CS 6810: Theory of Computing **Fall 2023**

Lecture 10: August 21, 2023

Lecturer: Eshan Chattopadhyay *Scribe: Esther Wang*

1 Circuit Upper Bounds

Theorem 1.1. *Any function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a $\frac{2^n}{cn}$ sized circuit for some constant c.*

Last lecture we saw that such a function can be computed with a $cn2^n$ sized circuit simply by determining the truth table for f, and writing out the corresponding CNF or DNF. We show that we can improve this bound to $\frac{2^n}{cn}$.

We begin by observing that for some $f(x_1, x_2, ...x_n)$, x_n can either be 0 or 1. We define 2 new functions, f_0, f_1 such that

$$f_0(x_1, ...x_{n-1}) = f(x_1, ...x_{n-1}, 0)$$

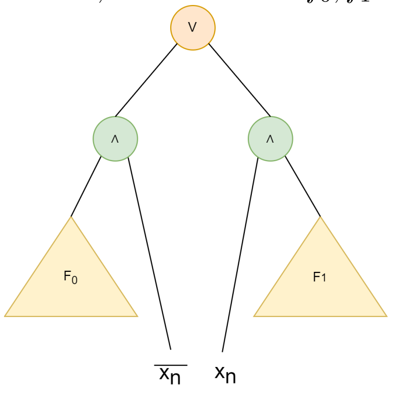
and similarly

$$f_1(x_1, ...x_{n-1}) = f(x_1, ...x_{n-1}, 1)$$

Then, it holds that

$$f = (f_0 \wedge \overline{x_n}) \vee (f_1 \wedge x_n)$$

That is, subcircuits for f_0, f_1 can be combined as follows into a circuit for f:

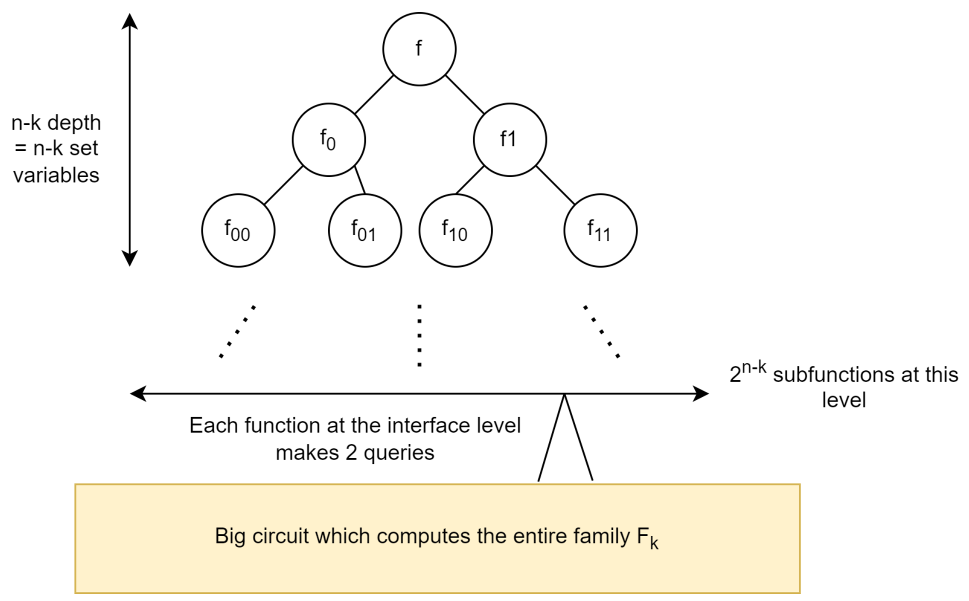


We note that this requires 6 new wires. This gives us the following recurrence for circuit size:

$$T(n) = 2T(n-1) + O(1) = C * 2^n$$

knowing that $T(1) = O(1)$. Then, we can determine that recursively constructing subcircuits in this manner allows us to construct a circuit for a function with n inputs using $c * 2^n$ wires.

