Circuit Complexity Notes

Caleb Koch

August 2019

Contents

Monotone circuit lower bounds

Definition 0.1 (Monotone functions)**.** For $x, y \in \{0,1\}^n$ we say $x \leq y$ if $x_i \leq y_i$ for $i \in [n]$. We say that a function $f : \{0,1\}^n \to \{0,1\}$ is monotone if $f(x) \leq f(y)$ for every $x \leq y$.

Given x, the only way to form y such that $x \leq y$ is to change some (possibly 0) bits that are 0 in x to 1. Thus an equivalent characterization is that f is monotone if, whenever $f(x) = 1$, flipping a 0 bit in x cannot change the value of the function.

A Boolean circuit is *monotone* if it doesn't contain any NOT gates.

Proposition 0.2. *Every monotone function f is computed by a monotone circuit and every monotone circuit computes a monotone function.*

Proof. For the first statement, let f be monotone. Then f is computed by the circuit

$$\bigvee_{x:f(x)=1} x_{i_1} \wedge \cdots \wedge x_{i_j}$$

where $i_1, ..., i_j$ are exactly the bits that are 1 in x. The circuit is clearly monotone since there are no negations.

We prove the second statement by induction. In the base case, the circuit consists just of an input gate which computes a function $f(x) = x_i$ or a constant gate $f(x) = c$ for $c \in \{0,1\}$. Both are clearly monotone. Otherwise suppose inductively that an AND or OR gate have inputs wires $1, ..., n$ corresponding to monotone functions $f_1, ..., f_n$. Suppose that $f_1(x) \odot \cdots \odot f_n(x) = 1$ for $\odot \in \{\wedge, \vee\}$. Then in order to make this function evaluate to 0, one would have to change a bit in x so that there is at least one i such that $f_i(x) = 1$ but f_i evaluates to 0 on the new input. As f_i is monotone, this cannot be done by flipping a 0 bit in x. □

Given a graph G it is natural to ask whether G has a k-clique. This is formalized as a function $\text{CLIQUE}_{n,k} : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ which takes as input a string representing which edges are present in an n-vertex graph and outputs 1 iff the corresponding graph has a k-clique.[1] This function is monotone since if there is a k-clique, its existence won't be affected by adding more edges. In 1985, Razborov released a breakthrough result that gave a superpolynomial monotone circuit lower bound for $\text{CLIQUE}_{n,k}$ [Raz85]. A few years later in 1987, Alon and Boppana improved this lower bound by showing that no subexponentially sized monotone circuits can compute $\text{CLIQUE}_{n,k}$ [AB87].

[1] We do not consider graphs with isolated vertices so that the graph can be fully specified with its edges.

Theorem 0.3 (Lower bound for $\text{CLIQUE}_{n,k}$ [Raz85],[AB87])**.** *Let $k \leq n^{1/4}$, then any monotone circuit computing $\text{CLIQUE}_{n,k}$ must have size $n^{\Omega(\sqrt{k})}$*

Our proof here follows the exposition in [AB09] and the lecture notes in [Ais13], the former of which appears to be based on [Juk12].

Note that this bound gives an exponential lower bound by picking $k = n^{1/4}$.

Definition 0.4 (Positive/negative test graph)**.** A positive test graph with respect to k on n vertices is a graph that is isomorphic to K_k, a k-complete graph (i.e. the graph has a clique of size k and no other edges). Likewise a negative test graph is formed from a coloring $c : [n] \to [k-1]$ such that u, v have an edge between them iff $c(u) \neq c(v)$ (i.e. all vertices with different colors have edges between them).

Note that for a randomly chosen positive test graph Y, we have $\text{CLIQUE}_{n,k}(Y) = 1$ and likewise for a negative test graph N we have $\text{CLIQUE}_{n,k}(N) = 0$ both with probability 1.

The main proof technique will be to show that if a monotone circuit is "too small" then it will either assign 0 to a large portion of positive test graphs or it will assign 1 to a large portion of negative test graphs. Hence $\text{CLIQUE}_{n,k}$ cannot have monotone circuits that are "too small."

To do this, we approximate the circuit by replacing $\wedge, \vee$ gates with new gates $\sqcap, \sqcup$. The construction of the approximate circuit involves picking sunflowers.

We start with the sunflower lemma which was originally published in 1960 by Erdős and Rado [ER60].

Lemma 0.5 (Sunflower lemma [ER60])**.** *Let $\mathcal{Z}$ be a collection of distinct sets of cardinality at most ℓ. If $|\mathcal{Z}| > (p-1)^\ell \ell!$ for $p \geq 2$ then $\mathcal{Z}$ contains a sunflower $Z_1, ..., Z_p \in \mathcal{Z}$ with p petals.*

A sunflower is a collection of sets such that the pairwise intersection of those sets is constant, i.e. there is a Z such that $Z_i \cap Z_j = Z$ for all i, j. The set Z is thought to be the middle part of a sunflower with each set Z_i being a petal.

Proof. The proof is by induction on ℓ. For $\ell = 1$, we have $|\mathcal{Z}| > (p-1)$. Thus we can take $Z_1, ..., Z_p$ to be singleton sets each consisting of a unique element so that the common intersection is $\varnothing$.

Now suppose $\ell \geq 2$. Let $Z_1, ..., Z_t$ be the maximal subcollection of sets in $\mathcal{Z}$ with an empty center. Assume that $t < p$ since otherwise, the hypothesis is satisfied. Note that since each $|Z_i| \leq \ell$ we have

$$\left| \bigcup_{i=1}^{t} Z_i \right| \leq t\ell \leq (p-1)\ell.$$

Note that by maximality, every $Z \in \mathcal{Z}$ has a nonempty intersection with this union. Hence in particular, given $Z \in \mathcal{Z}$, a randomly chosen element from the union is in Z with probability at least $\frac{1}{(p-1)\ell}$ and thus at least $\frac{1}{(p-1)\ell}$ of the points in the union are in Z. Since this holds for an arbitrary Z by an averaging argument we get that there is some $x \in \cup_{i=1}^{t} Z_i$ that appears in at least a $\frac{1}{(p-1)\ell}$ fraction of the sets in $\mathcal{Z}$.

Let $F = \{Z \in \mathcal{Z} : x \in Z\}$. Then

$$|F| \geq \frac{|\mathcal{Z}|}{(p-1)\ell} > (p-1)^{\ell-1}(\ell-1)!.$$

Then let $G = \{Z \setminus \{x\} : x \in Z\}$. Note that for each $Z' \in G$ we have $|Z'| \leq \ell - 1$ and also $|G| = |F| > (p-1)^{\ell-1}(\ell-1)!$. Hence by the induction hypothesis there is a sunflower with p petals, $T_1, \ldots, T_p$ of G. Then $T_1 \cup \{x\}, \ldots, T_p \cup \{x\}$ is a p petal sunflower of $\mathcal{Z}$. □

A few weeks before this was written, a preprint was published which improved the $(p-1)^\ell \ell!$ lower bound on $|\mathcal{Z}|$ to $(\log \ell)^\ell$. Indeed it is conjectured that the log factor can be improved to a constant [ALWZ19].

We need one more definition before getting to the main proof.

Definition 0.6 (Clique indicators)**.** A clique indicator Φ_S for a set of vertices $S \subseteq [n]$ is a function $\Phi_S : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ where $\Phi_S(G) = 1$ iff the vertices in S form a clique in G.

Proof of Theorem 0.3. Let C be an arbitrary monotone circuit. We construct a new circuit $\hat{C}$ inductively by replacing gates $\wedge$ with $\sqcap$ and $\vee$ with $\sqcup$ where these are defined below. The approximators take the form of ORs of clique indicators. In particular, we require each approximator to be the OR of at most m clique indicators for some fixed $m > 0$ where each clique indicator has size at most ℓ (we leave these parameters unspecified for now, but not that ℓ will be less than k and m will be at least $(p-1)^\ell \ell!$ for a carefully chosen p).

The construction of $\hat{C}$ from C proceeds inductively. In the base case C is a single input gate $x_{i,j}$. Then we let the approximator be $\Phi_{\{i,j\}}$.

For the inductive step consider the case $C = C_1 \vee C_2$. Let $A = \Phi_{S_1} \vee \cdots \vee \Phi_{S_i}$ and let $B = \Phi_{S_{i+1}} \vee \cdots \vee \Phi_{S_{i+j}}$ be the approximators for C_1 and C_2. Consider the approximator $\Phi_{S_1} \vee \cdots \vee \Phi_{S_{i+j}}$. If $i + j \leq m$ then this is our approximator for $C_1 \vee C_2$. Otherwise, we need to use the sunflower lemma to reduce the number of sets in this OR. In particular if $S_{q_1}, \ldots, S_{q_p}$ form a sunflower with center S then we can

replace the clique indicators $\Phi_{S_{q_1}}, \ldots, \Phi_{S_{q_p}}$ with the clique indicator Φ_S. We repeat this step until $i + j \leq m$. We call the new approximator $A \sqcup B$.

Now consider the case $C = C_1 \wedge C_2$. Let $A = \Phi_{S_1} \vee \cdots \vee \Phi_{S_i}$ and let $B = \Phi_{S_{i+1}} \vee \cdots \vee \Phi_{S_{i+j}}$ be the approximators for C_1 and C_2. Naively we have

$$A \wedge B = \bigvee_{1 \leq q \leq i < r \leq i+j} \Phi_{S_q} \wedge \Phi_{S_r}.$$

We transform this into the approximator

$$\bigvee_{1 \leq q \leq i < r \leq i+j} \Phi_{S_q \cup S_r}.$$

We discard any Φ_Z in this approximator with $|Z| > \ell$. Moreover we pluck sunflowers until the total number of clique indicators is $\leq m$ as in the $\vee$ case. We call the resulting approximator $A \sqcap B$.

This completes the construction of the approximator $\hat{C}$ of C. Now it remains to show that $\hat{C}$ closely approximates C on positive and negative test graphs if C is small.

Lemma 0.7. *The number of positive test graphs Y such that $C(Y) = 1$ but $\hat{C}(Y) = 0$ is at most*

$$\textsc{size}(C) m^2 \binom{n-\ell-1}{k-\ell-1}.$$

Proof. Our proof proceeds by showing that each gate of C contributes at most $m^2 \binom{n-\ell-1}{k-\ell-1}$ such graphs. The proof proceeds by induction on the circuit. In the base case, the approximators are exactly the circuits they approximate, so no errors are introduced.

