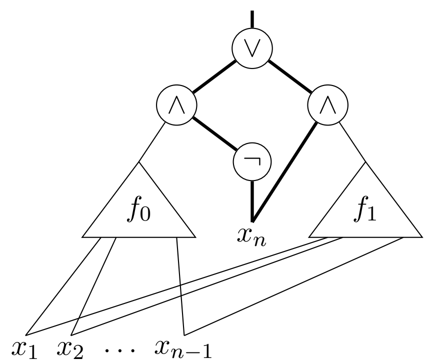
To improve upon this result, we consider handling the variables one at a time.

Lemma 2.2. *Any language $L \in \{0,1\}^*$ is in* Size$(O(2^n))$.

Proof. Similar to above, fix $n \in \mathbb{N}$, let $L_n := L \cap \{0,1\}^n$, and let $f_n : \{0,1\}^n \to \{0,1\}$ be the boolean function with $f_n(x) = 1 \iff x \in L_n$. Note that,

$$f(x_1, x_2, \ldots, x_n) = \Big(\underbrace{f(x_1, x_2, \ldots, x_{n-1}, 0)}_{f_0(x_1, x_2, \ldots, x_{n-1})} \wedge \overline{x_n}\Big) \vee \Big(\underbrace{f(x_1, x_2, \ldots, x_{n-1}, 1)}_{f_1(x_1, x_2, \ldots, x_{n-1})} \wedge x_n\Big).$$

We can apply this same decomposition on f_0 and f_1, and recurse $n-2$ additional times until we have taken care of all of the variables. A diagram for the first level of this decomposition is below.



Note that the number of wires in the circuit of size n is equal to 2 times the number of wires in the curcuit of size $n-1$, plus 6 additional (bolded) wires. Therefore, the sizes of these circuits can be upper bounded by,

$$T(1) = 2, \qquad T(n) = 2 \cdot T(n-1) + 6.$$

Solving this recurrence, we find that $T(n) = 2^{n+2} - 6 = O(2^n)$. □

To improve this bound one final time, we note that the number of "sub-circuits" is increasing exponentially in each decomposition step of the above proof. Thus, at some point, there will be sub-circuits than possible functions that they can compute. We can reduce the circuit size by cleverly curtailing the decomposition.

Lemma 2.3. *Any language $L \in \{0,1\}^*$ is in* Size$\Big(O\big(\frac{2^n}{n}\big)\Big)$.

Proof. Suppose that we unfold the recursion from the previous lemma for $n-k$ steps (for some constant k that we will fix later). We have used $O(2^{n-k})$ wires so far, and are left to consider functions of k variables. There are 2^{2^k} such functions, each of which can be computed by a circuit of size $O(2^k)$ (by the previous lemma). This gives a total circuit size of

$$O\Big(2^{2^k} \cdot 2^k + 2^{n-k}\Big).$$

Let $k = \log_2 n - 1$. Then, this simplifies to

$$O\Big(\frac{n}{2} \cdot (\sqrt{2})^n + \frac{2^n}{2n}\Big) = O\Big(\frac{2^n}{2n}\Big).$$

□

In fact, this bound is tight; there exists languages that is only computable by circuits of size $\Omega\big(\frac{2^n}{n}\big)$.

2.2 A Lower Bound Result

Lemma 2.4. *There is a sufficiently large constant c such that there is some language $L \in \{0,1\}^*$ that is not in* Size $\left(\frac{2^n}{c \cdot n}\right)$.

Proof. We establish this claim with a counting argument; namely, we find c such that the number of circuits of size $\frac{2^n}{c \cdot n}$ is smaller than the number of functions $\{0,1\}^n \to \{0,1\}$.

The number of such functions is 2^{2^n}, as describing such a function amounts to selecting a subset of $\{0,1\}^n$ to map to 1, and $|\{0,1\}^n| = 2^n$.

