Circuits Notes

Alex Creiner

November 3, 2023

1 Basic Definitions and Containments

Definition 1.1. An **n-input Boolean circuit** is a directed acyclic graph with n **sources** (vertices with no incoming edges) and one **sink** (vertex with no outgoing edges). Non-source vertices are called **gates**, and each gate has an associated type. There are three types, denoted $\wedge$, $\vee$, and $\neg$. The **fan-in** or **width** of a gate is the number of incoming edges, and the number of outgoing edges is called the **fan-out**. Not gates (those whose type is $\neg$) are required to have fan-in and fan-out 1. The **size** of a circuit C, denoted $|C|$, is the number of vertices in the graph. The **depth** of a circuit is the shortest distance from a sink to one of it's sources.

We next define the **output** of an n-input circuit C on an **input** $x \in \{0,1\}^n$, which we denote $C(x)$. To do this, we define recursively the function $val(v)$ where v is a vertex. We assume an ordering to the sources. For a source vertex v, let $val(v)$ be the corresponding bit-value of x. If v is a not gate, let $val(v)$ be $\overline{val}(v')$, where v' is the single vertex whose outgoing edge enters v. If v is an and gate, let $val(v) = \bigwedge val(v_i)$ i.e. the the conjunction over all $val(v')$s with v' leading to v. Similar for or gates. Finally let $C(x) = val(v)$ where v is the sink.

We say that a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is **computed** by the n-input Boolean circuit C if $C(x) = f(x)$ for all n-bit strings x. A **circuit-family** is a collection of circuits $\mathcal{C} = \{C_n\}_{n\in\omega}$ such that for each n C_n is an n input circuit. We say that a language L is decided by a circuit family $\mathcal{C}$ if for any string x of length n, $x \in L \iff C_n(x) = 1$.

Define the class $\boldsymbol{AC}^i$ to be the set of languages L such that there exists a circuit family $\mathcal{C}$ and an integer $k \in \omega$ such that $\mathcal{C}$ decides L, $|C_n| = O(n^k)$ for all n, and C_n has depth $O(\log^i(n))$ for all n. Note in particular that $\boldsymbol{AC}^0$ is the class of languages decidable by polynomial size circuits of constant depth. Define the class $\boldsymbol{NC}^i$ in the exact same way, except with the additional requirement that the $\wedge$ and $\vee$ gates are limited to having fan-in exactly 2. (So $\boldsymbol{AC}$ circuits are allowed to have unbounded fan-in). Finally define $\boldsymbol{AC}$ and $\boldsymbol{NC}$ to be the union over all integers of their respective hierarchies.

Fact 1.1. *If C is an n-input Boolean circuit of depth d, then there exists a depth $d+1$ circuit C' computing the same Boolean function in which all $\neg$ gates are distance exactly one from the sources, and there are at most n of them.*

Proof. The idea is to repeatedly using DeMorgan's law in order to 'push down' not gates below any and/or gates they might be above. For instance, if there is a not gate above an and gate with fan-in k, then we can replace the and gate with an OR gate, and attach k many NOT gates in between whatever vertices connected to the former AND gate and the new OR gate. We can do this repeatedly until we've pushed all not gates down to the bottom of the circuit. At that point we will see many sources fanning out to simply be negated and feed into a single gate, and these redundancies can be consolidated. The result is a circuit of at most one higher depth than before, with at most n many NOT gates which sources fan-out, feed into, and then fan-out again where they are needed. This also shows that, assuming unbounded fan-out, any n-input constant depth circuit requires only at most n not gates. $\square$

Fact 1.2. *If C is an n-input Boolean circuit of depth d without any NOT gates (simply for simplicity, see above), then there is a depth d' (I think less than $2d$?) circuit computing the same Boolean function in which all gates distance i from a source of the same type, all gates distance $i+1$ from the source are of the opposite type, and so forth, i.e. we can assume without loss of generality a circuit of alternating layers of AND and OR. Furthermore the new circuit has size less than or equal to $2S$ (S the size of C).*

Proof. We can do this by abusing the unbounded fan-in, and making repeated use in particular of our ability to pad a circuit with trivial 'buffer' gates, AND and OR gates of fan-in 1. Depth 1 circuits only have a single gate (excluding a possible layer of NOTs), so this case is trivial. Consider the depth 2 case, and particular the simple case of a single AND gate (acting as the sink) of fan-in 2, connecting to an OR gate and an AND gate, each of which also has fan-in 2, connecting to sources x_1, x_2, x_3, x_4 (the first two connect to the OR, the second to the AND). To construct our alternating equivalent circuit, we first shove an OR gate in between the sink and the AND gate which was a distance 1 from it. This is as we mentioned simply a trivial buffer - it has fan-in 1. To balance this extension on the other side, we add two more buffer AND gates in between the sources x_1 and x_2 and the OR gate which is distance 1 from the sink. The resulting circuit is clearly equivalent. It consists of an AND gate sink, being fed into by a layer of two OR gates, themselves being fed by a layer of three AND gates (two of which are trivial) before finally reaching the sources. The above argument can easily be generalized with an induction over the depth. □

Fact 1.3. *If C is an n-input Boolean circuit of depth d and size S, then there is a circuit of the same depth and size less than or equal to $(10S)^d$ where each gate has fan-out 1.*

Proof. First we assume that all NOT gates have been shoved down to the bottom of the circuit. Suppose we have a gate with fan-out 2 (for simplicity). The idea is to consider the sub-circuit which includes the gate fanning out and everything connected to it. If we simply have the inputs which connect to this circuit fan out into a duplicate of this subcircuit, then we are left with another subcircuit whose output is identical, and which can be fed into a gate so that a fan-out from the gate in question is no longer necessary. As a crude bound, in the worst case this involves creating a copy of subcircuit for each layer of depth, leaving us with one of size $|S|^d$. □

Since the $\boldsymbol{NC}^i$ classes are a restriction of the $\boldsymbol{AC}^i$ classes, it is clear that $\boldsymbol{NC}^i \subseteq \boldsymbol{AC}^i$ for all i. However, it is also the case that

Fact 1.4. $\boldsymbol{AC}^i \subseteq \boldsymbol{NC}^{i+1}$ *for all i.*

Proof. Consider first an arbitrary gate (either AND or OR) with fan-in n for some n. Let l be minimal so that $n \leq 2^l$. We claim that this gate can be represented by a depth l collection of $O(n)$ many of the same gate all of which have fan-in 2. We show this by induction on l. Without loss of generality we will assume we are dealing with AND gates. For $l = 1$, the case is trivial. n equals either 2 or 1. In the first case, our circuit is already in the desired form. In the latter case, we can simply have the input fan-out to a copy of itself, and feed both copies into a single AND gate. Suppose for some l the claim is true - that is, there is a depth l circuit composed of $O(n)$ many copies of AND gates of fan-in 2, implementing an AND gate of fan-in 2^l (for now we will assume $n = 2^l$ exactly). By simply arranging two copies of this circuit side by side, and connecting their outputs by a single and gate in an additional layer, we will obtain a depth $l + 1$ circuit which implements an AND gate of fan-in 2^{l+1}. For $2^l < n < 2^{n+1}$, we can simply use the fan-out trick which we used in the base case for $n = 1$, however many times is necessary. The case of OR gates is identical.

With this known, suppose we have a circuit C of size n^k for some n, and of depth $\log^i(n)$. Note that due to the size of the circuit we can assume that all gates have fan-in at most n^k. By replacing all AND and OR gates of arbitrary fan-in with subcircuits of depth $\log(n^k) = O(\log(n))$, we obtain a new circuit with larger depth which implements the same Boolean function. In the worst case, every level of the circuit has AND or OR gates of fan-in n^k. Thus for each of the $\log^i(n)$ levels, we must add $k \log(n)$ many intermediary levels, resulting in a depth $O(\log^{i+1}(n))$ depth circuit, with size at most $O(n^{k+1})$. Thus if L is a language in $\boldsymbol{AC}^i$, we can use the process described above for each member of the circuit family to create a family of circuits satisfying the conditions of $\boldsymbol{NC}^{i+1}$ deciding L. □

Corollary 1.1. $\boldsymbol{NC}^0 \subseteq \boldsymbol{AC}^0 \subseteq \boldsymbol{NC}^1 \subseteq \boldsymbol{AC}^1 \subseteq \dots$

Corollary 1.2. $\boldsymbol{AC} = \boldsymbol{NC}$.

Note that $\boldsymbol{NC}^0$ is an almost completely mostly trivial class of problems. The reason for this is that it would take at minimum $\log(n)$ many layers of circuits just to link n inputs to a single output. Thus this hardly even qualifies as a class of problems, and as far as I can tell actually has no problems in it. It isn't a real complexity class and shouldn't be treated as one.

Of particular importance to us is the parity function. This can be thought of as a language

$$PARITY = \{x : x \text{ has an odd number of 1's}\}$$

It can also be thought of as a family of Boolean functions. For $n \in \omega$, $f_p(x)$ is simply the modulo 2 sum of the bits of x.

Fact 1.5. *$PARITY \in \boldsymbol{NC}^1$.*

Proof. For $n = 1$, the parity function is simply the value of the bit, so this circuit is trivial. For $n = 2$, the parity function is simply the XOR of the bits. I.e. $f_p(x_1x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_2 \wedge x_2)$. The circuit which computes this function is clear; simply have both inputs fan-out into NOT gates, then feed each of these sources into two AND gates, and finally feed the output of those into a single OR gate. For $n = 3$, we repeat the circuit described for $n = 2$, and then feed that as an input into a copy of the same circuit, with the second input being the third bit. For $n = 4$, feed x_1 and x_2 into copies of this circuit, and then feel the third output into another copy. Notice that this is the same binary tree concept that was used to split an AND gate with unlimited fan-in into a log-depth circuit of AND gates with fan-in 2, except that this time we are using this basic depth 3 XOR circuit in place of a simple AND gate. In any case for $2^l < n < 2^{l+1}$, a depth $3(l + 1) = O(\log(n))$ and size $\leq 5n = O(n)$ circuit then can be constructed to decide parity in the sense we are describing. Thus $PARITY \in \boldsymbol{NC}^1$. □

Before attempting to relate these classes to the complexity classes defined in terms of Turing machines, let us consider the 'biggest' of these classes which could be considered feasible.

Definition 1.2. We denote the set of languages decidable by polynomial size circuits, with no further restrictions, as $\boldsymbol{P_{/poly}}$.

The notation will be clear in a moment. That $\boldsymbol{P} \subseteq \boldsymbol{P_{/poly}}$ is clear by the Cook-Levin Theorem. There we represented Turing machines of varying inputs as polynomial size circuits for the sake of relating them to $CIRCUITSAT$. This class has some issues though.

Fact 1.6. *Every unary language is in $\boldsymbol{P_{/poly}}$. That is, if $L \subseteq \{1^n : n \in \omega\}$, then $L \in \boldsymbol{P_{/poly}}$.*

Proof. Note that for such a language there can be at most one accepted input per string of any length, and it is simply that sequence of 1's. The circuit which accepts such a string is either a single AND gate in the case of unbounded fan-in, or a cascade of AND gates in the cases of fan-in bounded to 2. We can simply let C_n be this circuit for those n such that $1^n \in L$. For those 1^n not in L, we can construct a simply circuit which outputs 0 for all inputs. This can be done by having all inputs fan out into a NOT gate, and then sending all inputs along with their negations into a single AND gate. □

Note that in fact these circuits are $\boldsymbol{AC}^0$ circuits. This is a bit of a problem. Consider the following unary language

$$UHALT = \{1^n : n\text{'s binary expansion encodes a pair } (M, x) \text{ such that } M \text{ halts on input } x\}$$

This is clearly an undecidable language since if it weren't, one could solve the halting problem by coding the appropriate n and running the algorithm for $UHALT$. Thus to adequately relate these classes to computability we must place a restriction on them, known as uniformity.

Definition 1.3. For $i \in \omega$, define **Uniform-$\boldsymbol{AC}^i$** to be the collection of languages which are decidable by uniform circuit families $\{C_n\}_{i\in\omega}$ such that there exists a Turing machine which, given an n, can produce an output string describing C_n using space $O(\log(n))$. Same for $\boldsymbol{NC}^i$. For uniform-$\boldsymbol{P_{/poly}}$, we allow the Turing machines to run in polynomial time.

Unless otherwise noted, any reference to any of these circuit complexity classes will be assumed to be their uniform variants. With this said, we can now demonstrate some relationships that these have with other complexity classes.