Now suppose A, B are approximators of subcircuits feeding into an OR gate. Then we take the OR, $A \vee B$, and pluck sunflowers until we get to the desired number of clique indicators. Each plucking replaces some Φ_S with $\Phi_{S'}$ for $S' \subseteq S$ and hence can only accept more graphs. Thus on positive test graphs, $A \sqcup B$ introduces no errors.

Now suppose that A, B are subcircuits feeding into an AND gate. Note that the formation of $\Phi_{S_q \cup S_r}$ from $\Phi_{S_q} \wedge \Phi_{S_r}$ doesn't introduce any new errors on positive test graphs. Then we discard those clique indicators with $|S_q \cup S_r| > \ell$. We are thus removing at most $(\ell + 1)$ vertices from the set of clique indicators and hence the number of positive graphs which are now falsely rejected is at most

$$\binom{n-\ell-1}{k-\ell-1},$$

one for each possible choice of $k - (\ell + 1)$ vertices outside the clique tested by $S_q \cup S_r$. There are at most m^2 such pairs S_q, S_r and hence we

introduce at most $m^2\binom{n-\ell-1}{k-\ell-1}$ errors. Note that as with the OR case, we introduce no errors by plucking sunflowers from the approximator. □

We prove a likewise lemma for negative test graphs.

Lemma 0.8. *The number of negative test graphs which are rejected by C but accepted by $\hat{C}$ is at most*

$$\text{SIZE}(C)m^2\left(\frac{\binom{\ell}{2}}{k-1}\right)^p(k-1)^n.$$

Proof. The proof format is the same as in the previous lemma. In the base case, we introduce no new errors.

Now suppose A, B are approximators for subcircuits feeding into an OR gate. We form $A \vee B$ which introduces no new errors and then pluck at most m sunflowers until we reach the desired amount of clique indicators. We show that each plucking can only add a few additional negative graphs. In particular, given a sunflower $Z_1, ..., Z_p$ with center Z, we want to bound the number of negative graphs rejected by $\Phi_{Z_1} \vee \cdots \vee \Phi_{Z_p}$ but accepted by Φ_Z. For a given negative graph G if $\Phi_Z(G) = 1$ then all of the vertices in Z are assigned distinct colors. Likewise if $\Phi_{Z_i}(G) = 0$ then at least two vertices have the same color. Then we have

$$\begin{aligned}&\mathbb{P}[\Phi_Z(G) = 1 \wedge \Phi_{Z_1}(G) = 0 \wedge \cdots \wedge \Phi_{Z_p}(G) = 0] \leq \\ &\mathbb{P}[\Phi_{Z_1}(G) = 0 \wedge \cdots \wedge \Phi_{Z_p}(G) = 0 | \Phi_Z(G) = 1] = \\ &\prod_{i=1}^{p}\mathbb{P}[\Phi_{Z_i}(G) = 0|\Phi_Z(G) = 1] \leq \prod_{i=1}^{p}\mathbb{P}[\Phi_{Z_i}(G) = 0]\end{aligned}$$

Note that the conditioning step is necessary to invoke independence since the sets $Z_i \setminus Z$ are disjoint.

where the probabilities are taken over a random negative test graph G. Since $\Phi_{Z_i}(G) = 0$ iff at least two vertices have the same color in Z_i (for two fixed vertices this occurs with probability at most $1/(k-1)^2(k-1) = 1/(k-1)$) , we can write

$$\prod_{i=1}^{p}\mathbb{P}[\Phi_{Z_i}(G) = 0] \leq \prod_{i=1}^{p}\binom{\ell}{2}\frac{1}{k-1} = \binom{\ell}{2}^p\frac{1}{(k-1)^p}.$$

Thus since we are plucking at most m sunflowers and there are $(k-1)^n$ possible colorings, we are introducing at most $m\binom{\ell}{2}^p\frac{1}{(k-1)^p}(k-1)^n$ errors.

The case of $\wedge$ is similar since we only introduce errors when we pluck sunflowers. However, note that in this case we start out with at most an OR of m^2 clique indicators and hence there are at most m^2 pluckings giving us the bound in the lemma statement. □

As a final lemma, we wish to show that any approximator disagrees with $\text{CLIQUE}_{n,k}$ on a lot of graphs.

Lemma 0.9. *Let $A = \Phi_{S_1} \vee \cdots \vee \Phi_{S_q}$ be an approximator. Then either A outputs 0 on all graphs or A outputs 1 on more than*

$$\left(1 - \frac{\binom{\ell}{2}}{k-1}\right)(k-1)^n$$

negative test graphs.

Proof. If $q = 0$ then the disjunction is empty and hence the approximator outputs 0 on every graph. Otherwise, let Φ_S be some clique indicator in the disjunction. The probability that Φ_S rejects a given negative test graph is the same as the probability that two vertices in S are given the same color (since an edge is between two vertices iff they have the same color). Hence by the union bound we have

$$\mathbb{P}[\exists v, v' \in S : c(v) = c(v')] \leq \sum_{v,v' \in S} \frac{1}{k-1} \leq \frac{\binom{\ell}{2}}{k-1}.$$

Hence in particular the probability Φ_S accepts a random negative test graph is at least $1 - \frac{\binom{\ell}{2}}{k-1}$. The result follows since there are $(k-1)^n$ negative test graphs. □

Equipped with these lemmas, we proceed to the main part of the proof. Let C be a circuit computing $\text{CLIQUE}_{n,k}$ for $3 \leq k \leq n^{1/4}$. Define $\ell = \sqrt{k}$, $p = \lceil 10\sqrt{k}\log n \rceil$, and $m = (p-1)^\ell \ell!$. Consider the circuit $\hat{C}$ which approximates C. By Lemma 0.9 either $\hat{C}$ rejects all positive test graphs or $\hat{C}$ accepts more than $\left(1 - \frac{\binom{\ell}{2}}{k-1}\right)(k-1)^n$ negative test graphs. Suppose $\hat{C}$ rejects all test graphs. The total number of positive test graphs Y is $\binom{n}{k}$ since each positive test graph is fully specified by the k vertices in the k-clique. In this case, each such graph satisfies $C(Y) = 1, \hat{C}(Y) = 0$ and so we have by Lemma 0.7 that

$$\binom{n}{k} \leq \textsc{size}(C) m^2 \binom{n-\ell-1}{k-\ell-1}.$$

We compute

$$\frac{\binom{n}{k}}{\binom{n-\ell-1}{k-\ell-1}} = \frac{n(n-1)\cdots(n-\ell)}{k(k-1)\cdots(k-\ell)} \geq \left(\frac{n}{k}\right)^{\ell+1}$$

and

$$\begin{aligned} m &= (p-1)^\ell \ell! \\ &\leq p^\ell (\ell)^\ell = (10\sqrt{k}\log n)^{\sqrt{k}} (\sqrt{k})^{\sqrt{k}} \end{aligned}$$

and so

$$m^2 \leq (100k^2(\log n)^2)^{\sqrt{k}}.$$

Hence,

$$\begin{aligned}\textsc{size}(C) &\geq \left(\frac{n}{k}\right)^{\sqrt{k}} \frac{1}{(10\sqrt{k}\log n)^{2\sqrt{k}}(\sqrt{k})^{2\sqrt{k}}} \\ &= \left(\frac{n}{100k^3(\log n)^2}\right)^{\sqrt{k}} \\ &\geq \left(\frac{n}{100n^{3/4}(\log n)^2}\right)^{\sqrt{k}} = \left(\frac{n^{1/4}}{100(\log n)^2}\right)^{\sqrt{k}}\end{aligned}$$

We use the assumption that $k \leq n^{1/4}$ here.

which is $n^{\Omega(\sqrt{k})}$.

Conversely if $\hat{C}$ accepts more than $\left(1 - \frac{\binom{\ell}{2}}{k-1}\right)(k-1)^n$ negative test graphs, we observe

$$\frac{\binom{\ell}{2}}{k-1} = \frac{\sqrt{k}(\sqrt{k}-1)}{2(k-1)} < \frac{1}{2}.$$

Hence in this case we have from Lemma o.8

$$\begin{aligned}\textsc{size}(C)m^2 \left[\frac{\binom{\ell}{2}}{k-1}\right]^p (k-1)^n &\geq \left[1 - \frac{\binom{\ell}{2}}{k-1}\right](k-1)^n \\ \textsc{size}(C)m^2 \left(\frac{1}{2^p}\right) &\geq \frac{1}{2}.\end{aligned}$$

Thus,

$$\begin{aligned}\textsc{size}(C) &\geq \frac{2^{p-1}}{m^2} \\ &\geq \frac{2^{10\sqrt{k}\log n}}{(100k^2(\log n)^2)^{\sqrt{k}}} \\ &\geq \left(\frac{n^{10}}{100n^{1/2}(\log n)^2}\right)^{\sqrt{k}}\end{aligned}$$

which again shows that $\textsc{size}(C)$ is $n^{\Omega(\sqrt{k})}$. □

Consequences of CLIQUE *lower bound*

Let $f : \{0,1\}^m \to \{0,1\}$ and $g : \{0,1\}^n \to \{0,1\}$ be monotone functions. The function f is a *monotone projection* of g if $f(x_1, \ldots, x_m) = g(\sigma_1, \ldots, \sigma_n)$ for $\sigma_i \in \{0, 1, x_1, \ldots, x_m\}$. Thus a circuit for g can also serve as a circuit for f. It follows that a lower bound on the size of the circuits required for computing f also serves as a lower bound on the size of the circuits for g. Thus, if we let f be $\text{CLIQUE}_{n,k}$ and we carefully pick g we can leverage the lower bound above to get lower bounds for other Boolean functions. In particular, consider the monotone function $\text{HAM}_n : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ which outputs 1 iff the input

graph has a Hamiltonian cycle. It can be shown that $\text{CLIQUE}_{n,k}$ is a monotone projection of $\text{HAM}_{O(n^2)}$ and hence HAM cannot have subexponential circuits. The authors in [AB87] explore a number of other functions $\text{CLIQUE}_{n,k}$ is a projection of, including a version of SAT, vertex cover, and k-colorability.