This bound is clearly not good enough yet. We make the following observation: we can use this recursive construction of smaller subcircuits, up until a certain depth level k, and brute force compute all functions for any smaller inputs. The following graphic illustrates the general idea:


n-k depth = n-k set variables
f
f0
f1
f00
f01
f10
f11
2^{n-k} subfunctions at this level
Each function at the interface level makes 2 queries
Big circuit which computes the entire family F_k

That is, we recursively set the last variable for $n-k$ levels of the recursive tree, using the construction described above. There are 2^{n-k} subfunctions at this point. There are still k unset variables. We define the following:

$$\mathcal{F}_k = \text{ family of all functions } \{0,1\}^k \to \{0,1\}$$

We know that $|\mathcal{F}_k| = 2^{2^k}$, and that each $g \in \mathcal{F}_k$ has a $c_1 * 2^k$ sized circuit. Therefore, we can construct a big circuit for the entire family $\mathcal{F}_k$ using $c_1 * 2^{2^k} * 2^k$ wires.

Now consider the functions at depth $n-k$. There are 2^{n-k} such functions. Each needs to make 2 queries to the circuit computing $\mathcal{F}_k$. Therefore, the total number of wires required consists of the wires required for the circuit computing $\mathcal{F}_k$, the wires at the interface level, and the unrolling of $n-k$ variables. That is,

$$c_1 * 2^{2^k} * 2^k + 2 * 2^{n-k} + c_2 * 2^{n-k} = c_1 * 2^{2^k} * 2^k + c_3 * 2^{n-k}$$

for some constants c_1, c_2, c_3.

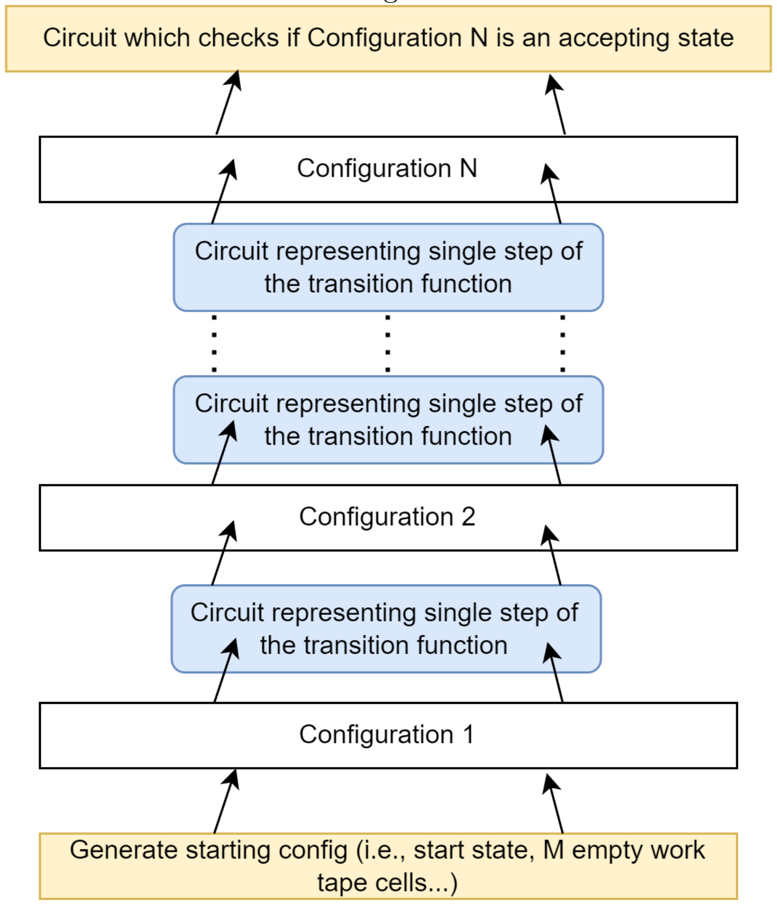
Now, let $k = \log(n) - 1$. Then,

$$\begin{aligned} c_1 * 2^{2^k} * 2^k + c_3 * 2^{n-k} &= c_1 * 2^{2^{\log(n)-1}} * 2^{\log(n)-1} + c_3 * 2^{n-\log(n)-1} \\ &= \frac{2^n}{cn} \end{aligned}$$