Fact 1.7. $\boldsymbol{NC}^1 \subseteq \boldsymbol{L}$

Proof. Let $L \in \boldsymbol{NC}^1$. So there exists a family $\{C_n\}$ of circuits deciding L which are of depth $O(\log(n))$, size $O(n^k)$ for some k, and in which all AND and OR gates have fan-in 2. Also we are assuming uniform $\boldsymbol{NC}^1$, i.e. the circuit C_n can be constructed in logarithmic space by a Turing machine. It is this Turing machine that we exploit in order to construct a log-space circuit deciding L. To do this is as simple as building the appropriate circuit C_n, and then computing $val(C_n)$ for the input x (where $|x| = n$). This however must be done recursively without building the entire circuit, which would likely exceed logarithmic space use. Given that we can construct the circuit itself in logarithmic space, the Church Turing thesis would imply that we can also in logarithmic space decide what the connecting vertices are to a specific gate index. Thus we recursively make our way 'down' from the sink gate to an input, remembering what kind of gate we encountered. These recordings only use logarithmic space since the circuit itself is logarithmic depth. Once we reach a source, we record it, and begin to move up. We delete the appropriate circuit elements on the way back up, keeping the number. If we encounter a NOT gate, we flip the number. If we encounter an OR or and AND gate, we stop moving up and begin moving back down, recording gate types as we did before. The rest is fairly obvious. □

We require a result about adjacency matrices, and for this it will be helpful to define *Boolean* matrix multiplication. If A and B are compatible matrices (we'll assume they are both square $n \times n$ for simplicity and because this is all we actually need) consisting only of 1's and 0's, then we can interpret these as literals, and define $C = AB$ where

$$C_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}$$

For the following results matrix multiplication will always be assumed to be of this form.

Lemma 1.1. *Let $G = (V, E)$ be a graph with $|V| = n$, and let A be the adjacency matrix of A with the addition of 1's along the diagonal, so that A represents vertex reachability by paths of length ≤ 1. Then for any k, A^k represents vertex reachability by paths of length k. (*

Proof. By induction on k. The base case $k = 1$ is trivial. Consider $A^{k+1} = A^k A$. By hypothesis $(A^k)_{ij}$ is a 1 iff there is a path from vertex i to vertex j of length less than or equal to k. Then $(A^{k+1})_{ij} = 1$ iff $\bigvee_{l=1}^{n} (A^k)_{il} \wedge (A)_{lj}$. Let's consider what this Boolean expression is claiming. It is claiming that for at least one l, $(A^k)_{il}$ and $(A)_{lj}$ are both 1. That is, there exists a path of length $\leq k$ from i to l, and then a path of length ≤ 1 from l to j. But this is true iff there is a path of length $\leq k+1$ from i to j (by way of l). This completes the induction. □

Next, consider an $n \times n$ Boolean matrix A. With only $O(n)$ circuit elements arranged in a depth $\log(n)+1$ $\boldsymbol{NC}$ subcircuit, we can easily compute the ij entry of A^2. Simply send, for each l, $(A)_{il}$ and $(A)_{lj}$ into an AND gate, and then fan the rest of these into an OR gate, split into a logarithmic number of layers as we've described many times already. By fanning each of these outputs out to create two new sets of matrix inputs, we can repeat the circuit with another identical layer to have a circuit with sinks corresponding to the entries of A^4. And so on.

Consider the augmented adjacency matrix of a graph A as described above. The maximum path length without cycles is of course n. Thus with $\log(n)$ many layers of the above circuit for computing A^{2^n}, we will have a circuit which computes $A^{2^{\log(n)}} = A^n$. This implies a circuit solution to the reachability problem - given an adjacency matrix for a graph, augmented with 1's along the diagonal, simply compute A^n via $\log(n)$ many layerings of our depth $\log(n)$ subcircuit, and then observe the $(1, 1)$ entry. The final size of such a circuit is of course $O(n \log(n))$, or rather $O(n^2)$.

Corollary 1.3. *$REACH \in \boldsymbol{NC}_2$. Also $REACH \in \boldsymbol{AC}_1$.*

Since $REACH$ is $\boldsymbol{NL}$-complete, we immediately have also that

Corollary 1.4. *$\boldsymbol{NL} \subseteq \boldsymbol{AC}_1$.*

Of course, since $CIRCUITVALUE$ is in $\boldsymbol{P}$, the log-space uniformity condition for $\boldsymbol{AC}$ and $\boldsymbol{NC}$ immediately implies that both are subsets of $\boldsymbol{P}$. Thus we have established the following chain:

$$\boldsymbol{AC}_0 \subseteq \boldsymbol{NC}_1 \subseteq \boldsymbol{L} \subseteq \boldsymbol{NL} \subseteq \boldsymbol{AC}_1 \subseteq \boldsymbol{NC}_2 \subseteq \ldots \subseteq \boldsymbol{AC} = \boldsymbol{NC} \subseteq \boldsymbol{P}$$

2 Parity

We proceed to embark on the long process of showing that the parity function not only isn't in $\boldsymbol{AC}^0$, i.e. cannot be computed by constant depth circuit families of polynomial size, but in fact cannot even be computed by constant depth circuit families of quasipolynomial size. This stronger result is necessary to construct an oracle separation of $\boldsymbol{PH}$ from $\boldsymbol{PSPACE}$. We require some notation and definitions.

Definition 2.1. Let C be an n input circuit, or alternatively a Boolean function (the following definition can be applied to either). A **restriction** is a partial mapping $\rho : A \subseteq \{0,1\}^n \to \{0,1,*\}$ - i.e. a fixing of some inputs as 1's, some as 0's, and others left alone. Those mapped to $*$ are seen as being let alone. For a fixed n, denote $\mathcal{R}_t$ to be the set of restrictions of t many variables (leaving $n-t$ many $*$'s). We denote the restricted circuit $C \upharpoonright \rho$. We call two restrictions ρ and σ on n variables **disjoint** if they restrict a disjoint set of variables to literals. In this case, if ρ restricts k many variables and σ restricts l many variables, then the composition $\rho \circ \sigma := \rho\sigma$ is a restriction in $\mathcal{R}_{k+l}$.

We can also define random restrictions via a parameter p, which represents the probability that an input will not be fixed as a literal (or alternatively, the probability that it *will* be fixed as a $*$). It is assumed that, among those inputs which are fixed as literals, the choice of 0 or 1 is determined by a coin flip. Note that for a circuit of n inputs, a random restriction with parameter p will not fix a guaranteed number of inputs as literals, but the expected number of them will be $n(1-p)$.

Fact 2.1. *Let $f_p : \{0,1\}^n \to \{0,1\}$ be the parity function on n many variables. Then for any $t \leq n$ and any restriction $\rho \in \mathcal{R}_t$, $f_p \upharpoonright \rho$ is either the parity function on $n-t$ variables, or it's negation is the parity function on $n-t$ variables.*

Proof. Suppose that an even number of literals fixed by ρ are fixed to 1. Then for a string with an odd number of 1's plugged into the remaining inputs, the restricted function will output 1 because the 'concatenation' of the restricted inputs with the actual new input string will still have an odd number of 1's. On the other hand if the new input string has an even number of inputs, then the total string input will also have an even number of inputs and thus the restricted function will output 0. On the other hand, if the restriction fixes an odd number of 1's, then by the same inspection we will see that the restricted function computes the negation of the parity function.

□

We can see then that if we restrict and number of variables of the parity circuit, we must *always* have either a parity circuit, or a circuit which we can attach a single not gate to create a parity circuit. The next thing we should note in preparation for the big result is even more obvious:

Fact 2.2. *The n input parity function is not simply the AND or the OR of any combination of inputs or their negations, for any $n > 1$*

Proof. The parity function has the property that for any n, exactly half of the inputs yield 1 and exactly half yield 0. Meanwhile, the ANDing of n inputs, regardless of which ones may or may not be negated prior to entering the gate, has the property that only a single combination of inputs will result in a 1, with all others returning a 0. Thus it is impossible, aside from the trivial case of $n = 1$, for an AND gate to decide parity, since it can never obtain symmetry between its 0 and 1 outputs. Likewise for the operation of OR. For these gates, all but one of the combinations will result in a 1, and so symmetry is impossible to obtain. □

Thus there do not exist any 'depth 1' circuit families for parity, nor do 'depth 2' circuit families in which one of the circuit layers is NOT gates. The next stage up from this are the special case of depth 3 circuit families in which each circuit is a DNF or a CNF, either an OR of AND's of various inputs and their negations, or an AND of OR's of various inputs and their negations. Generally these are considered to be depth 2 circuit families, since one can rather than have a layer of negations simply have a $2n$ input circuit in which every input also has it's negation as a separate input. It is clear that DNF circuits exist for parity, since one can simply have an AND gate for every different string such that the output is 1, and sending those to an OR gate. Note that this requires exactly 2^{n-1} AND gates, since the parity function has an equal number of 0 and 1 outputs. Also note that each AND gate takes all n inputs (some of which will be negations, but exactly 1 for each bit).

Fact 2.3. *If C is a CNF or DNF circuit deciding parity on n variables, then each term/clause must have fan-in n (i.e. every term/clause must take in every variable) and there must be at least 2^{n-1} terms/clauses.*

Proof. Let's start with the case of a CNF circuit. Suppose it decides parity and that one of it's terms doesn't take one of the inputs. Suppose the inputs are fixed in such a way that this particular term is 0. Then the output of the circuit itself will still be 0. However if we take that input and flip just the one bit that is missing from the term, the term will remain 0, and thus the circuit output will also remain 0. But now our circuit can no longer be deciding parity, since flipping a single bit flips the parity. Thus all terms must take in all n inputs. On the other hand, suppose we have a DNF circuit in which one of the clauses is missing an input. Fix an input such that the term is 1. This will render the output of the circuit is 1. But once again, if we flip the one bit missing from the circuit, then the term remains 1, and so will the output. Again we have a contradiction.

Turning to the number of terms/clauses, start with a CNF circuit deciding parity. Each OR gate outputs 0 on exactly one input sequence. Thus for every string such that parity is 0, we must have a different OR gate corresponding to this output. Since the parity function on n variables always accepts exactly 2^{n-1} strings, this means we need that many OR gates. Similarly for a DNF circuit: each and gate outputs 1 on exactly 1 input sequence. Thus we need one AND gate for every accepting string, again meaning at least 2^{n-1} many □

Corollary 2.1. *Any depth 2 circuit family deciding the parity language must be of size at least $2^{n-1} = O(2^n)$ and thus cannot be an $\boldsymbol{AC}^0$ family.*

We now state the switching lemma.

Theorem 2.1 (Hastad's Switching Lemma)**.** *Let G be a CNF circuit of bottom fanin $\leq t$ and ρ a random restriction with parameter p. Then the probability that $G \restriction \rho$ cannot be written as a DNF circuit of bottom fanin $\leq s$ is bounded by $(\alpha)^s$, where α is the unique positive root of the equation*

$$(1 + \frac{4p}{1+p}\frac{1}{\alpha})^t = (1 + \frac{2p}{1+p}\frac{1}{\alpha})^t + 1$$

Furthermore $\alpha = \frac{2pt}{\ln(\phi)} + O(p^2 t) < 5pt$ for sufficiently small (all?) p, where ϕ is the golden ratio. Moreover, without changing the bounds one can add the condition that not only is the new fanin $\leq s$, but in the case of a DNF circuit each clause accepts a disjoint set of inputs (i.e. if one is 'on', all others are 'off'), and in the case of a CNF circuit each term rejects a set of inputs disjoint from the others.

To prove this, we prove a technically stronger lemma more suitable to induction. Before that though we need some terminology. Define a **minterm** of a Boolean function G to be a restriction σ such that $G \restriction \sigma \equiv 1$, but no proper subrestriction $\sigma' \subset \sigma$ is sufficient to do this. I.e. a minterm is a minimally sized restriction which trivializes G. The size of a minterm is the number of variables which are set to literals by the restriction. The importance of minterms is that if we think of them as AND's of it's variables (i.e. if a minterm assigns x_1 to 1, x_5 to 0 and x_6 to 0, then the corresponding term is $x_1\bar{x}_2\bar{x}_3$), then the disjunction of all of these terms is equivalent to the function itself. (One direction of confirming this is trivial. The other direction is simple too: if $G \equiv 1$ under some fixing of literals σ, then this σ can be thinned out to a minterm, and this minterm will be green for the DNF circuit, making the disjunction itself green.) Therefore suppose we start with a CNF circuit G, and apply a random restriction ρ. Let $min(G) \geq s$ be the event that, after a random restriction ρ is applied to G, there exists a minterm of size greater than or equal to s. It follows that

$$\neg(min(G) \geq s) \iff G \restriction \rho \text{can be expressed as an s-DNF circuit}$$

Lemma 2.1. *Let $G = \bigwedge_{i=1}^{w} G_i$ where G_i are OR's of fanin $\leq t$. Let F be an arbitrary Boolean function (on some or none or all of the variables of G, doesn't matter). Let ρ be a random restriction in R_ρ. For $s \in \omega$, let $min(G) \geq s$ denote the event that $G \restriction \rho$ has a minterm of size greater than or equal to s*

$$P(min(G) \geq s | F \restriction \rho \equiv 1) \leq \alpha^s$$

This statement is stronger since by simply picking $F \equiv 1$ we obtain an unconditional statement. The F here is purely technical for the same of getting the result - it has no real philosophical content, as far as I can tell.