It's natural to ask if this method can be used for general circuits with NOT gates. The authors in [AM05] show that allowing up to $O(\log \log n)$ NOT gates for a circuit computing k-cliques on a graph with n vertices requires superpolynomial circuits. If one could improve this bound to $O(\log n)$ then NP $\neq$ P.

Alternating circuits with counters

One of the earliest lower bounds in circuit complexity came from showing that PARITY $\notin$ AC^0 which was discovered independently by Furst, Saxe, and Sipser in [FSS84] and Ajtai in [Ajt83]. The superpolynomial bounds of these authors were later improved by (independently) Razborov [Raz85] and Smolensky [Smo87]. A natural question then is how to "minimally" extend AC^0 so that it can compute PARITY. Such a thought leads to the definition of ACC or "alternating circuits with counters." These are formed by allowing MOD_m gates for $m \geq 2$ which evaluate to 0 iff the sum of the inputs is $0 \pmod m$.

Recall that AC^0 is the set of languages which have polynomial-size, constant depth circuits with unbounded fan-in.

Definition 0.10 (ACC^i)**.** A language is in $\text{ACC}^i[m_1, \ldots, m_k]$ if it has a circuit family of depth $O(\log^i n)$ and polynomial-size with unbounded fan-in consisting of AND, OR, and NOT gates as well as the gates $\text{MOD}_{m_1}, \ldots, \text{MOD}_{m_k}$. The class ACC^i consists of all languages in $\text{ACC}^i[m_1, \ldots, m_k]$ for some integers $m_1, \ldots, m_k \geq 2$.

The suggestion to study ACC^0 after the PARITY lower bound appears to be due to Barrington [Bar89]. We will mostly be interested in the class ACC^0. Note that the class $\text{ACC}^0[2]$ is sufficient for computing PARITY (in fact only a single MOD_2 gate is needed). Ryan Williams' celebrated result from 2011 shows that NEXP $\not\subseteq$ ACC^0 [Wil11]. Before this result, one of the best known lower bounds was due to Kannan [Kan82].

We follow and expand on the proof in [Aar16].

Theorem 0.11 (Kannan [Kan82])**.** $\text{NEXP}^{\text{NP}} \not\subseteq \text{P}_{/\text{POLY}}$.

Proof. First we show that $\text{EXP}^{\text{NP}^{\text{NP}}} \not\subseteq \text{P}_{/\text{POLY}}$. By a brute force counting argument due to Shannon [Sha49] one can show that there exist Boolean functions $f : \{0,1\}^n \to \{0,1\}$ that require circuits of size $\Omega(2^n/n)$. A function of the form $f : \{0,1\}^n \to \{0,1\}$ can be identified with the set $\{x \in \{0,1\}^n : f(x) = 1\}$ which has length $\leq 2^n$.

There are 2^{2^n} such sets. Hence such a function (set) can be identified uniquely with 2^n bits. Let $f_n : \{0,1\}^n \to \{0,1\}$ denote the lexicographically first (with respect to its 2^n identifier bitstring) function which requires circuits of size $\geq 2^n/n$. Let f

$$L = \bigcup_{n \geq 1} \{x \in \{0,1\}^n : f_n(x) = 1\}.$$

Then by construction $L \notin \mathrm{P}_{/\mathrm{poly}}$. However we claim that $L \in \mathrm{EXP}^{\mathrm{NP}^{\mathrm{NP}}}$. A subset $L'_n \subseteq \{0,1\}^n$ can be represented by 2^n bits as mentioned above. Thus the language $\mathcal{L} = \cup_{n \geq 1}\{L'_n \subseteq \{0,1\}^n : L'_n \text{ has a circuit of size } < (2^n/n)\}$ is in NP since given a 2^n bit string representing L'_n we can guess a circuit of size $< 2^n/n$ and check in $O(2^n)$ time that the circuit computes L'_n.[2] Moreover, the language $\mathcal{H} = \cup_{n \geq 1}\{H_n \subseteq \{0,1\}^n : \exists L'_n \notin \mathcal{L} \text{ with } L'_n < H_n\}$ where $<$ denotes lexicographic ordering is in $\mathrm{NP}^{\mathrm{NP}}$ since given a 2^n bit string representing H_n we can guess a set $L'_n < H_n$ in NP and check if $L'_n \notin \mathcal{L}$ using our NP oracle. Now we describe an $\mathrm{EXP}^{\mathcal{H}}$ machine for L. On input x our machine does a binary search on length 2^n bit strings. The oracle for $\mathcal{H}$ allows us to check whether we should recurse on a lexicographically smaller or larger string. Since there are 2^{2^n} such strings this binary search takes time 2^n. It follows that $L \in \mathrm{EXP}^{\mathrm{NP}^{\mathrm{NP}}}$ and hence $\mathrm{EXP}^{\mathrm{NP}^{\mathrm{NP}}} \subsetneq \mathrm{P}_{/\mathrm{poly}}$.

[2] Note that since the input size is 2^n this runs in time polynomial in the input.

Now suppose for contradiction that $\mathrm{NEXP}^{\mathrm{NP}} \subseteq \mathrm{P}_{/\mathrm{poly}}$. Then $\mathrm{NP} \subseteq \mathrm{P}_{/\mathrm{poly}}$ and in particular $\mathrm{PH} = \Sigma_2$ by Karp-Lipton. Thus $\Delta_3 \subseteq \Sigma_2$ and $\mathrm{P}^{\mathrm{NP}^{\mathrm{NP}}} = \mathrm{NP}^{\mathrm{NP}}$. By a standard padding argument[3] we get that $\mathrm{EXP}^{\mathrm{NP}^{\mathrm{NP}}} = \mathrm{NEXP}^{\mathrm{NP}}$. But this shows that $\mathrm{EXP}^{\mathrm{NP}^{\mathrm{NP}}}$ has polynomial size circuits which is a contradiction by our aforementioned result. □

[3] See for example [Aar16, Proposition 17]

This lower bound wasn't improved significantly for several years until Buhrman, Fortnow, and Thierauf [BFT98] showed that $\mathrm{MAEXP} \not\subseteq \mathrm{P}_{/\mathrm{poly}}$.[4]

[4] The class MAEXP is the EXP version of MA.

Theorem 0.12 ([BFT98])**.** $\mathrm{MAEXP} \not\subseteq \mathrm{P}_{/\mathrm{poly}}$

Proof. Suppose for contradiction that $\mathrm{MAEXP} \subseteq \mathrm{P}_{/\mathrm{poly}}$. Then as $\mathrm{PSPACE} \subseteq \mathrm{MAEXP}$ we get that $\mathrm{PSPACE} \subseteq \mathrm{P}_{/\mathrm{poly}}$. We show that this latter result implies that $\mathrm{MA} = \mathrm{PSPACE}$. Let $L \in \mathrm{PSPACE}$. To determine if $x \in L$, have Merlin send Arthur the $\mathrm{P}_{/\mathrm{poly}}$ circuit $C_{|x|}$ from the circuit family for L. Then using the circuit $C_{|x|}$, Arthur can simulate the interactive proof from the result $\mathrm{IP} = \mathrm{PSPACE}$ to verify that $x \in L$ without consulting Merlin again. It follows that $\mathrm{MA} = \mathrm{PSPACE}$. Using a padding argument as in the proof of Theorem 0.11, this result implies that $\mathrm{MAEXP} = \mathrm{EXPSPACE}$. But this is a contradiction since $\mathrm{NEXP}^{\mathrm{NP}} \subseteq \mathrm{EXPSPACE} \not\subseteq \mathrm{P}_{/\mathrm{poly}}$ by Theorem 0.11. □

This proof appears in [Aar16, Theorem 56] and partially in [AB09, Theorem 8.22]

Note that if a complexity class $\mathcal{C}$ satisfies $\mathcal{C} \not\subseteq \mathrm{P}_{/\mathrm{POLY}}$ then $\mathcal{C} \not\subseteq \mathrm{ACC}^0$ as $\mathrm{ACC}^0 \subseteq \mathrm{P}_{/\mathrm{POLY}}$. Hence Theorem 0.12 implies for example that $\mathrm{MAEXP} \not\subseteq \mathrm{ACC}^0$. Williams' result that $\mathrm{NEXP} \not\subseteq \mathrm{ACC}^0$ marks a significant improvement on these previous results. A key ingredient in his proof is the representation of ACC^0 as SYM^+ circuits.[5] A symmetric gate is one whose output depends only on the number of ones. For example, $\wedge, \vee, \mathrm{MOD}_m$ are all symmetric gates. The representation is originally due to Yao [Yao90] but was subsequently improved by Beigel and Tarui [BT91].

[5] Interestingly, in reference to SYM^+ circuits, Arora and Barak write "we wish to mention an alternative representation of ACC^0 circuits that be useful in further lower bounds" [AB09] in 2009, a couple years before Williams' proof came out that relied on exactly what they had anticipated!

The proof requires the Valiant-Vazirani lemma.

Lemma 0.13 (Valiant-Vazirani lemma)**.** *Let $\mathcal{H}$ be a pairwise independent hash function collection where $f \in \mathcal{H}$ is a function from $\{0,1\}^n$ to $\{0,1\}^k$. Let $S \subseteq \{0,1\}^n$ satisfy $2^{k-2} \leq |S| \leq 2^{k-1}$. Then*

$$\mathbb{P}_{f \in \mathcal{H}}[\exists! x \in S \text{ with } h(x) = s] \geq \frac{1}{8}$$

where s is any fixed $s \in \{0,1\}^k$.

$\exists! x$ is notation for "there exists a unique x"

Theorem 0.14 ([Yao90] and [BT91])**.** *Let $L \in \mathrm{ACC}^0$. Then L can be computed by a depth-2 circuit consisting entirely of $\vee$ gates at the first level and a single symmetric gate at second (output) level. The $\vee$ gates have polylogarithmic fan-in whereas the the symmetric gate has quasipolynomial fan-in.*[6]

[6] Note that the polylog and quasipolynomial factors are with respect to n, the size of the input.