Theorem 1.2. *P* $\subseteq$ *P/poly*

Let $L \in P$. Then, there exists some TM M computing L in time n^c. We assume a uniform number of steps $N \in O(n^c)$ on any input of a given length. There are overheads which achieve this, such as a quadratic overhead to make an arbitrary TM oblivious. Therefore, on some input of length x, we can represent configuration states in $P \in O(n^c)$ bits (tape head locations, tape cell contents - the number of which is bounded by the time M takes in computation, state of M).

Now, we observe that the configuration in step i can be computed from the configuration in step $i-1$ with a poly-sized (on P) circuit representing the transition function of M. Therefore, our circuit consists of the following:


Circuit which checks if Configuration N is an accepting state
Configuration N
Circuit representing single step of the transition function
Circuit representing single step of the transition function
Configuration 2
Circuit representing single step of the transition function
Configuration 1
Generate starting config (i.e., start state, M empty work tape cells...)

We add in poly-sized circuits that just generate initial configuration, and check if the Nth state is an accepting state. Since each circuit is poly-sized, and there are polynomially many such circuits, this is a poly-sized circuit computing L.

2 Turing Machines With Advice

Definition 2.1. *$L \in \texttt{DTIME}(T(n))/\texttt{a}(n)$ if there exists a TM M that runs in $O(T(n))$ time and some $\{\alpha_n\}_{n\in\mathbb{N}}$, $\alpha_n \in \{0,1\}^{a(n)}$ such that $x \in L \iff M(x, \alpha_{|x|}) = 1$.*

Theorem 2.2. $\texttt{P/poly} = \bigcup_{c_1,c_2\in\mathbb{N}} \texttt{DTIME}(n^{c_1})/n^{c_2}$.

Suppose L has a poly-sized circuit family $\mathcal{C}$ computing it. We let our "advice" sequence simply be our circuits; that is, $\{\alpha_i = C_i\}$ where C_i is the circuit in $\mathcal{C}$ for inputs of size i. We construct turing machine M such that on input x, C_n, if $|x| = n$, it simulates C_n on x and returns the result. This is possible because by definition C_n must be polysized.

In the other direction, suppose we have some polytime TM M which computes L with polysized advice. From Theorem 1.2 we know we can construct a polysized circuit for M without the advice. However, since the advice is poly-sized, for every α_i advice for inputs of size i, we construct a separate (fixed) circuit for the advice, taking only inputs of 0,1. This circuit will also be polysized by virtue of the fact that α_i must be polynomial in length. We attach each of these to our circuit for M, to create our poly-sized circuit family.

3 Circuit Lower Bounds

In the next few lectures we will study the circuit class AC^0 (defined below) and prove strong lower bounds against this class. While AC^0 is itself an interesting class, some of the motivation for studying lower bounds for this class stems from the interesting technical tools that have been developed over several decades to prove such lower bounds.

Definition 3.1. *AC^0 is the class of constant-depth polynomial-sized circuits with unbounded fan-in, consisting of AND, OR and NOT gates.*

Over the next couple of lectures we will prove the following.

Theorem 3.2. *$MAJORITY \notin AC^0$*

Theorem 3.3. *$PARITY \notin AC^0$*

On a high level, the proof approaches are based on finding weaknesses of AC^0 circuits that are not exhibited by PARITY or MAJORITY. We will make this concrete in upcoming lectures - in particular, we will see two very different approaches for proving lower bounds against AC^0.

The first approach is based on showing that circuits in AC^0 can be approximated by low-degree polynomials. In contrast, the second approach will be based on showing that AC^0 circuits 'simplify' when a random subset of coordinates are set to uniformly random values. We leave more details to future lectures.