Proof. We go by induction on w. The case $w = 0$ is trivial. Suppose now that the statement is true for all values less than w for some fixed w. The analysis will focus on the first OR gate, G_1. First note that in general for any events A, B, C, $P(A|B) \leq \max\{P(A|B \cap C), P(A|B \cap \bar{C})\}$. This is because if without loss of generality $P(A|B \cap C) \geq P(A|B \cap \bar{C})$, then

$$\begin{aligned} P(A|B) &= P(A \cap C|B) + P(A \cap \bar{C}|B) \\ &= P(C)P(A|B \cap C) + P(\bar{C})P(A|B \cap \bar{C}) \\ &\leq P(C)P(A|B \cap C) + P(\bar{C})P(A|B \cap C) \\ &= P(A|B \cap C) \end{aligned}$$

After applying the restriction ρ, either G_1 is forced to be 1 or it isn't. Therefore by the above we can say

$$\begin{aligned} &P(min(G) \geq s|F \upharpoonright \rho \equiv 1) \\ &\leq \max\{P(min(G) \geq s|F \upharpoonright \rho \equiv 1 \wedge G_1 \upharpoonright \rho \equiv 1), P(min(G) \geq s|F \upharpoonright \rho \equiv 1 \wedge G_1 \upharpoonright \rho \not\equiv 1)\} \end{aligned}$$

It thus suffices to show that both of these conditional probabilities are less than α^s. Beginning with the first one, note that if $G_1 \upharpoonright \rho \equiv 1$, then $G \upharpoonright \rho$ is effectively a CNF circuit with $w - 1$ many OR gates. Furthermore $F \wedge G_1 \upharpoonright \rho \equiv 1$. Since the inductive hypothesis applies to all Boolean functions, we can take this conjunction as the F and apply the inductive hypothesis to immediately obtain the bound α^s in this first case. The remainder of the proof will deal with the second.

Considering G_1 more closely, let

$$G_1 = \bigvee_{i \in T} x_i$$

where $|T| \leq t$. We are assuming without loss of generality that all inputs to G_1 are positive (i.e. not negated). Let $\rho = \rho_1\rho_2$, where ρ_1 is the restriction to the variables in T and ρ_2 the rest of it. Then $G_1 \upharpoonright \rho \not\equiv 1$ is equivalent to the condition that ρ_1 assigns only 0's and stars to variables in T, and also equivalent to $G_1 \upharpoonright \rho_1 \not\equiv 1$. (This is why we can assume positive values for G_1 wlog; if some of them are negative then just ρ_1 as assigning positive values to negations.) Observe next that since G_1 is not trivialized by ρ, any minterm σ will have to set some variable in T to a 1. (The minterm can still assign 0's and 1's to other variables in T though.) What we are going to do is partition the minterms of $G \upharpoonright \rho$ according to what variables in T they give values to. Call such a subset Y.

A minterm of $G \upharpoonright \rho$ must give values to the variables which are still variables after ρ_1. We'll denote the event that ρ_1 assigns stars to all variables in Y by $\rho(Y) = *$. Furthermore let $min(G)^Y \geq s$ denote the event that $G \upharpoonright \rho$ has a minterm of size at least s whose restriction to the variables in T assigns value to the variables of Y. Note that the union of these events for all $Y \subseteq T$ (each conjuncted with $\rho_1(Y) = 0$) is equivalent to the event $min(G) \geq s$. Therefore we have

$$\begin{aligned} &P(min(G) \geq s|F \upharpoonright \rho \equiv 1 \wedge G_1 \upharpoonright \rho_1 \not\equiv 1) \\ &\leq \sum_{Y \subseteq T, Y \neq \varnothing} P(min(G)^Y \geq s \wedge \rho_1(Y) = *|F \upharpoonright \rho \equiv 1 \wedge G_1 \upharpoonright \rho_1 \not\equiv 1) \\ &= \sum_{Y \subseteq T, Y \neq \varnothing} P(\rho_1(Y) = *|F \upharpoonright_\rho \equiv 1 \wedge G_1 \upharpoonright_{\rho_1} \not\equiv 1) \times P(min(G)^Y \geq s|F \upharpoonright_\rho \equiv 1 \wedge G_1 \upharpoonright_{\rho_1} \not\equiv 1 \wedge \rho_1(Y) = *) \end{aligned}$$

For the remainder of the proof we can now assume a fixed Y and focus on the two probabilities inside of the sum. For the first and obviously simpler one, we actually begin by looking at an even simpler probability. We claim that

$$P(\rho_1(Y) = *|G_1 \upharpoonright_{\rho_1} \not\equiv 1) = \left(\frac{2p}{1+p}\right)^{|Y|}$$

To see this, note that $G_1 \upharpoonright_{\rho_1} \not\equiv 1$ is equivalent to saying that $\rho_1 x_i$ is either a 0 or a $*$ for each $i \in T$. Thus for a fixed $i \in Y$,

$$\begin{aligned} P(\rho_1(x_i) = *|\rho_1(x_i) = 0 \vee \rho_1(x_i) = *) &= \frac{P(\rho_1(x_i) = *)}{P(\rho_1(x_i) = 0) + P(\rho_1(x_i) = *)} \\ &= \frac{p}{\frac{1-p}{2} + p} \\ &= \frac{2p}{1+p} \end{aligned}$$

Since variable assignments are independent this gives us that $P(\rho_1(Y) = *|G_1 \restriction_{\rho_1} = \left(\frac{2p}{1+p}\right)^{|Y|}$. To show that this is also a bound for what we want, observe the following general observation:

For any events A, B, C, $P(A|B \wedge C) \leq P(A|C)$ is equivalent to $P(B|A \wedge C) \leq P(B|C)$

Seeing $A = (\rho_1(Y) = *)$, $B = (F \restriction_\rho \equiv 1)$, and $C = (G_1 \restriction_{rho_1} \not\equiv 1)$, the probabiltiy we currently have a number for it $P(A|C)$, and we wish to show that $P(A|B \wedge C)$ is less than this. By the claim above it suffices to show that

$$P(F \restriction_\rho \equiv 1|\rho_1(Y) = * \wedge G_1 \restriction_{\rho_1} \not\equiv 1) \leq P(F \restriction_{\rho_1} \equiv 1|G_1 \restriction_{\rho_1} \not\equiv 1)$$

But this is obviously true by inspection: the condition that $\rho_1(Y) = *$ cannot increase the probability that a function is determined. (The very idea of 'restricting to a star' is that we are leaving inputs *undetermined.* This kind of a restriction could only ever decrease the probability that a function is trivialized.) Therefore, contingent on showing that general fact from probability theory, we have that

$$P(\rho_1(Y) = *|F \restriction_\rho \equiv 1 \wedge G_1 \restriction_{\rho_1} \leq \left(\frac{2p}{1+p}\right)^{|Y|}$$

To see the general claim, simply note that

$$\frac{P(A \cap B \cap C)}{P(B \cap C)} \leq \frac{P(A \cap C)}{P(C)}$$

Simply multiply both sides by $P(B \cap C)$ and divide both sides by $P(A \cap C)$ to get the equivalent. Now we deal with the second term. We must bound

$$P(min(G)^Y \geq s|F \restriction_\rho \equiv 1 \wedge G_1 \restriction_{\rho_1} \not\equiv 1 \wedge \rho_1(Y) = *)$$

To do this we first decide to start thinking about minterms as consisting of two parts, i.e. $\sigma = \sigma_1\sigma_2$, where σ_1 is the assignments to variables in Y, and σ_2 assigns values to variables in the complement of T (note that since we are assuming no assignments to variables outside of Y, this is a complete partition of the minterm). The key is that if σ is a minterm of $G \restriction_\rho$, then σ_2 is a minterm of $G \restriction_{\rho\sigma_1}$, and this restricted G has effectively one less OR gate attached to it due to the trivialized G_1. The goal is to somehow express the probability we want in such a way that the inductive hypothesis can be applied to this circuit. Doing this requires careful consideration of the events being conditioned on. We need to 'distill out' any reference to G_1, and express our probability in terms of a probability which ignores any variables connected to G_1, and is under the assumption that G_1 is taken care of already. Observe

$$\begin{aligned} &P(min(G)^Y \geq s|F \restriction_\rho \equiv 1 \wedge G_1 \restriction_{\rho_1} \not\equiv 1 \wedge \rho_1(Y) = *) \\ &\leq \max_{\rho_1(Y)=*, \rho_1(T) \in \{0,*\}^{|T|}} P_{\rho_2}(\min(G)^Y \geq s|(F \restriction_{\rho_1}) \restriction_{\rho_2} \equiv 1) \end{aligned}$$

Where P_{ρ_2} is there to emphasize that this is a probability specifically taken over variables *not connected to* G_1. However, we still need to address G_1, since the event $\min(G)^Y \geq s$ involves it. To deal with this, for a fixed assignment to variables in G_1, σ_1, let $\min(G)^{Y,\sigma_1}$ denote the event that $G \restriction_{\rho_1}$ has a minterm of size greater than or equal to s which agrees with σ_1 on those variables. This allows us to further partition our probability, not just in terms of the set of variables Y receiving assignments in G_1 but also now in terms of the actual details of those restrictions. We have

$$\begin{aligned}
&\max_{\rho_1(Y)=*,\rho_1(T)\in\{0,*\}^{|T|}} P_{\rho_2}(\min(G)^Y \geq s|(F\restriction_{\rho_1})\restriction_{\rho_2}\equiv 1)\\
&\leq \sum_{\sigma_1\in\{0,1\}^{|Y|},\sigma_1\neq 0^{|Y|}} \left(\max_{\rho_1(Y)=*,\rho_1(T)\in\{0,*\}^{|T|}} P_{\rho_2}(\min(G)^{Y,\sigma_1} \geq s|(F\restriction_{\rho_1\sigma_1})\restriction_{\rho_2}\equiv 1)\right)
\end{aligned}$$

We are so close to being able to apply the inductive hypothesis. The only remaining consideration is that $G\restriction_{\rho_1\sigma_1}$ might depend on variables in $T-Y$. However, since we are explicitly fixing out minterm to deal with variables in Y, we are not considering these variables at all. Thus any variables connected to G_1 have nothing to do with the circuit we are really talking about. They might as well not be there. Finally, we can apply the inductive hypothesis. Since σ_1 is a minterm for $G_1\restriction_{\rho_1}$, we are really discussing minterms of size $s-|Y|$. Thus

$$\begin{aligned}
&\sum_{\sigma_1\in\{0,1\}^{|Y|},\sigma_1\neq 0^{|Y|}} \left(\max_{\rho_1(Y)=*,\rho_1(T)\in\{0,*\}^{|T|}} P_{\rho_2}(\min(G)^{Y,\sigma_1} \geq s|(F\restriction_{\rho_1\sigma_1})\restriction_{\rho_2}\equiv 1)\right)\\
&\leq \sum_{\sigma_1\in\{0,1\}^{|Y|},\sigma_1\neq 0^{|Y|}} \alpha^{s-|Y|}// \qquad = (2^{|Y|}-1)\alpha^{s-|Y|}
\end{aligned}$$

Finally, putting it all together, (and including the $Y=\varnothing$ case, since we know that the term associated with it will be 0) we have

$$\begin{aligned}
P(min(G)\geq s|F\restriction_{\rho}\equiv 1 \wedge G_1\restriction_{\rho_1}\not\equiv 1) &\leq \sum_{Y\subseteq T}\left(\frac{2p}{1+p}\right)^{|Y|}(2^{|Y|}-1)\alpha^{s-|Y|}\\
&\leq \alpha^s\sum_{i=0}^{|T|}\binom{|T|}{i}\left[\left(\frac{4p}{1+p}\frac{1}{\alpha}\right)^i-\left(\frac{2p}{1+p}\frac{1}{\alpha}\right)^i\right]\\
&= \alpha^s\left[\left(1+\frac{4p}{1+p}\frac{1}{\alpha}\right)^{|T|}-\left(1+\frac{2p}{1+p}\frac{1}{\alpha}\right)^{|T|}\right]\\
&\leq \alpha^s\left[\left(1+\frac{4p}{1+p}\frac{1}{\alpha}\right)^{t}-\left(1+\frac{2p}{1+p}\frac{1}{\alpha}\right)^{t}\right]\\
&= \alpha^s
\end{aligned}$$

The second to last step here, replacing $|T|$ with t, can be done since $|T|\leq t$ and we will see in a moment that the entire difference is increasing in this argument. This completes the induction.