A circuit which has the properties in Theorem 0.14 is called a SYM^+ circuit. An equivalent definition of a SYM^+ circuit on n inputs of size s is a pair (p, Ψ) where p is a multilinear polynomial $p \in \mathbb{Z}[x_1, \ldots, x_n]$ with each coefficient c satisfying $|c| \leq s^{\log^{O(1)}(s)}$ and $\deg(p) \leq \log^{O(1)}(s)$ and Ψ is a function $\Psi : \mathbb{Z} \to \{0,1\}$. This is because for a monomial $c \prod_{i \in I} x_i$ we can evaluate the product $\prod_{i \in I} x_i$ simply as $x_1 \wedge \cdots \wedge x_{|I|}$. We can then convert the ANDs into ORs using De Morgan's law (since we also have access to the negation of the variables). Then once we compute the product as $a = \prod_{i \in I} x_i$ we can add c gates of the form $a \wedge 1$ which feed into the symmetric function. Since c is bounded by $s^{\log^{O(1)}(s)}$ this adds a quasipolynomial overhead to the fan-in of the symmetric gate. Note that the sum of the monomials can be absorbed into Ψ by redefining it as

$$\Psi'(x_1, \ldots, x_k) = \Psi\left(\sum_{i=1}^{k} x_i\right)$$

which is still a symmetric function.

Before we give the proof we need one more requisite concept. In the proof of Toda's theorem, one constructs "Toda's polynomials"[7],

[7] Also called "modulus amplifying polynomials" as in [BT91].

p_t, which have the property that

$$p_t(x) \pmod{m^{2^t}} \equiv x \pmod{m}$$

if $x \pmod{m}$ is 0 or 1. For example take $g(x) = 3x^2 - 2x^3$. Let $g^k(x)$ denote the composition of g with itself k times. Then one can show via induction that

$$g^t(x) \pmod{m^{2^t}} \equiv x \pmod{m}$$

whenever $x \pmod{m} \in \{0,1\}$. Note also that $\deg(g^t) \leq 2^t$ and the coefficients are bounded by $(3 \cdot 3^2)^t$. Hence in particular the degree and the coefficients are bounded by a polynomial in 2^t.

Proof. Let C be an arbitrary circuit with size s and depth d. Assume without loss of generality that the circuit consists entirely of $\mathrm{MOD}_2, \mathrm{MOD}_3, \vee$ gates and the constant 1. The case can be extended to arbitrary MOD gates using the main ideas in this restricted case. Then a $\neg$ gate can be simulated by a MOD_2 gate with the constant 1 fed into it and hence the $\neg$ gates and the $\vee$ gate can simulate the $\wedge$ gates. We also assume that C is layered with each layer i receiving inputs from level $i-1$ and layer 1 is the input layer. Each layer consists of a single type of gate. Since C has constant depth and polynomial size we can ensure that every gate has fan-out 1 by duplicating gates at each level (thus adding a polynomial overhead to the size for each level d).

The first step is to reduce the fan-in of the $\vee$ gates to be polylogarithmic. Specifically we construct a set $\mathcal{D}$ of circuits of size at most $O(s^2)$ and depth $O(d)$ consisting of the same gate types as C such that each $\vee$ gate has fan-in $O(\log^2 s)$ and for every fixed $x \in \{0,1\}^n$ we have

$$\mathbb{P}_{D \sim \mathcal{D}}[D(x) = C(x)] > 7/8.$$

Let $\vee$ be denoted by a function $\mathrm{OR}(x_1, \ldots, x_{2^k})$ for k chosen to be the smallest integer such that the fan-in of the gate is at most 2^k (so 2^k over counts the input wires and we can fix the over counted inputs to be 0 without loss of generality). Then pick $\ell \in \{2, \ldots, k+1\}$ at random and select a random hash function h from the family $\mathcal{H}_{k,\ell} = \{f | f : \{0,1\}^k \to \{0,1\}^\ell\}$. Let $s = 0 \cdots 01 \in \{0,1\}^\ell$. Without loss of generality we can regard h as a function from $[2^k] \to \{0,1\}^\ell$. Let $S = \{i \in [2^k] : x_i = 1\} \subseteq \{0,1\}^k$. Then by Lemma 0.13 if ℓ satisfies $2^{\ell-2} \leq |S| \leq 2^{\ell-1}$ we have that

$$\mathbb{P}\left(\sum_{\substack{i=1 \\ h(i)=1}}^{2^k} x_i \pmod 2 = 1 \right) \geq \frac{1}{8}$$

since if there is a unique $i \in S$ with $h(i) = 1$ then there is one term in the sum which evaluates to 1 by the definition of S. Now if we factor in the probability that we selected the correct ℓ we get that with probability at least $1/(8k)$ the sum evaluates to 1.

Pick $t = k \log_k(s)$ such hash functions independently. We then take the OR of t sums which will give the correct answer with high probability. This new gate looks like

$$\bigvee_{j=1}^{t} \left(\sum_{\substack{i=1 \\ h_j(i)=1}}^{2^k} x_i \pmod 2 \right).$$

The probability of an error (i.e. the probability that the sums all yield 0) is bounded above by

$$\left(\frac{1}{8k}\right)^t \leq \frac{1}{8^t(s)} \leq \frac{1}{8s}.$$

We reuse the same t hash functions for all the gates. Taking the union bound over all s gates we get that the probability of an error is at most $1/8$. For each set of t hash functions we get a different circuit. So the total number of circuits in $\mathcal{D}$ is at most $|\mathcal{H}_{k,\ell}|^t$. We can assume that $s \leq 2^k \leq 2s$ since the fan-in of every OR gate is at most s. There exists pairwise independent hash families $\mathcal{H}_{k,\ell}$ of size at most $(2^k)^2 \leq (2s)^2 = O(s^2)$. We thus have

$$|\mathcal{D}| \leq |\mathcal{H}_{k,\ell}|^t \leq s^{2k \log_k s} \leq s^{2 \log_k^2 s} = s^{O(\log^2 k)}.$$

Note also that in the new circuits the fan-in is at most $t = k \log_k s = O(\log^2 s)$.

We can now rewrite the circuit so that it's final gate is a symmetric function and each OR gate has polylogarithmic fan-in and the symmetric function has quasipolynomial fan-in:

$$C(x) = \text{MAJ}(D_1(x), \ldots, D_{|\mathcal{D}|}(x)).$$

Note that since $\mathbb{P}_{D \sim \mathcal{D}}[D(x) = C(x)] > 0.5$ this representation of C is guaranteed to be correct. Note also that MAJ is a symmetric gate.

The next step is to modify the circuit in preparation for depth reduction. Specifically we transform the circuit into C' which operates internally on integers. The inputs are still 0/1 values. The output of C' is the result of a symmetric gate. Thus there is a function $\Psi : \mathbb{Z} \to \{0,1\}$ such that $C(x) = \Psi(C'(x))$. For the time being the function Ψ starts out as the majority function on the D_i (which can be viewed as a function $\mathbb{Z} \to \{0,1\}$ where $z \in \mathbb{Z}$ is the number of inputs that are 1 since majority is symmetric).

We rewrite the ORs as

$$\bigvee_{i=1}^{k} x_i = 1 - \prod_{i=1}^{k} (1 - x_i).$$

(Note this does not significantly change the fan-in). Then we replace the MOD gates using Fermat's little theorem

$$\mathrm{MOD}_p(x_1, \ldots, x_k) = \left(\sum_{i=1}^{k} x_i\right)^{p-1} \pmod{p}.$$

We can push all the multiplication operations to the bottom of the circuit, right after the inputs since multiplication binds tightly and $(x \pmod p)(x' \pmod p) = xx' \pmod p$. Also we push the additions towards the top of the gate since e.g. $(x + x') \pmod p = x \pmod p + x' \pmod p$ (i.e. we want addition to be the last thing we do). We can also assume that the circuit is still layered into multiplication gates, addition gates, and $\mathrm{MOD}_2, \mathrm{MOD}_3$ gates where multiplication gates occur only at the bottom most layer.

Now we remove the top layers by offloading some of their functionality to the function Ψ. We start with MOD_3 gates. Take a family of modulus amplifying polynomials $\{p_t\}_{t=1}^{\infty}$ where each $\deg(p_t) \leq \mathrm{poly}(t)$ and $p_t(y) \pmod{3^t} \equiv y \pmod 3$. Note that each $y_i \pmod p$ feeds into an addition gate. So the addition gate has the form

$$\sum_{i=1}^{k} y_i \pmod 3$$

where k is the fan-in of the addition gate. Then we can apply our modulus amplifying polynomials and

$$\sum_{i=1}^{k} y_i \pmod 3 = \sum_{i=1}^{k} p_t(y_i) \pmod{3^t} = \left(\sum_{i=1}^{k} p_t(y_i)\right) \pmod{3^t}$$

where t is the smallest t satisfying $3^t/2 > k$. This way $\sum_{i=1}^{k}(y_i \pmod 3) \leq 2k < 2(3^t/2) = 3^t$ which ensures that $\sum_{i=1}^{k}(p_t(y_i) \pmod{3^t}) = \left(\sum_{i=1}^{k} p_t(y_i)\right) \pmod{3^t}$. Thus we can absorb the modulo 3^t into the final function Ψ by redefining it as $\Psi'(x) = \Psi(x \pmod{3^t})$. We also add multiplication gates at the bottom layer for computing each p_t. However since the degree of each p_t is at most $\mathrm{poly}(t)$ this adds multiplication gates of fan-in at most $O(\mathrm{poly}(t))$ which is polylogarithmic in k which is in turn polylogarithmic in s. Also note that we add at most s such polynomials (one for each mod gate) and hence the size is still $O(s)$. Moreover the coefficients of these polynomials is also a polynomial in t.

We repeat this step in an analogous fashion for mod 2 gates. It follows then that the circuit has the form

$$\sum_{i=1}^{k} \prod_{j \in S_i} c_j x_i$$

for some sets $S_1, ..., S_k \subseteq \{1, \ldots, s\}$ where $|S_i| \leq \log^{O(1)}(s)$ and k and $|c_j|$ are $2^{\log^{O(1)}(s)}$. Thus the polynomial and the function Ψ form the SYM^+ circuit as desired.