To count the number of size s circuits on n input variables, we consider their bit representations. Such a circuit has $O(s)$ gates, whose labels can be each described by a constant number of bits. Each of the $O(s)$ wires is described by its 2 endpoints, which are each identified by $O(\log s)$ bits. Therefore, the description requires $O(s \log s)$ bits, meaning there are $2^{O(s \log s)} = s^{O(s)}$ such circuits. We can choose $c' \in \mathbb{N}$ such that this number of circuits is at most $s^{c' \cdot s}$ for all s.

Now, we substitute $s = \frac{2^n}{c \cdot n}$:

$$\left(\frac{2^n}{c \cdot n}\right)^{\frac{c' \cdot 2^n}{c \cdot n}} = \left(\frac{2}{c \cdot n}\right)^{\frac{c' \cdot 2^n}{c}} \leq \left(2^{2^n}\right)^{\frac{c'}{c}}.$$

Therefore, taking $c = c' + 1$ ensures that there are more functions $\{0,1\}^n \to \{0,1\}$ than circuits of size s. □

3 Turing Machines with Advice

Just as with oracle Turing machines, it can often be useful to endow a Turing machine with some additional power and then study how this expands what it is able to compute. Here, we introduce the concept of a Turing machine that "takes advice". Then we compare the power of Turing machines with advice to polynomial-sized boolean circuit families.

Let $T, a : \mathbb{N} \to \mathbb{N}$ be functions. Then, the class of languages decidable by a $T(n)$-time Turing machine with $a(n)$ advice is denoted $\mathsf{DTIME}(T(n))/a(n)$. A language $L \in \mathsf{DTIME}(T(n))/a(n)$, if there exists a sequence $\{\alpha_n\}_{n \in \mathbb{N}}$ of *advice strings*, with each $\alpha_n \in \{0,1\}^{a(n)}$, and a Turning machine M for which

$$M(\langle x, \alpha_n \rangle) = 1 \iff x \in L$$

for every $x \in \{0,1\}^n$ and the computation of $M(\langle x, \alpha_n \rangle)$ requires at most $O(T(n))$ steps.

Theorem 3.1. $\mathsf{P}/\text{poly} = \displaystyle\bigcup_{c,d \in \mathbb{N}} \mathsf{DTIME}(n^c)/n^d$.

Proof. For the forward containment, suppose that $L \in \mathsf{P}/poly$. Consider an input string x, with $|x| = n$. Let α_n be a description of the size-n circuit that recognizes $L \cap \{0,1\}^n$. Note that this circuit contains polynomially-many wires, so has a polynomial description. Upon receiving input $\langle x, \alpha_n \rangle$, the Turing machine M should simulate the circuit described by α_n on input x, which can be done in polynomial time. Since this is true for all $x \in \{0,1\}^*$, $L \in \bigcup_{c,d \in \mathbb{N}} \mathsf{DTIME}(n^c)/n^d$.

For the reverse containment, suppose that $L \in \bigcup_{c,d \in \mathbb{N}} \mathsf{DTIME}(n^c)/n^d$. For any $n \in \mathbb{N}$ we can use the construction from the proof of Theorem 1.1 to construct a polynomial-sized circuit C_n that outputs the same value as machine M on $\langle x, \alpha_n \rangle$. Since the value of the advice bits is fixed, we can modify C_n into a circuit C'_n that takes only x as input. We do this by "hard-wiring" these values of α_n into the circuit. With a constant number of wires, we can add gates to our circuit that deterministically output 0 and 1 (e.g. $0 = x_1 \wedge \neg x_1, 1 = x_1 \vee \neg x_1$). Then, we replace all wires that pass an input bit from α_n into a gate with a wire that passes from the constant gate corresponding to its value. Since this transformation only adds constantly many more wires, the size of C'_n remains polynomial. Since each C'_n recognizes $L \cap \{0,1\}^n$, $L \in \mathsf{P}/\text{poly}$. $\square$