Finally, we must deal with the actual mess of an equation. First we show that $\alpha \leq 5pt - O(p^2t)$ for all $p\in[0,1], t\in\omega$. First we demonstrate that positive solutions exist. To see this we exploit the fact that t is an integer and rewrite the equation in terms of it's binomial expansion.

$$\begin{aligned}
&\sum_{i=1}^{n}\binom{n}{i}\left(\frac{4p}{1+p}\frac{1}{\alpha}\right)^i-\sum_{i=1}^{t}\binom{t}{i}\left(\frac{2p}{1+p}\frac{1}{\alpha}\right)^i = 1\\
&\implies -1+\sum_{i=1}^{t}\binom{n}{i}2^i(2^i-1)\left(\frac{p}{(1+p)\alpha}\right)^i = 0
\end{aligned}$$

By Descartes' rule of signs this polynomial in $\frac{p}{(1+p)\alpha}$ has at most a single positive root (since all coefficients except for the -1 are positive). Again since all of the coefficients are positive, it is clear that this polynomial can be made as large as desired by choice of sufficiently small α. Further we can observe that by choice of sufficiently large α, the function can be made infinitely close to 0 regardless of t or p. Since polynomials are continuous we can conclude that there must be an $0 < \alpha < \infty$ which makes the polynomial sum 1, rendering the equation 0. Thus a unique positive solution α exists for any $p \in [0,1]$ and any $t \in \omega$. Since we now know there is a unique positive root, we can assume in advance the solution form $\alpha = \frac{2pt}{\gamma}$ for some $\gamma > 0$. The equation then becomes

$$\left(1 + \frac{2\gamma}{(1+p)t}\right)^t = \left(1 + \frac{\gamma}{(1+p)t}\right)^t + 1$$

Fix p, and consider the asymptotic solution as $t \to \infty$. The equation approaches

$$e^{(\frac{\gamma}{1+p})^2} = e^{\frac{\gamma}{1+p}} + 1$$

Seeing $e^{\frac{\gamma}{1+p}}$ as a single variable, we find ourselves staring at the polynomial equation for the golden ratio ϕ. Thus

$$e^{\frac{\gamma}{1+p}} = \phi \implies \gamma = (1+p)\ln(\phi)$$

Yielding the asymptotic solution

$$\alpha = \frac{2pt}{\ln(\phi)} \frac{1}{1+p} = \frac{2pt}{\ln(\phi)}(1 - p + p^2 - p^3 + \dots) = \frac{2pt}{\ln(\phi)} - O(p^2 t) \leq 5pt$$

Let $f(a,t) = \left(1 + \frac{a}{t}\right)^t$. Our overall equation is then

$$F(\gamma, t) := f(\frac{2\gamma}{1+p}, t) - f(\frac{\gamma}{1+p}, t) = 1$$

Suppose we were to show that F were increasing in both arguments. Then were we to have a solution in γ, we would have for any higher t that this difference must be greater than 1, meaning that γ must change, and furthermore would have to shrink, since F is increasing in it. This in turn would mean that $\alpha = \frac{2pt}{\gamma}$ would have to grow. Thus we would have at that point that the *solutions* α must be increasing with t. It would follow then that our asymptotic solution in t is bigger than all other solutions for an arbitrary fixed p, securing our bound. Thus it suffices to show that F is increasing in both arguments. (Call the first argument a.)

We begin by showing that $\frac{\partial}{\partial t} F(a,t) > 0$. To do this it suffices to show that $\frac{\partial}{\partial t} f(a,t)$ is increasing in a, since then

$$\frac{2a}{1+p} > \frac{a}{1+p} \implies \frac{\partial}{\partial t} f(\frac{2a}{1+p}, t) - \frac{\partial}{\partial t} f(\frac{a}{1+p}, t) > 0$$

But by pulling out the partial derivative operation this is precisely the statement that $F(a,t)$ is increasing in t. Recalling some logarithmic differentiation:

$$\begin{aligned}\frac{\partial}{\partial t}(1 + \frac{a}{t})^t &= (1 + \frac{a}{t})^t \frac{\partial}{\partial t} t \ln(1 + \frac{a}{t}) \\ &= (1 + \frac{a}{t})^t (\frac{t}{1 + \frac{a}{t}} + \ln(1 + \frac{a}{t})) \\ &= t(1 + \frac{a}{t})^{t-1} + (1 + \frac{a}{t})^t \ln(1 + \frac{a}{t})\end{aligned}$$

It is clear that this is indeed increasing with a for any $t \geq 1$, thus demonstrating that $F(a,t)$ is increasing in t. Turning to a, we go directly.

$$\begin{aligned}\frac{\partial}{\partial a}\left(1 + \frac{2a}{(1+p)t}\right)^t - \left(1 + \frac{a}{(1+p)t}\right)^t &= \left(1 + \frac{2a}{(1+p)t}\right)^t \frac{\partial}{\partial a} t \ln(1 + \frac{2a}{(1+p)t}) - \left(1 + \frac{a}{(1+p)t}\right)^t \frac{\partial}{\partial a} t \ln(1 + \frac{a}{(1+p)t}) \\ &= \frac{1}{1+p}\left[\left(1 + \frac{2a}{(1+p)t}\right)^t \frac{2}{1 + \frac{2a}{(1+p)t}} - \left(1 + \frac{a}{(1+p)t}\right)^t \frac{1}{1 + \frac{a}{(1+p)t}}\right] \\ &= \frac{1}{1+p}\left[2\left(1 + \frac{2a}{(1+p)t}\right)^{t-1} - \left(1 + \frac{a}{(1+p)t}\right)^{t-1}\right]\end{aligned}$$

This final term is clearly greater than 0 since the first bracketed term is always bigger than the second (and when $t = 1$, the entire quantity in brackets is 1). Thus the function is increasing in both a and t, completing the general bound on α. □

Corollary 2.2. *There exists an increasing sequence $\{n_k\}_{k\in\omega}$ such that for all $k \in \omega$ (greater than 1) and all $n > n_k$, the parity function on n variables cannot be computed by a depth k circuit containing less than or equal to $2^{\frac{1}{10}n^{\frac{1}{k-1}}}$ subcircuits of depth at least 2 and of bottom fanin less than or equal to $\frac{1}{10}n^{\frac{1}{k-1}}$*

Proof. By induction on k. For $k = 2$, there is of course only a single subcircuit of depth 2, but we have also seen that the bottom fanin must be exactly $n > \frac{1}{10}n$ for all bottom level gates. This establishes the base case. Note that this is true for all $n \geq 1$, so initially we set $n_0 = 1$.

Assume the statement holds up to some k, and suppose by way of contradiction that we have such a circuit deciding parity for some $n > n_0^k$ which has $\leq 2^{\frac{1}{10}n^{\frac{1}{k-1}}}$ subcircuits of depth at least 2 and bottom fanin $\leq \frac{1}{10}n^{\frac{1}{k-1}}$. Apply a random restriction with $p = n^{-\frac{1}{k+1}}$, and let $s = \frac{1}{10}n^{\frac{1}{k-1}}$. Consider bottom level depth two subcircuit of this and assume without loss of generality that all are CNF circuits (WLOG we can assume they are either all one or all the other, and the same argument works either way). By the switching lemma the probability that one of these cannot be written as a DNF circuit of bottom fanin s is at most α^s, where

$$\alpha < 5pt = 5n^{-\frac{1}{k-1}}(\frac{1}{10}n^{\frac{1}{k-1}}) = \frac{1}{2}$$

(for sufficiently small alpha?!) There are by assumption at most $2^{\frac{1}{10}n^{\frac{1}{k-1}}} = 2^s$ such subcircuits, so the probability that at least one of them fails to be expressible as an s-DNF is at most $2^s\alpha^s = (2\alpha)^s < 1$. Therefore the probability that *all* of these depth two subcircuits can be re-expressed as an s-DNF is at least $1 - (2\alpha)^s > 1$. Since this probability is positive, it follows that there must exist a restriction ρ which allows us to replace all bottom level depth 2 t-CNF subcircuits with depth 2 s-DNF subcircuits. Moreover the expected number of $*$'s under the random restriction ρ is $np = n \times n^{-\frac{1}{k-1}} = n^{\frac{k-2}{k-1}} := m$. We need to be sure that the random restriction we are choosing fixes at least this many inputs. Note however that the standard deviation of the number of $*$'s is $npq = m(1 - n^{-\frac{1}{k-1}}) \to 0$ as $n \to \infty$. Thus for sufficiently large n the probability of getting any less than m $*$'s will become infinitesimally small (as will the probability of getting any more but this is besides the point). Thus there exists an $n_k > n_{k-1}$ such that the restriction ρ guaranteed to exist by the switching lemma leaves us with a circuit acting on at least $m = n^{\frac{k-2}{k-1}}$ inputs. We will assume for simplicity that it is exactly this many. While we're at it, we also make sure that n_k is big enough that $m > n_{k-1}$ (we can always go bigger if we need to, to ensure this.)

Without loss of generality though we know that such a constant depth circuit is strictly alternating in its AND and OR's. Thus under this restriction we have two consecutive levels of OR gates, which can be collapsed into a single layer. Thus the depth of the circuit has reduced to $k-1$. What we are left with then is a circuit acting on m inputs, where $n^{\frac{1}{k-1}} = m^{\frac{1}{k-2}}$, and since the restriction of a parity circuit is either a parity circuit or the negation of one, we can assume this circuit is deciding parity on m inputs. (If it isn't deciding parity, just attach a NOT gate and filter it down to the bottom.) The bottom fanin of this circuit is by design $s = \frac{1}{10}n^{\frac{1}{k-1}} = \frac{1}{10}m^{\frac{1}{k-2}}$. It remains to count the number of subcircuits of depth at least 2. To do this, note that for every depth at least 2 subcircuit of this new circuit, there must have been at least one depth at least 3 circuit in the previous circuit. Thus the number of depth at least 2 subcircuits is no more than the number of depth at least 3 subcircuits of the previous circuit, which is of course itself no more than the number of depth at least 2 circuits of the previous circuit. Therefore our new m input parity circuit has at most $2^{\frac{1}{10}n^{\frac{1}{k-1}}} = 2^{\frac{1}{10}m^{\frac{1}{k-2}}}$ subcircuits of depth at least 2. But now we have demonstrated that this is a circuit which cannot exist by the inductive hypothesis. This completes the induction. □

Corollary 2.3. *There is an increasing sequence $\{n_k\}_{k\in\omega}$ such that for all k, there are no depth k parity circuits on $n > n_k$ many inputs which are of size less than $2^{\frac{1}{10}^{\frac{k}{k-1}}n^{\frac{1}{k-1}}}$.*

Proof. Suppose that such a circuit existed for some k and some $n > n_k$. Suppose without loss of generality that bottom level depth 2 subcircuits are CNF circuits, and then trivially extend each bottom level gate to create a depth $k+1$ circuit in which all bottom level gates are AND gates with fanin 1. Consider a random restriction ρ with $p = \frac{1}{10}$. Choose $s = \frac{1}{10}^{\frac{k}{k-1}}n^{\frac{1}{k-1}}$. Then plugging into the switching lemma we see $\alpha < 5pt = \frac{1}{2}$ (since $t = 1$), so the probability that any bottom level DNF subcircuit can't be re-expressed as an s-CNF is at most α^s. The trivial extension to fanin 1 gates obviously adds quite a few subcircuits to sum our probability over, but nonetheless the number of subcircuits of depth at least 2 is still bounded by the size, which we have an explicit upper bound for. Thus the probability that at least one of the subcircuits isn't expressible as an s-CNF is at most

$$2^{\frac{1}{10}^{\frac{k}{k-1}} n^{\frac{1}{k-1}}} \alpha^{\frac{1}{10}^{\frac{k}{k-1}} n^{\frac{1}{k-1}}} = (2\alpha)^s < 1$$

Thus the probability that all of them can be expressed as an s-CNF is at least $1-(2\alpha)^s > 0$. Therefore such a restriction must exist. Again considering the number of inputs for this new circuit, we have an expected value $m = \frac{n}{10}$. Note in particular here that, for large enough n, the size of this restriction is extremely likely to leave us with a much larger number of $*$'s than if we had restricted with $p = n^{-\frac{1}{k-1}}$ like in the previous proof.

For simplicity of the demonstration, suppose for the moment that under the restriction guaranteed to exist by the switching lemma, we have restricted to exactly $m = \frac{n}{10}$ many variables, i.e. $10m = n$. Firstly, this is a depth k circuit which can be assumed to decide parity on m inputs (again, just as before, if it does not, simply filter a NOT gate down or perform the equivalent rewiring of the inputs). It's bottom fanin, in terms of m, is

$$\frac{1}{10^{\frac{k}{k-1}}} n^{\frac{1}{k-1}} = \frac{1}{10^{\frac{1}{k-1}+1}} (10)^{\frac{1}{k-1}} m^{\frac{1}{k-1}} = \frac{1}{10} m^{\frac{1}{k-1}}$$

Moreover observe that the number of subcircuits of depth at least 2 is exactly what it was before we did the restriction. This is because what we did via our trivial extension is effectively only add bottom level gates. Thus the number of subcircuits of depth at least 2 is certainly bounded below the total number of gates, ie

$$\leq 2^{\frac{1}{10}^{\frac{k}{k-1}} n^{\frac{1}{k-1}}} = 2^{\frac{1}{10}^{\frac{1}{k-1}} \frac{1}{10} (10)^{\frac{1}{k-1}} m^{\frac{1}{k-1}}} = 2^{\frac{1}{10} m^{\frac{1}{k-1}}}$$

Thus this is a parity circuit that cannot exist by the above corollary. Of course, m isn't *exactly* $\frac{n}{10}$. However, the important thing is that it *can* be assumed to be much larger than n_{k-1}, as we previously observed, and this is all we need to arrive at the same contradiction. □

Corollary 2.4. *PARITY* $\notin \boldsymbol{AC}^0$. *In fact, for all k, not only are there no polynomial size constant depth circuit families for parity, there also aren't any quasipolynomial size circuits for the parity function in general. I.e. for any c and l, there exists an* n_k *(the same* n_k *from above) such that for all* $n > n_k$*, no depth k circuits of size less than* $n^{c \log^l(n)}$ *decide parity on n variables.*

Corollary 2.5. $\boldsymbol{AC}^0 \neq \boldsymbol{NC}^1$.

The nonexistence of quasipolynomial circuits for parity is necessary for the construction of an oracle separating $\boldsymbol{PSPACE}$ from $\boldsymbol{PH}$.