□

Note that we can extend the proof to arbitrary MOD gates by first rewriting MOD_c for c composite using standard techniques[8]. Then we can handle the case where MOD_p in essentially the same fashion as the MOD_3 gates.

[8] See for example [BT91, Fact 2.2].

These are also non-uniform lower bounds. If we restrict our attention to DLOGTIME-uniform circuits we also get some interesting lower bounds.

Theorem 0.15 (Allender and Gore [AG94])**.** $\mathrm{DLOGTIME\text{-}ACC}^0 \subsetneq \mathrm{PP}$.

The proof of this theorem relies on a *uniform* version of the Beigel-Tarui transformation. In particular given a uniform ACC circuit C, it can be transformed into a uniform SYM^+ circuit in $2^{\log^{O(1)}(n)}$ time where $n = |C|$. A uniform SYM^+ circuit is one whose circuit part is DLOGTIME-uniform in the sense that given the name of a gate, the fan-in of the gate can be computed in $O(\log^{O(1)} n)$ time. Also the symmetric gate Ψ is required to be uniform which means that given the input m, the value of $\Psi(m)$ can be computed in $O(\log^{O(1)} n)$ time.

Logical characterizations of circuit classes

A key representation of circuits used to place certain functions in their respective classes leverages the descriptive complexity of circuits. For example, the proof that iterated multiplication has DLOGTIME-uniform TC^0 circuits was proved in [HAB02] by giving a FOM (FO+ a majority quantifier) formula for the problem. Motivated by such results, we give an overview of the descriptive complexity of circuits, starting with the descriptive complexity of some decision classes.

Logic preliminaries

The *syntax* of first-order logic (FOL) is defined relative to a signature[9] which consists of **constants**, **predicates**, and **functions**. Each function and predicate has an arity greater than 0. Independent of the signature is a set of variables $x_0, x_1, \ldots$. A formula relative to a given signature is defined inductively starting with predicates on terms and then allowing for negation, conjunction, quantification etc of formulas. Terms are simply variables, constants, or functions on other terms.

[9] We will be following largely Neil Immerman's presentation from [Imm12]. He uses the term *vocabulary* instead of signature.

The *semantics* of FOL are given in terms of structures on a given signature. Structures, denoted $\mathcal{A}$, consist of a nonempty set $U_\mathcal{A}$ called the universe, an interpretation of each k-ary predicate P as a k-ary relation, $P_\mathcal{A} \subseteq U_\mathcal{A} \times \cdots \times U_\mathcal{A}$, likewise an interpretation of functions, and an assignment $c_\mathcal{A} \in U_\mathcal{A}$ for each constant as well as an assignment $x_\mathcal{A} \in U_\mathcal{A}$ for all variables x. The value of $\mathcal{A}(t)$ for a term t is an element of $U_\mathcal{A}$ defined inductively by $\mathcal{A}(c) = c_\mathcal{A}$, $\mathcal{A}(x) = x_\mathcal{A}$, and $\mathcal{A}(f(t_1, \ldots, t_k)) = f_\mathcal{A}(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k))$. This allows to inductively define the satisfaction relation $\mathcal{A} \vDash \varphi$ for a FOL formula φ. We say $\mathcal{A} \vDash P(t_1, \ldots, t_k)$ iff $(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k)) \in P_\mathcal{A}$. Likewise $\mathcal{A} \vDash \varphi \wedge \psi$ iff $\mathcal{A} \vDash \varphi$ and $\mathcal{A} \vDash \psi$ and so forth for $\varphi \vee \psi$, $\neg\varphi$ and $t_1 = t_2$. For quantifiers, we say $\mathcal{A} \vDash \exists x.\varphi$ iff $\exists a \in U_\mathcal{A}$ such that $\mathcal{A}_{[x \mapsto a]} \vDash \varphi$ where $\mathcal{A}_{[x \mapsto a]}$ is exactly $\mathcal{A}$ except that $x_{\mathcal{A}_{[x \mapsto a]}} = a$. We define $\mathcal{A} \vDash \forall x.\varphi$ analogously.

For a given formula φ, we say that $\mathcal{A}$ satisfies φ iff $\mathcal{A} \vDash \varphi$.

For a signature σ we write $\mathrm{STRUCT}[\sigma]$ to denote the set of structures and we write $\mathcal{L}(\sigma)$ to denote the set of first order formulas. A boolean variable is a variable that is restricted to being either 0 or 1 (which are unequal constants in the signature). We only consider universes that are finite and consist of non-negative integers $0, 1, \ldots, n$. An important signature is the signature of binary strings $\tau_s = \langle \leq, S \rangle$ where S is a unary predicate which counts the number of 1s in a binary string (i.e. $S(i) = 1$ iff the ith bit of the string of interest is 1) and $\leq$ is a binary predicate which indicates the usual ordering on integers. A τ_s-assignment $\mathcal{A}$ represents a binary string of length $|U_\mathcal{A}|$. Throughout this section, the structures are presumed to have a binary predicate $\leq$ which is a total ordering on elements of the universe. This is necessary to capture the concept of the strings in a language recognized by some model of computation. We also require that every signature contains the constants $0, 1$ and $c_{\max}$, which is the maximum element under the $\leq$ ordering. Finally, we assume every signature contains the *numeric predicates* PLUS, TIMES, and BIT where

$$\begin{aligned} \mathrm{PLUS}(i, j, k) &\text{ holds iff } i + j = k \\ \mathrm{TIMES}(i, j, k) &\text{ holds iff } ij = k \\ \mathrm{BIT}(i, j) &\text{ holds iff bit } j \text{ in the binary expansion of } i \text{ is 1.} \end{aligned}$$

We now introduce some terminology that is useful to discuss the relationship between computation and FOL.

Definition 0.16 (query, Boolean query [Imm12])**.** A *query* is a mapping $I : \mathrm{STRUCT}[\sigma] \to \mathrm{STRUCT}[\tau]$ that is polynomially bounded meaning $|U_{I(\mathcal{A})}| \leq p(|U_\mathcal{A}|)$ for some polynomial p. A *Boolean query* is a map of the form $I_b : \mathrm{STRUCT}[\sigma] \to \{0, 1\}$.

A Boolean query can be thought of as identifying a subset of structures for which $I(\mathcal{A}) = 1$.

A common example of Boolean query is the one that identifies those structures that satisfy a given formula. In particular, given $\varphi \in \mathcal{L}(\sigma)$, the map $I_\varphi(\mathcal{A}) = 1$ iff $\mathcal{A} \vDash \varphi$ is a Boolean query. This type of Boolean query is called a *FO Boolean query*. The set of all FO Boolean queries is denoted FO.

It will be useful to consider a particular type of queries called first-order (FO) queries which are definable using FOL formulas from the signature σ and map to structures for τ. The precise definition is formulated below.

Definition 0.17 (FO queries, FO Boolean queries)**.** Fix $k > 0$. A FO query, $I : \text{STRUCT}[\sigma] \to \text{STRUCT}[\tau]$ is a query defined by formulas $\varphi_0, \varphi_1, \ldots, \varphi_r, \psi_1, \ldots, \psi_s$ from $\mathcal{L}(\sigma)$ where $\varphi(x_1, \ldots, x_k)$ contains the free variables $x_1, \ldots, x_k$, $\varphi_i(x_1, \ldots, x_m)$ contains $m = kn$ free variables where the ith predicate of σ is of arity n, and $\psi_i(x_1, \ldots, x_k)$ contains the free variables $x_1, \ldots, x_k$. Let $\mathcal{A} \in \text{STRUCT}[\sigma]$. The universe $U_{I(\mathcal{A})}$ is given by

For a structure $\mathcal{A}$ and a formula $\varphi(x)$, we write $\varphi(a)$ for $a \in U_\mathcal{A}$ to denote the formula resulting from substituting a for x in φ. We extend this naturally to $\varphi(x_1, \ldots, x_k)$ and $\varphi(a_1, \ldots, a_k)$.

$$U_{I(\mathcal{A})} = \{(b_1, \ldots, b_k) \in U_\mathcal{A}^k | \mathcal{A} \vDash \varphi_0(b_1, \ldots, b_k)\}.$$

We write $v \in U_{I(\mathcal{A})}$ for a tuple in v and simply $\varphi_0(v)$ to denote $\varphi_0((v)_1, \ldots, (v)_k)$ where $(v)_i$ is the ith entry of v. Let P be the ith predicate of σ whose arity is n, and define

$$P_{I(\mathcal{A})} = \{(v_1, \ldots, v_n) \in U_{I(\mathcal{A})}^n | \mathcal{A} \vDash \varphi_i(v_1, \ldots, v_n)\}.$$

The jth constant symbol of σ is defined by

$$c_{I(\mathcal{A})} = v \in U_{I(\mathcal{A})} \text{ such that } \mathcal{A} \vDash \psi_j(v).$$

Note that this requires there to be a *unique* $v \in U_{I(\mathcal{A})}$ such that $\mathcal{A} \vDash \psi_j(v)$. This then defines the k-ary FO query I.

A Turing machine (TM) that outputs a string to a write-only tape for a given input string can be viewed as a query of the form $I : \text{STRUCT}[\tau_s] \to \text{STRUCT}[\tau_s]$. To relate languages carved out by TMs to structures of signatures, we fix a scheme for encoding arbitrary signatures as binary strings.

Definition 0.18 (bin_t)**.** Let τ be an arbitrary structure. We define a FO query

$$\text{bin}_\tau : \text{STRUCT}[\tau] \to \text{STRUCT}[\tau_s]$$

where $\tau_s = \langle S \rangle$ is the signature of binary strings. Fix some structure $\mathcal{A}$ of τ where $c_{\max} = n - 1$ in $\mathcal{A}$. Let $P_\mathcal{A}$ be correspond to a predicate of τ with arity j. Then $\text{bin}(P_\mathcal{A})$ is the binary string of length n^j where 1 in position i indicates that the ith element of $U_\mathcal{A}^j$ is in $P_\mathcal{A}^j$. Likewise for each constant $c_\mathcal{A}$ we write $\text{bin}(c_\mathcal{A})$ for the binary string

representing $c_{\mathcal{A}}$ which has length $\log n$ (since $c_{\mathcal{A}}$ is simply an integer in $\{0, \ldots, n-1\}$). We then define

$$\text{bin}_\tau(\mathcal{A}) = \prod_{X \in \{P, c : P, c \in \mathcal{A}\}} \text{bin}(X)$$

where $c \in \mathcal{A}$ denotes a constant in $\mathcal{A}$ and $P \in \mathcal{A}$ denotes a predicate in $\mathcal{A}$ and the product is string concatenation. Note that this defines $\text{bin}_\tau(\mathcal{A})$ as a single binary string, but such a string completely determines a structure of τ_s. Note also that $|\mathcal{U}_{\text{bin}_\tau(\mathcal{A})}|$ corresponds to the length of the string $\prod_{X \in \{P, c : P, c \in \mathcal{A}\}} \text{bin}(X)$ which is $O(n^j)$ where j is the max arity of any relation in τ.