3 Oracle Separation of PH from PSPACE

In this section we will construct an oracle A such that $\boldsymbol{PH}^A \neq \boldsymbol{PSPACE}^A$. Our test language witnessing the separation will be

$$L_A = \{1^n : \text{The number of strings of length } n \text{ in } A \text{ is odd}\}$$

As is usually the case with these kinda of constructions, we can see that $L_A \in \boldsymbol{PSPACE}^A$ regardless of the details of A. To see this, note that for any n there are at most 2^n strings of length n in A, and it will take n many bits to maintain a counter which goes this high. Simply query the machine for every string of length n, adding 1 to the counter if the string is in A, then inspect the lowest place value of the counter, returning that number.

Recall that a language L is in $\boldsymbol{PH}^A$ iff there exists a $k, l \in \omega$ and a relation $R^A(x_1, ..., x_k, y)$ which is polynomial time computable relative to A (where the polynomial is degree l) such that either

$$y \in L \iff \exists^p x_1 \forall^p x_2 \dots Q^p x_k R^A(x_1, \dots, x_k, y)$$

or

$$y \in L \iff \forall^p x_1 \exists^p x_2 \ldots Q^p x_k R^A(x_1, \ldots, x_k, y)$$

Where in both cases Q is $\exists$ if l is odd, and $\forall$ if l is even, and where for either one Q^p refers to quantification over strings of length $|y|^l$ (ie polynomial length). We will assume that the master alphabet under which languages are expressed is $\{0,1\}$. Note that we can enumerate all '***PH*** machines' by simply enumerating all machines generally and then dovetailing that enumeration with an enumeration over expressions of the above form. We fix such an enumeration for the purpose of diagonalizing out of every such 'machine' such that the R relation is polynomial time computable. Since it suffices to diagonalize out of every $\Sigma_i^{P,A}$ class, we will focus entirely on the first of these two forms of a quantified 'constant depth' expression.

We first need a few small lemmas.

Lemma 3.1. *Let*

$$\sigma(y) = \exists^P x_1 \forall^P x_2 \ldots Q^P x_i R^A(x_1, ..., x_i, y)$$

be a sentence in $\Sigma_i^{P,A}$*. Then there is an equivalent sentence*

$$\bar{\sigma}(y) = \exists^P x_1 \forall^P x_2 \ldots Q^P x_{i+2} R^A(x_1, ..., x_{i+2}, y)$$

in $\Sigma_{i+2}^{P,A}$ *where* $\bar{R}^A(x_1, ..., x_{i+2}, y)$ *is a predicate that can be computed by a Turing machine that makes at most one oracle query and runs in polynomial time.*

Essentially what this lemma is saying is that given an oracle machine at any level of the polynomial hierarchy, we can descend a bit further down the hierarchy to find the same problem decided by, at its bottom, a machine which only makes a single query, shunting the multiple queries within the quantification itself.

Proof. The intuitive explanation above gives us a clear idea of how to proceed. We show that $R^A(x_1, ..., x_n, y)$ can be expressed as $\exists^P r \forall^P s \bar{R}^A(x_1, ..., x_{i+2}, r, s, y)$ where $\bar{R}^A(x_1, ..., x_{i+2}, r, s, y)$ can be computed by a machine which makes only one oracle query and runs in polynomial time. Substituting this quantified sentence for R^A in σ then clearly gives us the new expression $\bar{\sigma}$. Let $M_R^A(x_1, ..., x_i, r, s, y)$ be a polynomial time machine that computes the predicate $R^A(x_1, ..., x_i, y)$. That is to say, given the inputs $x_1, ..., x_i$, the machine, on input y, runs a polynomial number of steps. At step, say, j, the machine might query the string q_j, and receive an answer r_j. r_j is really a single bit here, representing whether or not $q_j \in A$. One can then, for a fixed input y, visualize a binary tree T, whose nodes represent the query strings q_j, and whose edges represent the path to the next step's query given whether or not $q_j \in A$. Specifying an oracle A then really specifies a polynomial length path through this tree, with leaves representing whether or not the machine itself accepts.

Note that the predicate $R^A(x_1, ..., x_i, y)$ is equivalent to the sentence "there exists an accepting path in the tree T whose path is consistent with A" (that is to say, in which every query response dictating an edge is consistent with A). To formalize this into the sentence we want, let r range over all possible polynomial length sequences of (q_j, r_j) (query, response) pairs (clearly each of which is polynomial length), and s range over all possible queries q_s (each of which is of course polynomial length since the original machine runs in polynomial time). Let $\bar{R}^A(x_1, ..., x_i, r, s, y)$ be the predicate "$r = (q_1, r_1), ..., (q_t, r_i)$" is an accepting branch in T, and if q_s appears as a query node along the branch r, then the associated edge (q_s, r_s) is consistent with A (that is to say, if $r_s = 1$, then $q_s \in A$, and if $r_s = 0$, then $q_s \notin A$). The first part, verifying the path, can be done in polynomial time deterministically without an oracle by simply simulating M with these oracle answers. The second part, verifying (q_s, r_s), can be done with a single oracle query. Clearly then we have

$$R^A(x_1, ..., x_i, y) \iff \exists^P r \forall^P s \bar{R}^A(x_1, ..., x_i, r, s, y)$$

completing the proof. $\square$

Note in the above proof that the choice of structuring our sentence as a $\exists\forall$ alternation was arbitrary - it could just as easily have been $\forall\exists$. Thus by writing the two quantifier sentence such that the first quantifier matches Q_i, we can merge those two quantifiers and accomplish the same task with only a single other alteration. Thus we actually have the stronger corollary:

Corollary 3.1. *Let*

$$\sigma(y) = \exists^P x_1 \forall^P x_2 \dots Q^P x_i R^A(x_1, ..., x_i, y)$$

be a sentence in $\Sigma_i^{P,A}$. *Then there is an equivalent sentence*

$$\bar{\sigma}(y) = \exists^P x_1 \forall^P x_2 \dots Q^P x_{i+1} R^A(x_1, ..., x_{i+2}, y)$$

in $\Sigma_{i+1}^{P,A}$ *where* $\bar{R}^A(x_1, ..., x_{i+1}, y)$ *is a predicate that can be computed by a Turing machine that makes at most one oracle query and runs in polynomial time.*

We are finally ready to describe the diagonalization process. Obviously to speaking of an enumeration over the '***PH*** machines' is a bit of a strange thing. Less strange is to enumerate over the '*bmPH* expressions'. These are expressions of the form above, which implicitly is defined via an enumeration over all polynomial time computations for the sake of R^A. These are of course all relativized over an oracle A that we will be constructing, initially the empty set. It only bears saying right now that these machines are oracle machines in their having of the ability to query strings in A. Polynomial time bounds for machine computations will also serve as size bounds for the quantification.

With all of that said, consider a ***PH*** expression, wlog a Σ^l one for some l and some polynomial bound k:

$$\exists^p x_1 \forall^p x_2 \dots Q^p x_l R^A(x_1, x_2, \dots, x_l, y)$$

Our immediate goal is to build a circuit which in some sense represents this expression. Initially, it is fairly obvious how to go about this. From left to right, begin with the first quantifier $\exists$. For this we create a sink OR gate. This OR gate has fanin equal to the total number of strings being quantified over. This number the sum of the number of strings of length j for each j from 1 to n^k

$$\sum_{j=1}^{n^k} 2^j = \frac{1 - 2^{n^k+1}}{1-2} = 2^{n^k+1} - 1 = O(2^{n^k})$$

Each of these wires will come from an AND gate representing the next quantifier, from each of these quantifiers will come $O(2^{n^k})$ more wires, and so on. After the final quantifier's fanin, the reader will note that the selection of any particular wire at the bottom of this subcircuit corresponds to a particular selection of $x_1, x_2, \dots, x_l$. These are not going to be inputs, but rather determine the structure of the circuits. Likewise we will also be thinking of the actual input y to R^A as fixed prior to the construction of the circuit.

At the bottom of this we want to address circuits for the computations $R^A(x_1, x_2, \dots, x_l, y)$. Again, the inputs to the computation are fixed at this point. What is going to be allowed to vary is the elements of the oracle A. Whether the machine accepts or rejects is determined then by a polynomial number of queries of polynomial length strings to the oracle. One can then see the computation R^A, for a fixed $x_1, x_2, \dots, x_l, y$ as a DNF circuit whose clauses correspond to the collections of strings which need to be in the oracle for the computation to return a 'true'. Since there are $O(2^{n^k})$ many strings of polynomial length, there are $O(2^{2^{n^k}})$ many polynomial sized collections of strings of polynomial length. We are trying to keep the size of this circuit relatively small though, and so this is where we resort to the above lemmas.

By the above lemmas, we can assume that the poly-time computations only need to query a single string from the oracle, on the condition that we add another quantifier, or in the context of our circuit, another level of depth. Since we are now assuming the computations only depend on a single oracle, we no longer need a DNF subcircuit at all. In fact, the result of R^A is equivalent to either a literal (in the case that it doesn't actually use the oracle at all) or to the atomic statement of whether or not a single string of polynomial length is in the oracle. *These* are the inputs to our circuit - bits representing whether or not specific strings of polynomial length are in the oracle. These $O(2^{n^k})$ input bits fan out, and either they feed directly into the first layer of quantifiers/gates (in the case of R^A being true, or their negations to (in the case of R^A being false).

We now start to slowly bring in a diagonalization framework. At stage i, we are going to diagonalize out of the i^{th} ***PH*** expression, by ensuring it fails to be equivalent to our test language on a specific input. That specific input will be 1^n for some carefully chosen $n(i)$. We will commit to only adding strings of length $n(i)$ for any stage i, and this sequence of n's will be increasing (very quickly). Because of this, when looking at the inputs to our circuit, many will have been fixed already from prior commitments. We are in particular with those inputs of length exactly n. The strings of length greater than n but less than n^k we will assume will never be added to the oracle, and so we can *assume that all computations associated with inputs which are length less than n or greater than n are fixed and bound as literals.* The final input size to our circuit at the i^{th} stage of the construction is thus $m := 2^n$. Each corresponds to a string of length n which we may or may not decide to drop into the oracle.

What is the size of this circuit as a function of m? As we noted, each gate has fanout $2^{n^k+1} - 1$, and we have $l + 1$ layers of them. We'll assume as we more or less have already been doing that rather than have NOT gates we will simply have 2×2^n many inputs, each one having a negated dual. Thus the final size of the circuit is

$$(2^{n^k+1} - 1)^{l+1} \leq (2^{n^k+1})^{l+1} = (2^{\log^k(m)+1})^{l+1} = O(2^{l \log^k(m)})$$

So this is a circuit on m inputs which is size quasi-polynomial in m. This size is the key, since we can ensure for large enough n that this circuit can't decide something specific - parity. We are now finally ready to describe the diagonalization in full.

At stage i, we will define n to be large enough that several conditions are true at once. First and foremost, we wish to ensure that parity on $m = 2^n$ inputs cant be computed by circuits of the size we have in front of us. Recalling the details of this, we have that for $m > n_{l+1}$, there are no depth $l + 1$ parity circuits of size less than or equal to $2^{\frac{1}{10}{}^{\frac{l+1}{l+2}} m^1 l}$. This amounts to picking n large enough that $\log(n) > n_{l+1}$, and this will be the first condition. Second, we need to be sure that m is big enough that for all $m' \geq m$, that $(2^{\log^k(m)+1})^{l+1} < 2^{\frac{1}{10}{}^{\frac{l+1}{l+2}} m^1 l}$. Certainly this can be done since root functions grow faster than polylogarithmic functions regardless of degree. Finally, we need to be sure that in the next stage of the construction, $n(i + 1)$ is large enough that we need not worry about accidentally changing our minds about any strings we committed to adding or not adding to the oracle at stage i. We already mentioned that this amounts to picking $n(i + 1) > n(i)^k$ (Keep in mind that really k itself is a function of i; it is a parameter of the i^{th} $\boldsymbol{PH}$ expression.) Thus at stage i, we define $n(i)$ to be an integer large enough to satisfy all three of these conditions.

We can now make the crucial observation: If n is chosen as described above, then the circuit described above cannot possibly compute the parity function on m inputs. What does this mean in terms of oracles? Each m-bit string which we could plug into the Boolean circuit corresponds to a set of strings of length n which we could choose to add or not add. *In other words, the number of parity of the m-bit strings corresponds to the evenness or oddness of the number of n-bit strings which we would add to the oracle.* Indeed, since the circuit can't decide parity, there must be either an m-bit string consisting of an even number of 1's such that the circuit output is 1, or one consisting of an odd number of 1's such that the circuit output is 0. Pick a particular one of these, without loss of generality assume its the first case. Then we are adding an odd number of strings to the oracle, and the circuit is outputting a 0. This circuit's output is equivalent to the $\boldsymbol{PH}$ expression we started with. Thus this $\boldsymbol{PH}$ expression can't be deciding the language L_A on 1^n - there are an odd number of strings of length n in the oracle, but this expression acts as if we added an even number of them!

This completely describes the diagonalization process. If we do this for all i, then we will have ensured that every $\boldsymbol{PH}$ expression fails to correctly assess whether 1^n is in L_A for some n. Thus the language $L_A \notin \boldsymbol{PH}^A$, and we have diagonalized out of $\boldsymbol{PH}^A$.