It can be checked and confirmed that $\text{bin}_\tau(A)$ is in fact a FO query and that it's inverse is one as well. Using this encoding query, we can give a notion for the computability of an arbitrary query. Note that given a structure $\mathcal{A}$, the signature can be recovered, so it is safe to write $\text{bin}(\mathcal{A})$ instead of $\text{bin}_\tau(\mathcal{A})$.

Definition 0.19 (Computability of a query)**.** Let $I : \text{STRUCT}[\sigma] \to \text{STRUCT}[\tau]$ be a query. Let M be a TM such that

$$M(\text{bin}(\mathcal{A})) = \text{bin}(I(\mathcal{A}))$$

for all $\mathcal{A} \in \text{STRUCT}[\sigma]$. Then we say M computes I.

This definition makes sense since given such a TM M if we want to know $I(\mathcal{A})$ we can encode $\mathcal{A}$ as a binary string and evaluate $M(\text{bin}(\mathcal{A}))$, then decode the output using bin^{-1} to get $I(\mathcal{A})$. Similarly we say that M computes a boolean query $I : \text{STRUCT}[\sigma] \to \{0, 1\}$ if $M(\text{bin}(\mathcal{A})) = I(\mathcal{A})$ for all $\mathcal{A} \in \text{STRUCT}[\sigma]$.

A language L in a complexity class can thus be identified with a boolean query where $I(x) = 1$ iff $x \in L$. Fix some complexity class $\mathcal{C}$, then $Q(\mathcal{C})$ denotes the set of all queries computable in $\mathcal{C}$. In particular, we require that a query is computable in $\mathcal{C}$ iff each bit in $\text{bin}(I(\mathcal{A}))$ is computable in $\mathcal{C}$.

Definition 0.20 ($Q(\mathcal{C})$)**.** Let $I : \text{STRUCT}[\sigma] \to \text{STRUCT}[\tau]$ be a query. We say I is computable in $\mathcal{C}$ iff the language

$$\{(\mathcal{A}, i, a) : \text{ the } i\text{th bit of } \text{bin}(I(\mathcal{A})) \text{ is } a\}$$

for some fixed i and $a \in \{0, 1\}$ is in $\mathcal{C}$. The class $Q(\mathcal{C})$ includes all the Boolean queries in $\mathcal{C}$ and all the I's such that I is computable in $\mathcal{C}$.

Circuits and reductions in FOL

We are now in a position to prove our first theorem.

Theorem 0.21 ([Imm12, Theorem 3.1]). *FO* $\subseteq$ L.

Proof. Let σ be a signature and let $\varphi \in \mathcal{L}(\sigma)$ determine a first-order Boolean query $I_\varphi : \text{STRUCT}[\sigma] \to \{0,1\}$. We will construct a LogSpace TM M such that $\mathcal{A} \vDash \varphi$ iff $M(\text{bin}(\mathcal{A})) = 1$. Write φ in prenex normal form[10] as $\varphi \equiv Q_1.x_1 \dots Q_k.x_k\psi(x)$ for $Q_i \in \{\forall, \exists\}$. We inductively build M on k, the number of quantifiers. If $k = 0$, then we need to compute ψ which is a Boolean combination of atomic formulas (formulas consisting of predicates and numeric relations). Note that the arity of any predicate in $\mathcal{A}$ is bounded above by some constant λ (since this depends only on σ). Thus in order to determine if $\mathcal{A} \vDash P_{\mathcal{A}}(t_1, \dots, t_j)$, M needs to store at most $\lambda \log n$ bits since each term $t_1, \dots, t_k$ (which are constants, as we disallow function symbols) require $\log n$ bits store. It can then lookup the appropriate bit of its input, $\text{bin}(\mathcal{A})$, to determine whether $\mathcal{A} \vDash P_{\mathcal{A}}(t_1, \dots, t_j)$. It can then erase it's tape and move onto the next proposition, taking note of the Boolean connectives between them.

[10] A formula is in prenex normal form if all the quantifiers are pushed out in front of the formula and are followed by a quantifier-free formula.

Now inductively assume that all first-order queries with $k-1$ quantifiers are computable by a LogSpace machine. Write $Q_1\phi(x_1)$ where the formula $\phi(x_1)$ contains $k-1$ quantifies variables, so that if we substitute a constant for x_1 we can compute ϕ is LogSpace. Then to compute $\forall.x_1\phi(x_1)$ we simply cycle through all possible constants $c \in U_{\mathcal{A}}$ and run the LogSpace machine we get from the inductive hypothesis on $\phi(c)$ to check if $\mathcal{A}_{[x_1 \mapsto c]} \vDash \phi(x_1)$. If this holds for every constant c then we accept. We can likewise compute $\exists.x_1\phi(x_1)$. Note that this procedure only requires storing an extra $\log n$ bits for the constant c. □

Let $\text{AC}[t(n)]$ denote AC circuits of depth $O(t(n))$ so that $\text{AC}^i = \text{AC}[\log^i(n)]$. The signature of a circuit is $\tau_c = \langle E, G_\wedge, G_\vee, G_\neg, \iota, r\rangle$ where an internal node w is an AND gate iff $G_\wedge(w)$ holds, and likewise for $G_\vee, G_\neg$. The input relation ι indicates whether input x_i (which is a leaf in the circuit) is set to 1 or not. It is often given separately from the circuit C (which is a structure of τ_c and can be written $C = (V, E, G_\wedge, G_\vee, G_\neg, \iota, r)$). Also r is the constant representing the output node. We assume that any formula over τ_c includes parts that enforce basic structural properties of C, such as the fact that input gates come first with respect to the ordering $\leq$ on the universe. We can generalize this signature to allow for threshold (TC) circuits by adding a predicate $G_t(g, v)$ for the threshold gate, indicating that g is a threshold gate with threshold value v. Given $\mathcal{A} \in \text{STRUCT}[\tau]$, we can evaluate a circuit C on $\text{bin}(\mathcal{A})$ in the usual way. A circuit accepts $\mathcal{A}$ iff $C(\text{bin}(\mathcal{A})) = 1$. We assume a uniformity condition on circuits. In particular given a circuit C_n, there is a FO query $I : \text{STRUCT}[\tau_s] \to \text{STRUCT}[\tau_c]$ with $C_n = I(0^n)$. If $I \in$ FO

then the family $\mathcal{C} = \{C_n\}_{n=1}^{\infty}$ is said to be first-order uniform.

A *block of quantifiers* is a sequence $(Q_1 x_1 M_1) \dots (Q_s x_s M_s)$ where

$$\begin{aligned} Q_i &\in \{\forall, \exists\} \\ M_i &\text{ is quantifier-free} \\ (\forall x.M)\psi &\equiv \forall x.(M \to \psi) \\ (\exists x.M)\psi &\equiv \exists x.(M \wedge \psi). \end{aligned}$$

Let Q be a block of quantifiers. Then $[Q]^r$ is Q iterated r times, i.e.

$$[Q]^r = \underbrace{Q \cdots Q}_{r \text{ times}}.$$

Then we define FO$[t(n)]$ to be the set of FO Boolean queries defined by quantifier blocks iterated $O(t(n))$ times.

Definition 0.22. A set $S \subseteq \text{STRUCT}[\tau]$ is in FO$[t(n)]$ iff there are k quantifier free formulas M_i from $\mathcal{L}(\tau)$, a set $c_1, ..., c_j$ of constants and a quantifier block $Q = [(Q_1.x_1 M_1) \dots (Q_k x_k M_k)]$ such that for all $\mathcal{A} \in \text{STRUCT}[\tau]$ we have

$$\mathcal{A} \in S \Leftrightarrow \mathcal{A} \vDash \left([Q]^{t(|U_{\mathcal{A}}|)} M_0\right)(c_1/x_1) \cdots (c_j/x_j).$$

Recall that c/x is notation for the substitution of c into x. We stipulate the substitution of constants because the quantifier block Q may contain some free variables the must be substituted for to build an FOL formula.