Theorem 3.1. *There exists an oracle A such that $\boldsymbol{PH}^A \neq \boldsymbol{PSPACE}^A$*

Proof. Everything above. □

4 Fourier Analysis

From here on we will view Boolean functions as functions $f : \{0,1\}^n \to \{-1,1\}$, and will see the domain of this function as the group $\mathbb{Z}_2^n$, i.e. the product of n many copies of the integers modulo 2. We are interested in seeing these functions as a special case of functions $f : \mathbb{Z}_2^n \to \mathbb{R}$. These functions clearly form a real inner product space under standard scalar multiplication and function addition, and with the inner product defined by

$$\langle f, g \rangle := \frac{1}{2^n} \sum_{x \in \mathbb{Z}_2^n} f(x)g(x)$$

We denote this space $L^2(\mathbb{Z}_2^n)$, because that's what it is.

Towards defining and understanding the Fourier transform over elements of this space, we need to consider the **characters** of the group $\mathbb{Z}_2^n$. In general, a character of a group is a homomorphism from the group to the unit circle on the complex plane S_1. Many things can and should be noted immediately about this topic, both in general and with regard to our particular group $\mathbb{Z}_2^n$. Firstly note that for any homomorphism ϕ from a group to the unit circle, it must be the case that $\phi(x)$ is always a root of unity, since the order of any element of a finite group is finite. In our particular case of $\mathbb{Z}_2^n$, every element has order either 1 or 2. Thus homomorphisms of the type we are concerned with must map every element to either 1 or -1. The consequence of this is that the characters of a $\mathbb{Z}_2^n$ are themselves elements of $L^2(\mathbb{Z}_2^n)$, and in fact are Boolean functions.

We will denote characters of a group with χ. Let us next observe that the characters of $\mathbb{Z}_2^n$ are always orthogonal (as are the characters over any abelian group). Let χ_a and χ_b be different characters. Since they're different, there exists an $x \in \mathbb{Z}_2^n$ such that $\chi_a(x) \neq \chi_b(x)$. Let $y \in \mathbb{Z}_2^n$ be arbitrary. Then

$$\chi_a(y) \langle \chi_a, \chi_b \rangle = \frac{1}{2^n} \sum_{x \in \mathbb{Z}_2^n} \chi_a(y)\chi_a(x)\chi_b(x) \tag{1}$$

$$= \frac{1}{2^n} \sum_{x \in \mathbb{Z}_2^n} \chi_a(yx)\chi_b(x) \tag{2}$$

Recall that for any y in a group, that $x \mapsto xy$ is always a bijection. So for any $x_1 \in \mathbb{Z}_2^n$, there is a unique x_2 so that $x_1 = x_2 y$, and subsequently x_2 is unique so that $y^{-1}x_1 = x_2$. The point is that for every term in this sum, there is a unique term in the sum

$$\sum_{x \in \mathbb{Z}_2^n} \chi_a(x)\chi_b(y^{-1}x)$$

which is identical. Also obviously $y^{-1} = y$ here, but I'm keeping it somewhat general enough that this argument could be repeated for any finite group. In any case we have then that

$$\chi_a(y) \langle \chi_a, \chi_b \rangle = \frac{1}{2^n} \sum_{x \in \mathbb{Z}_2^n} \chi_a(yx)\chi_b(x) = \frac{1}{2^n} \sum_{x \in \mathbb{Z}_2^n} \chi_a(x)\chi_b(y) chi_b(x) = \chi_b(y) \langle \chi_a, \chi_b \rangle$$

i.e. for any $y \in \mathbb{Z}_2^n$, we have that $\chi_a(y) \langle \chi_a, \chi_b \rangle = \chi_b(y) \langle chi_a, \chi_b \rangle$. But since there is a particular y on which we have said that $\chi_a(y) \neq \chi_b(y)$, it must follow that $\langle \chi_a, \chi_b \rangle = 0$.

By the obvious isomorphism between functions in $L^2(\mathbb{Z}_2^n)$ and 2^n dimensional vectors with entries in $\mathbb{R}$, it is clear that the dimension of this space is exactly 2^n. It turns out that we can define a collection of just that many characters on $\mathbb{Z}_2^n$ with the assistance of our parity function.

Definition 4.1. For any $S \subseteq \{1, 2, \ldots, n\}$, define the function χ_S via

$$\chi_S(x_1 x_2 \ldots x_n) = \begin{cases} 1 & \text{if } \sum_{i \in S} x_i \text{ is even} \\ -1 & \text{if } \sum_{i \in S} x_i \text{ is odd} \end{cases}$$

The sets S can be alternatively viewed as n bit strings where the i^{th} bit is a 0 or 1 depending on whether or not $i \in S$. If we see S this way the above definition of χ_S can be re-expressed as

$$\chi_S(x) = (-1)^{S \cdot x}$$

where $S \cdot x = \sum_{i=1}^n S_i x_i$. From here on we will use $\oplus$ to denote addition modulo 2. Note that for a string $x = x_1 x_2 \ldots x_n$, $\bigoplus_{i=1}^n x_i$ is precisely the parity of x.

Note that the sum in this definition is merely a way to describe taking the parity of the substring whose indexes are specified by S. Clearly these are different functions for each S. What requires confirmation is the following

Fact 4.1. *For each S, χ_S is a character of $\mathbb{Z}_2^n$.*

Proof. We need to show that for any two n bit strings x and y, that $\chi_S(x \oplus y) = \chi_S(x)\chi_S(y)$ where addition here refers to bitwise addition mod 2. This is much easier to see than it might initially seem. Note that the product $\chi_S(x)\chi_S(y)$ is 1 iff $\chi_S(x) = \chi_S(y)$, ie iff either x and y both have an even number of 1's or both have an odd number of 1's. Meanwhile

$$\chi_S(x+y) = \begin{cases} 1 & \text{if } \bigoplus_{i \in S}(x_i \oplus y_i) = 0 \\ -1 & \text{if } \bigoplus_{i \in S}(x_i \oplus y_i) = 1 \end{cases} \tag{3}$$

$$= \begin{cases} 1 & \text{if } \bigoplus_{i \in S} x_i + \bigoplus_{i \in S} y_i = 0 \\ -1 & \text{else} \end{cases} \tag{4}$$

$$\tag{5}$$

In particular we can note here that $\chi_S(x \oplus y)$ is 1 precisely when the total sum of the 1's from both strings is even. But this itself is precisely when either both have an even number of 1's or both have an odd number of 1's, which is the same condition for being 1 as that of the product. Thus these functions are the same. □

Since the characters of an abelian group G are always orthogonal in $L^2(G)$, there can only ever be as many of them as the dimension of this space. In this case, we have identified 2^n unique characters, and thus this defines all possible characters over $\mathbb{Z}_2^n$. Moreover

Definition 4.2. The characters over $\mathbb{Z}_2^n$ form an orthogonal basis for $L^2(\mathbb{Z}_2^n)$. Thus for any function $f \in L^2(\mathbb{Z}_2^n)$, we can express f in this basis via the formula

$$f(x) = \sum_{S \subseteq \{1,2,\ldots,n\}} \langle f, \chi_S \rangle \, \chi_S(x)$$

Note that $\hat{f}$, as a function of the S sets viewed as n-bit strings, is itself a function in $L^2(G)$ - a different one. We call the mapping $\mathcal{F}$ defined by $f \mapsto \hat{f}$ the **Fourier transform** of f. We also call the inner product

$$\hat{f}(S) := \langle f, \chi_S \rangle = \frac{1}{2^n} \sum_{i \in S} f(x)\chi_S(x)$$

The S^{th} **Fourier coefficient** of f. The **degree** of f is the cardinality of the largest set S such that $\hat{f}(S) \neq 0$.

The Fourier transform allows us to convert any Boolean function into a polynomial of sorts.

Definition 4.3. A real valued **multilinear polynomial** is a multivariable function $f(x_1 x_2 \ldots x_n)$ of the form

$$f(x_1 x_2 \ldots x_n) = \sum_{S \subseteq \{1,\ldots,n\}} a_S \prod_{i \in S} x_i$$

where a_S is a real number for each S.

It is clear that if a function has a multilinear polynomial representation, then that representation is unique, since it is defined by a coefficient a_S for each term, and these terms have no way of mingling or simplifying. To see how the Fourier transform implies a conversion algorithm for any function into a multilinear polynomial, note first that for any single x_i, we have

$$(-1)^{x_i} = 1 - 2x_i$$

Thus given a function $f : \mathbb{Z}_2^n \to \mathbb{R}$, we have

$$f(x) = \sum_{S \in \{0,1\}^n} \hat{f}(S)(-1)^{S \cdot x} \tag{6}$$

$$= \sum_{S \in \{0,1\}^n} \hat{f}(S) \prod_{i \in S}(1 - 2x_i) \tag{7}$$

As an example consider the simply 2-bit Boolean function representing equivalence. That is, $f(x_1x_2) = (x_1 \iff x_2)$. This function maps 00 and 11 to 1, and 01 and 10 to -1. This is the same table of inputs and outputs as $\chi_{1,2}$, and thus $f(x_1x_2) = \chi_{1,2}$. But then

$$\begin{aligned} \chi_{1,2}(x_1x_2) = (-1)^{x_1+x_2} &= (-1)^{x_1}(-1)^{x_2} && (8)\\ &= (1-2x_1)(1-2x_2) && (9)\\ &= 4x_1x_2 - 2x_1 - 2x_2 + 1 && (10) \end{aligned}$$

Thus

$$(x_1 \iff x_2) \approx 4x_1x_2 - 2x_1 - 2x_2 + 1$$

The degree here is clearly 2.

Fact 4.2. *The **degree** of a function $f : \mathbb{Z}_2^n \to \mathbb{R}$ is the highest number of variables in a term of f expressed as a multilinear polynomial. Equivalently, it is the cardality of the largest $S \subseteq \{1, \ldots, n\}$ such that $\hat{f}(S) \neq 0$.*

Proof. Clearly the degree of χ_S for any S is the cardinality of S. By inspection of the Fourier representation of a function, it becomes clear that the degree will be the highest cardinality S such that $\hat{f}(S) \neq 0$. $\square$

Lemma 4.1. *For a Boolean function f on n bits, consider the probability experiment of drawing an n bit string uniformally at random. Then for any $S \subseteq \{1, \ldots, n\}$,*

$$\hat{f}(S) = P\left(f(x) = \bigoplus_{i \in S} x_i\right) - P\left(f(x) \neq \bigoplus_{i \in S} x_i\right)$$

Proof. Note that $\hat{f}(S) = \frac{1}{2^n}\sum_{x \in \{0,1\}^n} f(x)(-1)^{S \cdot x}$. We have seen that $(-1)^{S \cdot x}$ is 1 when x has an even number of 1's out of those bits indexed by S, and -1 when it has an odd number. Since f is Boolean, each $f(x)$ is either 1 or -1. Thus the products in the sum are 1 precisely when either x has an even number of 1's and $f(x) = 1$, or when x has an odd number of 1's and $f(x) = -1$. Stated more simply, these product terms are 1 precisely when $f(x)$ agrees with the output of the parity function acting on those bits indexed by S, i.e. $f(x) = \bigoplus_{i \in S} x_i$. Similarly these product terms are -1 precisely when $f(x)$ disagrees with the parity of x over S. If we split the sum then into two sums, the first consisting of all 1's and then subtracting from that all -1's, then we have precisely the formula that we were trying to prove. $\square$

The switching lemma has implications for the degree of Boolean functions.

Lemma 4.2. *Suppose f is given by a CNF or DNF formula, where expressed as a Boolean circuit has bottom fanin at most t. Apply a random restriction ρ with parameter p. Then for any s, the probability that the degree of $f \upharpoonright \rho$ exceeds s is at most $(5pt)^s$ (for sufficiently small p?)*

Proof. Suppose that, after applying a random restriction with parameter p to f, that the switching lemma applies i.e. the CNF can be flipped to a DNF (wlog) of bottom fanin at most s, and such that each clause accepts a disjoint set of inputs. Consider an S such that $|S| > s$. We claim that $f \hat{\upharpoonright} \rho(S) = 0$. To see this, note that $|S| > s$ means that for each bottom level AND gate, there is a string index in S which is not one of that clause's inputs. This means that for any string accepted by the AND gate which is of even parity (relative to S), there is another accepted by it which is of odd parity (flip the missing bit in S). Likewise the other way. It follows that, relative to S, each AND gate accepts the same number of strings of even and odd parity. It follows that the overall DNF circuit itself accepts the same number of even and odd parity strings relative to S, since each bottom level gate accepts a disjoint set of inputs. But if this is the case then $P\left(f(x) = \bigoplus_{i \in S} x_i\right) = P\left(f(x) \neq \bigoplus_{i \in S} x_i\right)$, and so by the above lemma it follows that $f \hat{\upharpoonright} \rho(S) = 0$. A similar argument gives the same result for DNF circuits. $\square$

Lemma 4.3. *Let f be a Boolean function computed by a circuit of size M and depth d, and suppose ρ is a random restriction with parameter $p = \frac{1}{10^d s^{d-1}}$. Then the probability that the degree of $f \upharpoonright \rho$ is greater than s is at most $\frac{M}{2^s}$. Larger choices of p are sufficient for the same bound.*

Proof. First, we hit with a restriction of parameter $p = \frac{1}{10}$. After that we will hit with $d-1$ more restrictions of parameter $\frac{1}{10s}$. Upon hitting with the first restriction, we claim that with probability at least $1 - \frac{1}{2^s}$, we can assume that each bottom level gate has fanin at most s (and this is without using the switching lemma). To see this, consider an individual bottom level gate. Assume wlog that the bottom level gates are AND gates. We break the situation up into two cases.

For the first case, suppose that this bottom level AND gate has fanin more than $2s$. Since these are AND gates, it will be trivialized to 0 if any of it's inputs are restricted to 0. The probability that this happens to a particular input is of course the probability that it is not made a $*$ by ρ, times the conditional probability that it is made a 0 given this, i.e. $\frac{9}{10}\frac{1}{2} = \frac{9}{20}$. So $\frac{11}{20}$ is the probability that an individual input to the AND gate doesn't deletes the gate from consideration post-restriction. Therefore the probability that the gate isn't trivialized is at most $\frac{11}{20}^{2s} < \frac{1}{2^s}$. Since a gate being trivialized is equivalent to it's fanin being 0, and the probability that the fanin exceeds 0 is bigger than the probability that the fanin exceeds s, and thus the probability that the fanin exceeds s is also at most $\frac{1}{2^s}$.

For the second case, suppose that the bottom level AND gate has fanin less than or equal to $2s$. To have fanin exceeding s is of course equivalent to having the $\leq 2s$ many inputs assigned more than s many $*$'s. If $k \leq 2s$ is the fanin of the gate, the number of $*$'s assigned to the gate follows a binomial distribution, and the probability that the number exceeds s is

$$\sum_{i=s+1}^{k} \binom{k}{i} \left(\frac{1}{10}\right)^i \left(\frac{9}{10}\right)^{k-i} \leq \sum_{i=s+1}^{k} \binom{k}{i} \left(\frac{1}{10}\right)^i \tag{11}$$

$$\leq \sum_{i=s+1}^{2s} \binom{2s}{i} \left(\frac{1}{10}\right)^s \tag{12}$$

$$\leq \sum_{i=0}^{2n} \binom{2s}{i} \left(\frac{1}{10}\right)^s \tag{13}$$

$$= 2^{2s} \left(\frac{1}{10}\right)^s \tag{14}$$

$$= \left(\frac{4}{10}\right)^s < \frac{1}{2^s} \tag{15}$$

The takeaway from all of this is that, upon hitting our circuit with a random restriction with parameter $\frac{1}{10}$, the probability that any gate has fanin exceeding s is at most $\frac{1}{2^s}$, and so the probability that at least one of these gates fails to have fanin at most s is $\frac{m_1}{2^s}$, where m_1 is the number of bottom level gates.

Now we apply $d-2$ more restrictions each with parameter $p = \frac{1}{10s}$. Assuming that the previous restriction was successful in ensuring bottom fanin at most s, the probability that any bottom level DNF subcircuits can't be flipped into DNF circuits with the same maximal bottom fanin is at most $(5\frac{1}{10s}s)^s = \frac{1}{2^s}$. The probability that at least one of these bottom subcircuits fails to flip is at most $m_2\frac{1}{2^s}$, where m_2 is the number of level 2 gates.

Finally, after applying all of these restrictions, assuming no failures we will have a single CNF or DNF circuit of bottom fanin at most s. We hit with a final restriction of $p = \frac{1}{10s}$. At this point we refer to the previous lemma, which tells us that there is an at most $\frac{1}{2^s}$ chance of the final restricted function having degree exceeding s. The overall probability that the degree of the final circuit exceeds s is at most the sum of the probabilities that each individual stage of this process fails, i.e. it is at most $\sum_{i=1}^{d} m_i \frac{1}{2^s} = \frac{M}{2^s}$. □

We now begin to relate the degree of the function f to the degree of it's restrictions. Some notation is in order. We will denote $S \subseteq \{1, \ldots, n\}$ to be the set of $*$'s fixed by a random restriction, and so $L = S^c$ is the set of literals. There are many $|L|$ bit strings that could occur here with equal probability, since the literals are either 0 or 1. Thus we will use $r \in \{0,1\}^|L|$ to denote the possible literals which could occur in the situation that S is fixed. Finally we will let $f \restriction r$ denote a particular restriction of f.

Lemma 4.4. *For any $A \subseteq \{1, \ldots, n\}$, we have*

$$\hat{f}(A) = \frac{1}{2^{|L|}} \sum_{r \in \{0,1\}^{|L|}} \chi_{A \cap L}(r) \hat{f} \restriction r(A \cap S)$$

Proof. The key to making sense of this expression on the right is also the key to proving it, and that is to note the hat on the f. Recall that $\hat{f}(A) = \langle f, \chi_A \rangle = E(f(x)\chi_A(x))$, where E is the expected value. Expanding this rhs accordingly gives

$$\frac{1}{2^{|L|}} \sum_{r\in\{0,1\}^{|L|}} \chi_{A\cap L}(r) \left[\frac{1}{2^{|S|}} \sum_{x\in\{0,1\}^{|S|}} f \restriction r(x)\chi_{A\cap S}(x) \right] \tag{16}$$

$$= \frac{1}{2^{|L|}} \sum_{r\in\{0,1\}^{|L|}} \frac{1}{2^{|S|}} \sum_{r\in\{0,1\}^{|S|}} \chi_{A\cap L}(r)\chi_{A\cap S}(x)(f \restriction r)(x) \tag{17}$$

Consider the product of these two characters inside the double sum. With both r and x fixed, we have fixed a full length string $y \in \{0,1\}^n$. What this product is doing is taking the parity of the bits of y indexed by L (or at least those of A), then the parity of the bits indexed by S (again, specifically those in A), and multiplying them. If the parity of the first term is even and that of the second is odd, or vice versa, then the overall product is clearly -1. But we can also see in this situation that y has an even plus an odd number of bits *overall* in A, i.e. $\chi_A(y) = -1$. Similarly if both of these are 1, then y has an even *overall* parity in A, and thus $\chi_A(y) = 1$, again agreeing with the product. We've thus establishes that

$$\chi_{A\cap L}(r)\chi_{A\cap S}(x) = \chi_A(y)$$

Similarly, if we consider the term $(f \restriction r)(x)$, we see that we are simply plugging in some of the digits of y and calling it a restriction, then plugging in the rest of it. It is thus clear that $(f \restriction r)(x) = f(y)$. If the $\frac{1}{2^{|S|}}$ is pulled out and merged with the $\frac{1}{2^{|L|}}$, the result is clearly $\frac{1}{2^n}$. Finally, the double sum clearly ranges over all possible y, and we can re-express it as simply the sum over all $y \in \{0,1\}^n$. We have established that the above expression is equal to

$$\frac{1}{2^n} \sum_{y\in\{0,1\}^n} \chi_A(y)f(y) = \hat{f}(A)$$

□

Lemma 4.5. *Let $f : \{0,1\}^n \to \{-1,1\}$ a Boolean function and $S \subseteq \{1,\ldots,n\}$ (set of stars intuitively). For any $B \subseteq S$,*

$$\sum_{C\subseteq L} \hat{f}(B\cup C)^2 = \frac{1}{2^{|L|}} \sum_{r\in\{0,1\}^{|L|}} \hat{f} \restriction r(B)^2$$

Proof. By the previous lemma, and noting in advance that we have $(B\cup C)\cap L = C$ and $(B\cup C)\cap S = B$, we have

$$\sum_{C\subseteq L} \hat{f}(B\cup C)^2 = \sum_{C\subseteq L} \left[\frac{1}{2^{|L|}} \sum_{r\in\{0,1\}^{|L|}} \chi_{(B\cup C)\cap L}(r)\hat{f} \restriction r((B\cup C)\cap S) \right]^2 \tag{18}$$

$$= \sum_{C\subseteq L} \left[\frac{1}{2^{|L|}} \sum_{r\in\{0,1\}^{|L|}} \chi_C(r)\hat{f} \restriction r(B) \right]^2 \tag{19}$$

$$= \sum_{C\subseteq L} \left[\frac{1}{2^{|L|}} \sum_{r\in\{0,1\}^{|L|}} \chi_C(r)\hat{f} \restriction r(B) \right] \left[\frac{1}{2^{|L|}} \sum_{r\in\{0,1\}^{|L|}} \chi_C(r)\hat{f} \restriction r(B) \right] \tag{20}$$

$$= \frac{1}{2^{2|L|}} \sum_{r_1,r_2\in\{0,1\}^{|L|}} \hat{f} \restriction r_1(B)\hat{f} \restriction r_2(b) \left[\sum_{C\subseteq L} \chi_C(r_1)\chi_C(r_2) \right] \tag{21}$$

$$= \frac{1}{2^{2|L|}} \sum_{r_1,r_2\in\{0,1\}^{|L|}} \hat{f} \restriction r_1(B)\hat{f} \restriction r_2(b) \left[\sum_{C\subseteq L} \chi_C(r_1 \oplus r_2) \right] \tag{22}$$

Where the final substitution of $\chi_C(r_1)\chi_C(r_2)$ for $\chi_C(r_1 \oplus r_2)$ follows since χ_C is a homomorphism over $\mathbb{Z}_n^2$. In fact, we claim that this sum is $2^{|L|}$ if $r_1 = r_2$ and 0 otherwise. To show this, first consider the case that $r_1 = r_2$. Then $r_1 \oplus r_2 = 00\dots0$, and so $\chi_C(r_1 \oplus r_2) = 1$, and of course there are $2^{|L|}$ many of these 1's. The case $r_1 \neq r_2$ is a bit trickier, and requires some counting. Since they aren't equal, there is a set of indices E on which they differ. Of course, the result of χ_C depends on the C, and in fact we can say in advance that $\chi_C(r_1 \oplus r_2) = 1$ if $|C \cap E| = 0 mod 2$, and -1 if $|C \cap E| = 1 mod 2$. We can be sure of this since on any index where r_1 and r_2 are the same, the digit is guaranteed to be 0, and won't contribute to the sum. We can now observe that $\sum_{C \subseteq L} \chi_C(r_1 \oplus r_2)$ is equal to the number of subsets C such that $|C \cap E|$ is even *minus* the number of subsets C such that $|C \cap E|$ is odd. Fix the indices of C which don't intersect with E, i.e. fix $C - E$, noting that there are $2^{|L|-|E|}$ ways to do this. For each of these, C could have no elements from E, or just 1, or 2 or... up to $|E|$. Thus we have

$$\sum_{C \subseteq L} \chi_C(r_1 \oplus r_2) = 2^{|L|-|E|} \left[\sum_{\alpha \text{ even}} \binom{|E|}{\alpha} - \sum_{\alpha \text{ odd}} \binom{|E|}{\alpha} \right]$$

Here there is a useful little fact from combinatorics: for a fixed n, the sum of all even combinations minus the sum of all odd combinations (from a set of n objects) is always 0. To see this, observe

$$0 = (1-1)^n = \sum_{i=0}^{n} \binom{n}{i} (-1)^i = \sum_{i \text{ even}} \binom{n}{i} - \sum_{i \text{ odd}} \binom{n}{i}$$

Thus the term in brackets is 0, and we have shown that the sum itself thus must be 0. Returning with this to our original expression, we can see that the only nonzero terms of the overall sum will appear when $r_1 = r_2$, allowing us to replace the outermost double sum with a single sum over the strings in $\{0,1\}^{|L|}$, and the term in brackets with $2^{|L|}$. This leaves us with

$$\frac{1}{2^{2|L|}} \sum_{r_1, r_2 \in \{0,1\}^{|L|}} \hat{f} \restriction r_1(B) \hat{f} \restriction r_2(b) \left[\sum_{C \subseteq L} \chi_C(r_1 \oplus r_2) \right] = \frac{2^{|L|}}{2^{2|L|}} \sum_{r \in \{0,1\}^{|L|}} \hat{f} \restriction r(B)^2$$

Canceling the numerator with half the denominator completes the proof. ☐

Lemma 4.6. *Suppose we fix a set S of stars, for some restriction. We can then talk about the probability experiment of fixing the literals, where each index of L is equally likely to be a 0 or a 1. Also fix an integer k. I.e. we are sampling an $r \in \{0,1\}^{|L|}$ uniformally at random. Then*

$$\sum_{A \subseteq L, |A \cap S| > k} \hat{f}(A)^2 \leq P(deg(\hat{f} \restriction r) > k)$$

Proof. First we re-express the lhs as a double sum, and apply our above lemma to the inner of the two:

$$\begin{aligned} \sum_{A \subseteq L, |A \cap S| > k} \hat{f}(A)^2 &= \sum_{B \subseteq S, |B| > k} \sum_{D \subseteq L} \hat{f}(B \cup D)^2 \\ &= \sum_{B \subseteq S, |B| > k} \frac{1}{2^{|L|}} \sum_{r \in \{0,1\}^{|L|}} \left(\hat{f} \restriction r(B) \right)^2 \\ &= \sum_{r \in \{0,1\}^{|L|}} \frac{1}{2^{|L|}} \sum_{B \subseteq S, |B| > k} \left(\hat{f} \restriction r(B) \right)^2 \end{aligned}$$

Now observe that if r is such that the degree of $f \restriction r$ is less than or equal to k, then $\hat{f} \restriction r(B) = 0$ for any $|B| > k$. Furthermore, since $\hat{f} \restriction r$ is a Boolean function and has norm 1, it follows that if r is such that the degree of $f \restriction r$ is greater than k, then $\sum_{B\subseteq S,|B|>k} \left(\hat{f} \restriction r(B)\right)^2 \leq 1$. Thus

$$\begin{aligned}
\sum_{r\in\{0,1\}^{|L|}} \frac{1}{2^{|L|}} \sum_{B\subseteq S,|B|>k} \left(\hat{f} \restriction r(B)\right)^2 &\leq \sum_{r\in\{0,1\}^{|L|},deg(f\restriction r)>k} \frac{1}{2^{|L|}} && (23)\\
&= \frac{\text{Number of } r \in \{0,1\}^{|L|} \text{ such that } deg(f \restriction r) < k}{\text{Total number of such restrictions}} && (24)\\
&= P(deg(f \restriction r) > k) && (25)
\end{aligned}$$

□

One more technical lemma and we can prove the main result. This little fact is simply the result of basic ideas from probability, and will be the starting point to proving our main result.