Then we have

Theorem 0.23. *FO$[t(n)]$ = AC$[t(n)]$.*

Proof. We start by showing that FO$[t(n)] \subseteq$ AC$[t(n)]$. An FO$[t(n)]$ Boolean query is given by a quantifier block $Q = [(Q_1 x_1.M_1) \cdots (Q_k x_k M_k)]$ and an initial formula M_0 and a tuple of constants $\bar{c}$. The goal will be to construct a first-order query I such that $C_n = I(0^n)$, where C_n is an AC^0 circuit satisfying

$$\mathcal{A} \vDash (\text{Q}^{t(|U_{\mathcal{A}}|)M_0})(\bar{c}/\bar{x}) \Leftrightarrow C_{|U_{\mathcal{A}}|}(\mathcal{A}) = 1$$

for all $\mathcal{A} \in \text{STRUCT}[\tau]$. Define $\varphi^r = (Q_i x_i.M_i) \cdots (Q_k x_k.M_k)[Q]^q M_0$ for $r = k(q+1) + 1 - i$ and $0 \leq r \leq t(n)k$. Intuitively φ^r is the "innermost" r quantifiers of $[Q]$. Note that $r \equiv 1 - i \pmod{k}$. We build C in the following manner. At the input level, we start by hardwiring the evaluation of $M_1, \dots, M_k$ as k is a constant and M_i is fixed. In particular we have $O(1)$-size subcircuits $\langle M_i, b_1, \dots, b_k \rangle$ such that

$$\langle M_i, b_1, \dots, b_k \rangle(\text{bin}(\mathcal{A})) = 1 \Leftrightarrow \mathcal{A} \vDash M_i(b_1, \dots, b_k).$$

Using these subcircuits we show how to evaluate whether $\mathcal{A} \vDash \varphi^r(b_1, \dots, b_k)$. We define a subcircuit $\langle 2r, b_1, \dots, \hat{b}_i, \dots, b_k \rangle$ such that

$$\langle 2r, b_1, \dots, \hat{b}_i, \dots, b_k \rangle(\text{bin}(\mathcal{A})) = 1 \Leftrightarrow \mathcal{A} \vDash \varphi^r(b_1, \dots, b_k).$$

We build such subcircuits inductively. If $Q_i = \forall$, we let $\langle 2r, b_1, \ldots, \hat{b}_i, \ldots, b_k \rangle$ be an AND gate. Otherwise, it is an OR gate. Before defining the wiring of this gate, we introduce a new subcircuit $\langle 2r-1, b_1, \ldots, \hat{b}_i, \ldots, b_k \rangle$ (note here $2r-1$ is odd, distinguishing it from the other subcircuits). This gate is exactly

$$\langle 2r-1, b_1, \ldots, \hat{b}_i, \ldots, b_k \rangle =$$

$$\text{AND}(\langle M_i, b_1, \ldots, b_k \rangle, \langle 2r-2, b_1, \ldots, b_i, \hat{b}_{i+1}, \ldots, b_k \rangle).$$

Then the gate $\langle 2r, b_1, \ldots, \hat{b}_i, \ldots, b_k \rangle$ is defined as either an AND or an OR (depending on Q_i) whose inputs are $\langle 2r-1, b_1, \ldots, b_i, \hat{b}_{i+1}, \ldots, b_k \rangle$ where b_i ranges over $U_{\mathcal{A}}$. It follows then that

$$\mathcal{A} \vDash (\mathrm{Q}^{t(|U_{\mathcal{A}}|)M_0})(\overline{c}/\overline{x}) \Leftrightarrow C_{|U_{\mathcal{A}}|}(\mathcal{A}) = 1$$

and $|C|$ is polynomial in n since we construct $O(t(n)k) = O(t(n))$ gates.

For the other direction we show that $\mathrm{AC}[t(n)] \subseteq \mathrm{IND}[t(n)] \subseteq \mathrm{FO}[t(n)]$ where $\mathrm{IND}[t(n)]$ ([Imm12, Defn. 4.16]) is the sublanguage of $\mathrm{FO}(\mathrm{LFP})$ in which only fixed points of first-order formulas φ for which $|\varphi|$ is $O(f(n))$ are included. The proof of $\mathrm{IND}[t(n)] \subseteq \mathrm{FO}[t(n)]$ proceeds by showing that the effect of the least-fixed-point operator is equivalent to that of a quantifier block followed by a fixed formula. In particular, one can find a quantifier block Q such that $\mathcal{A} \vDash \mathrm{LFP}\varphi \Leftrightarrow [Q]^t \mathbf{false}$ [Imm12, Lemma 4.20]. The result then follows from the definition of $\mathrm{FO}[t(n)]$.

The proof that $\mathrm{AC}[t(n)] \subseteq \mathrm{IND}[t(n)]$ proceeds by giving a direct construction of an inductive formula for a circuit C. One constructs the formula $V(x, b)$ which represents gate x with Boolean value b. Then $V(r, 1)$ captures the acceptance condition of C in $\mathrm{IND}[t(n)]$ where r is the output node of C. See [Imm12, Theorem 5.22] for the explicit construction of $V(x, b)$.

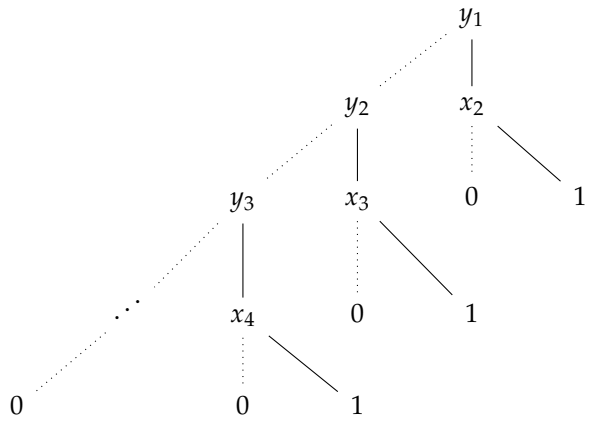
□

Barrington's Theorem

A bounded width branching program (BWBP) is a restricted model of computation which initially looks like it lies somewhere between a deterministic automaton and a Boolean circuit. Intuitively, a BWBP can be thought of as a type of finite state machine where each state is allowed to inspect c bits of the input and at most c bits from the previous state (for some universal constant c) and then decides a new set of at most c bits to pass to the next state. The final state then outputs a single bit indicating whether to accept or reject the input.

Formally a BWBP is a restricted type of decision diagram. A decision diagram (DD) is a directed acyclic graph whose internal nodes

or *branches* are labeled with x_i, one of the input variables, and whose leafs are labeled with 0 or 1. Each internal node has two outgoing edges called the high branch and the low branch. An input x induces a path through the tree by traversing the high branch of every internal node x_i if $x_i = 1$ and traversing the low branch otherwise. The input is said to be accepted by the DD if the path induced by x ends in a 1. Thus a DD on $x_1, \ldots, x_n$ naturally computes a Boolean function $\{0,1\}^n \to \{0,1\}$. For example consider the DD below



which computes a function $f(y_1, x_1, \ldots, y_n, x_n)$. Note that we use the convention that dotted branches are low branches and solid branches are high branches. In this case then, $f(y_1, x_1, \ldots, y_n, x_n) = 1$ iff $y_i = x_{i+1} = 1$ for some $1 \leq i \leq n-1$. The depth and size of a DD are measured in the usual manner.

A BWBP is a special type of DD where the nodes can be organized into levels such that outgoing edges of nodes in level i can only connect to nodes in level $i+1$ and furthermore, the maximum number of nodes in any particular layer (the *width* of a BWBP) is bounded by a universal constant c (the constant must hold for all BWBPs in a family computing some language, regardless of the input size).[11] Note that any family $\{f_n\}_{n=1}^{\infty}$ of Boolean functions can be computed by width 2 BWBPs by writing the function in DNF form. Thus, we restrict our attention to BWBPs of *polynomial size*.

[11] In the analogy at the beginning of the section, each state is a level and the c bits is the width of the BWBP. Note we can restrict each level to compute the same bit i in the input at a cost of doubling the width and adding a polynomial amount of nodes to the size.

In particular let BWBP denote the set of all languages computable by polynomial size BWBPs. When this model was initially developed in the it was known that PARITY $\in$ BWBP and in particular that BWBP $\subseteq \mathrm{NC}^1$. But it was unclear until 1986 whether e.g. MAJORITY is in BWBP or not. The following theorem settled the question.

Theorem 0.24 ([Bar89]). BWBP $= \mathrm{NC}^1$.

The proof relies on the following representation of a BWBP. A BWBP with width w is a sequence of instructions of the form $\langle i, f, g \rangle$ where i is an index of an input variable, and $f, g : [w] \to [w]$ are

functions. The number of instructions is the length of the program. The execution of the instruction involves checking if $x_i = 1$ in which case f is returned. Otherwise, g is returned. The result of the computation is the function $h_k \circ \cdots \circ h_1$ where there are k instructions and h_j is the function resulting from the execution of the jth instruction. Note then the output is a function $[w] \to [w]$. A family $\{B_n\}_{n=1}^{\infty}$ of BWBPs of this form compute a language L if for each n there is a set $F_n \subseteq \{f | f : [w] \to [w]\}$ such that for all $x \in \{0,1\}^n$ we have $B_n(x) \in F_n \Leftrightarrow x \in L$. Note that $|F_n| \leq w^w$ is bounded by a constant.

Barrington considers a restricted type of BWBP where the functions f, g are all permutations of $[w]$ so that $f, g \in S_w$, the symmetric group on w elements. A w permutation BWBP $\{B_n\}$ *c-cycle recognizes* a language L if there is a fixed $\sigma \in S_w$, σ a c-cycle such that $B_n(x) = \sigma$ iff $x \in L$ and $B_n(x) = 1$ otherwise (note here $1 \in S_w$ is the identity permutation).

The technical statement of Barrington's theorem is then the following

Theorem 0.25 ([Bar89])**.** *Let L be recognized by depth-d NC^1 Boolean circuit. Then L is 5-cycle recognized by B with the length of B bounded by 4^d.*

The theorem relies on three lemmas.

Lemma 0.26. *If B c-cycle recognizes L with output $\sigma \in S_w$, and $\tau \in S_w$ is any other c-cycle, then there is a permutation BWBP of the same length as B that also c-cycle recognizes L and outputs τ.*

Proof. In S_n cycle type completely determines conjugacy classes (see for example [DF04, Chapter 4, Proposition 11]). Thus since τ and σ have the same cycle type, they are conjugate. In particular there is some $\theta \in S_w$ with $\tau = \theta\sigma\theta^{-1}$. So to get the new BWBP we can just change every function in an instruction $f \in S_w$ to $\theta f \theta^{-1}$ and the product (which corresponds to function composition) will then be $\theta\sigma\theta^{-1}$. □

Lemma 0.27. *If B c-cycle recognizes L and B is length ℓ, then there is a length ℓ permutation BWBP that recognizes $\overline{L}$*

Proof. Let σ be the output of B. Let $\langle i, \mu, \nu \rangle$ be the last instruction of B. Define B' to be B except that the last instruction is $\langle i, \mu\sigma^{-1}, \nu\sigma^{-1} \rangle$. Then $B'(x) = \sigma^{-1}$ iff $x \notin L$ and $B'(x) = 1$ iff $x \in L$. □

Note that σ^{-1} is a c-cycle since it is obtained by simply reversing the order of the elements in the cycle.

We also use the fact that there are 5 cycles whose commuter is a five cycle as evidenced by

$$(12345)(13542)(12345)^{-1}(13542)^{-1} = (13254).$$

We can now prove the theorem. For the proof, it is sufficient to let $w = 5$ and consider S_5.

Proof of Theorem 0.25. Let C be a depth d NC^1 circuit. Our proof is by induction on d. If $d = 0$ then C consists of a single output gate x_i. Then $B = \langle i, (12345), 1\rangle$ computes C. Assume the output gate is an AND gate. If it's an OR gate then $L = L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$ and we can compute the complements using Lemma 0.27. Thus, WLOG consider $L = L_1 \cap L_2$. Inductively, L_1 and L_2 are 5-cycle recognized by B_1 and B_2 of length at most 4^{d-1}. Using Lemma 0.26 we can pick the 5-cycles of B_1 and B_2 to be σ_1 and σ_2 such that $[\sigma_1, \sigma_2] \in S_5$ (which we showed was possible above). Let B_1' and B_2' yield σ_1^{-1} and σ_2^{-1} respectively. Then let $B = B_1 B_2 B_1' B_2'$ be the concatenation of the instructions. Note that if $x \notin L_1 \cap L_2$ this yields 1 and otherwise we get $[\sigma_1, \sigma_2]$. The length of this new program is at most $(4)4^{d-1} = 4^d$. □

We write $[\sigma, \tau]$ for the commuter of σ and τ, i.e. $[\sigma, \tau] = \sigma\tau\sigma^{-1}\tau^{-1}$.

Proof of Theorem 0.24. We start by showing the direction $\mathrm{NC}^1 \subseteq$ BWBP. Let $L \in \mathrm{NC}^1$ and let $\{C_i\}$ be a family of depth-$O(\log n)$ and size $O(\mathrm{poly}(n))$ circuits computing L. For each C_i we find a corresponding, length $4^{(O(\log n))} = O(n)$, BWBP B_i as given by Theorem 0.25. Thus B_i has width 5 and polynomial size.

Now for the converse suppose L is a language recognized by a width w branching program B of length ℓ. We exhibit a depth $O(\log \ell)$ circuit that computes L. Note that a function $f : [w] \to [w]$ can be written as an w^2 matrix A such that $A_{ij} = 1$ iff $f(i) = j$. Note then each row consists of exactly one 1 and the rest 0s. The product of two such matrices thus gives the function composition. If A is the matrix for f and B is the matrix for g then BA is the matrix for $f \circ g$ since $(BA)_{ij} = 1$ iff $B_{ik} = 1$ and $A_{kj} = 1$ for some k indicating that $g(i) = k$ and $f(k) = j$ and hence $f(g(i)) = j$. We hard-code every product of such a matrix (which requires hard-coding w^4 matrices which is constant). Then for a given input we can compute a product of ℓ such matrices using a binary tree of depth $\log \ell$. For example to compute $ABCD$ we would compute AB and CD in parallel and depth $O(1)$ since we've already hard-coded all 2-matrix products and then we get matrices $E = AB$ and $E' = CD$ and we compute EE' which requires one additional layer of depth. Once we get the product we can then directly check if it is in a predetermined set of matrices that represent the accepting functions.[12] □

[12] The accepting set of functions from $[w] \to [w]$ is constant and has at most w^w elements. So we can also hard-wire this set with constant overhead.

Remark 0.28. *The only property of S_n we exploited was that there are elements σ_1 and σ_2 such that $\sigma_1, \sigma_2, [\sigma_1, \sigma_2]$ are all conjugate to each other. In other words, $\sigma_1, \sigma_2, [\sigma_1, \sigma_2]$ are all in the same conjugacy class.*[13]

Note that the property mentioned in the above remark is true only of unsolvable groups.

[13] For a group G the conjugacy class of $a \in G$ is $C_G(a) = \{b : a = gbg^{-1}$ for some $g \in G\}$. The conjugacy classes for an equivalence relation $\sim$ where $a \sim b$ iff $C_G(a) = C_G(b)$.

Proposition 0.29. *Suppose G is a group and there are elements $a, b \in G$ such that $b, [a,b] \in C_G(a)$ and $[a,b] \neq 1$ then G is unsolvable.*

Proof. We first note every normal subgroup of G is the union of conjugacy classes. Namely for $N \trianglelefteq G$ we have

$$N = \bigcup_{n \in N} C_G(n)$$

since given $n \in N$ we have $gng^{-1} \in N$ for every g and so $C_G(n) \subseteq N$. Thus the derived subgroup[14] of G, $[G,G] \trianglelefteq G$, is the union of conjugacy classes. Since $[a,b] \in [G,G]$ it follows that $C_G([a,b]) \in [G,G]$ and hence $a, b \in [G,G]$ as well. In particular $a, b, [a,b] \in G^{(2)} = [[G,G],[G,G]]$ and likewise $a, b, [a,b]$ is in every subgroup in the derived series. So since $[a,b] \neq 1$ the derived series cannot terminate and G is unsolvable. □

[14] The subgroup $G^{(i)}$ is defined inductively by $G^{(0)} = G$ and $G^{(i+1)} = [G^{(i)}, G^{(i)}]$. The subnormal series $G^{(0)} \trianglerighteq G^{(1)} \trianglerighteq \cdots$ is called the derived series. The subgroup $G^{(1)}$ is the derived subgroup.

Note also that we have a sort of converse theorem: if G is any finite unsolvable group then the languages which have G-BPs are exactly the languages in NC^1. Here a G-BP is analogous to the BWBPs which 5-cycle recognize a language except instead of picking elements of S_5 in the instruction, we pick elements of G, an arbitrary group. In this setting, the output is simply a product of elements of G.

Theorem 0.30. *Let G be a finite unsolvable group. Let $|G| = g$. A depth-d circuit with fan-in 2 can be computed by a length $(4g)^d$ G-BP.*

Proof. The proof is virtually the same as in the case of the proof of Theorem 0.25. Since G is unsolvable, its derived series must terminate. In particular there is some nontrivial subgroup of G, H such that $[H,H] = H$. We consider only branching programs that output an element of H. Let $g = |G| \geq |H|$ and fix $a \in H$. Let C be a depth d circuit with fan-in 2. We give a length $(4g)^d$ H-BP that outputs a if the circuit C accepts its input and e otherwise.

We note that as in Lemma 0.26 if we are given an H-BP B that outputs $\sigma \in H$ then we can find an H-BP B' that is the same length as B and outputs $\tau \in H \setminus \{e\}$ for any chosen τ. We simply modify the last instruction $\langle i, \mu, \nu \rangle$ to be $\langle i, \mu\sigma\tau^{-1}, \nu \rangle$.

Now we proceed inductively in the same manner as in the proof of Theorem 0.25. The steps are the same up to the inductive step. Suppose we have inductively B_1 and B_2 each of length at most $(4g)^{d-1}$ which compute L_1 and L_2 respectively. We want to find an H-BP to compute $L_1 \cap L_2$ which outputs a. Write $a = [\gamma_1, \lambda_1] \cdots [\gamma_g, \lambda_g]$ using the fact that $[H,H] = H$. Then we form the H-BP Γ_i which outputs γ_i by modifying B_1 and likewise we form $\hat{\Gamma}_i$ which computes γ_i^{-1}.

Similarly we form Λ_i by modifying B_2 to output λ_i and likewise form $\hat{\Lambda}_i$ by modifying B_2 to output λ_i^{-1}. Then the new H-BP we form is

$$\Gamma_1\Lambda_1\hat{\Gamma}_1\hat{\Lambda}_1\cdots\Gamma_g\Lambda_g\hat{\Gamma}_g\hat{\Lambda}_g$$

which outputs a if the input is in $L_1 \cap L_2$ and e otherwise. The length of this program is at most $4g(4g)^{d-1} = (4g)^d$. □

References

[Aar16] Scott Aaronson. $P \stackrel{?}{=} NP$. In *Open problems in mathematics*, pages 1–122. Springer, 2016.

[AB87] Noga Alon and Ravi Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1):1–22, 1987.

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[AG94] Eric Allender and Vivek Gore. A uniform circuit lower bound for the permanent. *SIAM Journal on Computing*, 23(5):1026–1049, 1994.

[Ais13] James Aisenberg. The razborov monotone circuit lower bound. https://www.math.ucsd.edu/~sbuss/CourseWeb/Math262A_2013F/Scribe07.pdf, October 2013. [Online; accessed 1-September-2019].

[Ajt83] Miklós Ajtai. Σ_1^1-formulae on finite structures. *Annals of pure and applied logic*, 24(1):1–48, 1983.

[ALWZ19] Ryan Alweiss, Shachar Lovett, Kewen Wu, and Jiapeng Zhang. Improved bounds for the sunflower lemma. *arXiv preprint arXiv:1908.08483*, 2019.

[AM05] Kazuyuki Amano and Akira Maruoka. A superpolynomial lower bound for a circuit computing the clique function with at most (1/6) log log n negation gates. *SIAM Journal on Computing*, 35(1):201–216, 2005.

[Bar89] David A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

[BFT98] Harry Buhrman, Lance Fortnow, and Thomas Thierauf. Nonrelativizing separations. In *Proceedings. Thirteenth Annual IEEE Conference on Computational Complexity (Formerly: Structure in Complexity Theory Conference)(Cat. No. 98CB36247)*, pages 8–12. IEEE, 1998.

[BT91] Richard Beigel and Jun Tarui. On ACC (circuit complexity). In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 783–792. IEEE, 1991.

[DF04] David Steven Dummit and Richard M Foote. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.

[ER60] Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.

[FSS84] Merrick Furst, James Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical systems theory*, 17(1):13–27, 1984.

[HAB02] William Hesse, Eric Allender, and David A Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002.

[Imm12] Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 2012.

[Juk12] Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012.

[Kan82] Ravi Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and control*, 55(1-3):40–56, 1982.

[Raz85] Alexander A Razborov. Lower bounds for the monotone complexity of some boolean functions. In *Soviet Math. Dokl.*, volume 31, pages 354–357, 1985.

[Sha49] Claude E Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, 1949.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 77–82. ACM, 1987.

[Wil11] Ryan Williams. Non-uniform ACC circuit lower bounds. In *2011 26th Annual IEEE Conference on Computational Complexity*, pages 115–125. IEEE, 2011.

[Yao90] Andrew Chi-Chih Yao. On ACC and threshold circuits. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 619–627. IEEE, 1990.