Lemma 4.7 (Chernoff Bounds)**.** *Let $X = \sum_{i=1}^{n} X_i$ be a sum of independent Bernoulli random variables each with probability p_i. Then for any $a, t > 0$,*

$$P(X > a) \geq 1 - \exp\left(-\left(\sum_{i=1}^{n} p_i\right)(1-e^{-t}) + ta\right)$$

Proof. Recall Markov's Inequality: For any random variable A and any $a > 0$, $P(A \geq a) \leq \frac{E(A)}{a}$. Then for any $t > 0$,

$$\begin{aligned}
P(X > a) &= P(e^{-tX} < e^{-ta})\\
&= e^{-ta}e^{ta}P(e^{-tX} < e^{-ta})\\
&\geq e^{-ta}e^{ta}\left((1 - \frac{E(e^{-tX}}{e^{-ta}})\right) && (1)\\
&= 1 - e^{ta}E(e^{-t\sum X_i})\\
&= 1 - e^{ta}\prod_{i=1}^{n} E(e^{-tX_i}) && (2)\\
&= 1 - e^{ta}\prod[e^{-t}p_i + (1-p_i)] && (3)\\
&= 1 - e^{ta}\prod[1 - p_i(1-e^{-t})]\\
&\geq 1 - e^{ta}\prod e^{-p_i(1-e^{-t})} && (4)\\
&= 1 - e^{ta}e^{-(\sum p_i)(1-e^{-t})}
\end{aligned}$$

Here (1) follows from applying the complement rule to Markov's inequality, (2) comes from the independence of the X_i's, (3) comes from simply carrying out the expected value of each Bernoulli, and (4) comes from the fact that $1 + \alpha \leq e^{\alpha}$ for all $\alpha \in \mathbb{R}$. □

Lemma 4.8. *Let f be a Boolean function on n variables, t an integer, and $0 < p < 1$, with $pt > 8$. Consider the probability experiment of picking a subset $S \subseteq \{1,\ldots,n\}$ such that each variable appears in S independently with probability p. Then*

$$\sum_{A\subseteq\{1,\ldots,n\}} (\hat{f}(A))^2 \leq 2E\left(\sum_{|A|>t,|A\cap S|>\frac{pt}{2}} \hat{f}(A)^2\right)$$

Proof. Fix a particular A of size greater than t, and consider the random variable $|A \cap S|$. This can be seen as a sum of independent Bernoulli random variables X_i, where each X_i is 1 iff index i is in both S and A.

Clearly this probability is $p\frac{|A|}{n} \geq \frac{pt}{n}$. Then applying the Chernoff bound derived above with $a = \frac{pt}{2}$, we have

$$\begin{aligned} P(|S \cap A|) > \frac{pt}{2}) &\geq 1 - \exp\left(\sum_{i=1}^{n} -(pt)(1 - e^{-\alpha}) - \frac{\alpha pt}{2}\right) \\ &\geq 1 - \exp\left(-pt(1 - e^{-\alpha}) + \frac{\alpha pt}{2}\right) \\ &= 1 - \exp\left(-pt(1 - e^{-\alpha} - \frac{\alpha}{2})\right) \end{aligned}$$

Where $\alpha > 0$ is a parameter; we can pick anything we want for it, and the second line of this follows from the fact that since $p_i \geq \frac{pt}{n}$ for all i, $\sum p_i \geq pt$. The function $1 - e^{\alpha} - \frac{\alpha}{2}$ has a global maximum near $\alpha = 0.6$ of $0.15 > 0.125 = \frac{1}{8}$. Thus choosing this for α, we obtain the final bound that the probability of the size of $|S \cap A|$ exceeding $\frac{pt}{2}$ is at least $1 - e^{-\frac{pt}{8}}$.

But by hypothesis $-\frac{pt}{8} < -1$, meaning that $e^{-\frac{pt}{8}} < \frac{1}{e}$, i.e. $1 - e^{-\frac{pt}{8}} > 1 - \frac{1}{e} > \frac{1}{2}$. For each $A \subseteq \{1, \ldots, n\}$ with $|A| > t$, we can define the Bernoulli random variable $X_A(S)$ which is 1 if $|A \cap S| > \frac{pt}{2}$ and 0 otherwise, and observe that the right hand side is two times the sum of these random variables multiplied by the constant $\hat{f}(A)^2$. Further we note that by our Chernoff observations the expectation of each of these is greater than $\frac{1}{2}$. Using linearity of expectation we obtain

$$\begin{aligned} E\left(\sum_{A \subseteq \{1,\ldots,n\}, |A \cap S| > \frac{pt}{2}} \hat{f}(A)^2\right) &= E\left(\sum_{A \subseteq \{1,\ldots,n\}} \hat{f}(A)^2 X_A\right) \\ &= \sum_{A \subseteq \{1,\ldots,n\}} \hat{f}(A)^2 E(X_A) \\ &\geq \frac{1}{2} \sum_{A \subseteq \{1,\ldots,n\}} \hat{f}(A)^2 \end{aligned}$$

□

Theorem 4.1. *Let f be a Boolean function computed by a circuit of depth d, size M, and let t be an integer such that $\frac{1}{10} t^{\frac{1}{d}} > 8$. Then*

$$\sum_{A \subseteq \{1,\ldots,n\}, |A| > t} \hat{f}(A)^2 \leq (2M) 2^{-\frac{1}{20} t^{\frac{1}{d}}} \tag{26}$$

Proof. Fix $p = \frac{1}{10t^{\frac{d-1}{d}}}$, and $s = \frac{pt}{2} = \frac{1}{20t^{\frac{1}{d}}}$ (note this implies $t = \frac{sp}{2}$). Consider the two stage probability experiment of first sampling a set $S \subseteq \{1, \ldots, n\}$ where each index appears in S with probability p, and then sampling a string r from $L = S^c$ where each bit is 0 or 1 independently and with equal probability. Note this is equivalent to choosing a random restriction ρ with parameter p. By lemma 4.7 we have

$$\begin{aligned} \sum_{A \subseteq \{1,\ldots,n\}, |A| > t} \hat{f}(A)^2 &\leq 2E(\sum_{A \subseteq \{1,\ldots,n\}, |A \cap S| > \frac{pt}{2}} \hat{f}(A)^2 \\ &\leq 2E\left(P(deg(f \restriction r) > \frac{pt}{2} (= s)\right) \end{aligned}$$

We are now clearly close to a position to apply lemma 4.3. To ensure that we are allowed to use it, observe that by choice of p and s,

$$\begin{aligned} p = p^{\frac{d}{d}} &= \frac{1}{10(\frac{sp}{2})^{\frac{d-1}{d}}} = \frac{1}{10}\left(\frac{2}{sp}\right)^{\frac{d-1}{d}} \\ \implies p^{\frac{2d-1}{d}} &= \frac{1}{10}\left(\frac{2}{s}\right)^{\frac{d}{d-1}} \\ \implies p &= \frac{1}{10^{\frac{d}{2d-1}}}\left(\frac{2}{s}\right)^{\frac{d-1}{2d-1}} \geq \frac{2^{d-1}}{10^d s^{d-1}} > \frac{1}{10^d s^{d-1}} \end{aligned}$$

Where the second to last inequality follows from the fact that $s > \frac{pt}{2} > 4$, so that $\frac{2^{d-1}}{10^d s^{d-1}} = \frac{1}{10}\left(\frac{1}{20}\right)^{d-1} < 1$, ensuring that taking a $2d-1^{th}$ root produces a bigger number than not taking it. By lemma 4.3 then we have that $P(deg(f \restriction r) > s) \leq \frac{M}{2^s}$, a constant with respect to S. The expectation of the degree of f under the restriction is thus bounded below this constant, and we are left with

$$\sum_{A \subseteq \{1,\ldots,n\}, |A|>t} \hat{f}(A)^2 \leq \frac{2M}{2^s}$$

Noting again that $s = \frac{1}{20}t^{\frac{1}{d}}$ completes the proof. $\square$

5 Descriptive Complexity of Oracle Class Separating PH and PSPACE

First, we need to demonstrate an oracle relative to which PH is equal to PSPACE.

Lemma 5.1. *Let E be an $\boldsymbol{EXP}$-complete problem. Then $\boldsymbol{PH}^E = \boldsymbol{PSPACE}^E$.*

Proof. First, note that clearly $\boldsymbol{EXP} \subseteq \boldsymbol{PH}^E$, since $\boldsymbol{EXP} \subseteq \boldsymbol{P}^E$. Next we show that $\boldsymbol{PSPACE}^E \subseteq \boldsymbol{EXP}$, so that we will have

$$\boldsymbol{EXP} \subseteq \boldsymbol{PH}^E \subseteq \boldsymbol{PSPACE}^E \subseteq \boldsymbol{EXP}$$

which obviously completely the proof. To see this, consider a machine M^E operating in polynomial space, say time n^{k_1}. Overlooking the oracle queries for a moment, we know that we can simulate space n^{k_1} machines in time $O(C^{n^{k_1}})$ for some constant $C > 1$. Consider the machine which simulates M^E in this manner, and then, when M^E would normally query E, simply deterministically solves E in whatever time is required, say time $O(2^{n^{k_2}})$. Clearly this machine decides the same language as M^E, and furthermore it operates in worst case time $O(2^{cn^{k_1}+n^{k_2}} = O(2^{n^k})$ for some $k \in \omega$. Thus $L(M^E) \in \boldsymbol{EXP}$, demonstrating the claim. $\square$

As before, let us define the class $\boldsymbol{S}$ to be the set of oracles A relative to which $\boldsymbol{PH}^A \neq \boldsymbol{PSPACE}^A$.

Lemma 5.2. $\boldsymbol{S} \in \Pi^0_2$

Proof. For an oracle A, we define what we will call a $\boldsymbol{PSPACE}^A$-expression. These are of the form

$$\phi(x) = \exists^p_{x_1} \forall^p_{x_2} \ldots Q^p_{x_{n^k}} R^A(y,x)$$

where y is the concatenation of all strings of length x_i, and R^A is a relation relative to A corresponding to a Turing machine (we put no time or space bounds on it for now) computation. We will assume for simplicity that the same polynomial bound k is used for the strings being quantified over and the *number of alternations of quantifiers.* Enumerate over all such expressions (this can easily be done computably and in polynomial time). We define a particular language U for which membership of it in $\boldsymbol{PH}^A$ is equivalent to $\boldsymbol{PSPACE}^A = \boldsymbol{PH}^A$. From this it will follow that $\boldsymbol{S}^C$ is Σ^0_2, since expressing that a language is in the class $\boldsymbol{PH}^A$ will show itself to be easily expressible as a Σ^0_2 expression.

The language U is defined as follows. The language will be over the alphabet $\{0,1,p\}$ for some symbol p. For a pair of binary strings n, xp^l for some $l \in \omega$, $(n, xp^l) \in U$ iff $\phi_n(x)$, i.e. the n^{th} $\boldsymbol{PSPACE}^A$ expression "accepts" x, and the machine relation $R(y,x)$ associated with this expression halts on the appropriate y's appended with x in time $|xp^l|$. Thus the p's act as padding for the checking of runtimes longer than simply the length of x. We claim that this language is in $\boldsymbol{PSPACE}^A$. To show this note that that $R^A(y,x)$ is computable in polynomial time relative to A, since y, x is always a polynomial length input in $|x|$ and R^A can be simulated by a universal Turing machine with polynomial efficiency loss. Since polynomial time computations only use at worst polynomial space, we can easily maintain polynomial size stacks in order to check through every possible combination of x_i using polynomial space.

Suppose that $U \in \boldsymbol{PH}^A$. For simplicity we use the identity $\boldsymbol{AP}^A = \boldsymbol{PSPACE}^A$ and show that an arbitrary polynomial time alternating Turing machine decided a language that can also be decided in $\boldsymbol{PH}$. Let M^A be an alternating Turing machine which halts in polynomial time. The configuration graph of an alternating Turing machine can be seen as a polynomial depth circuit of alternating AND and OR gates, where the type for each non-leaf gate is determined by the type of the corresponding configuration. Likewise all configurations reachable in a polynomial number of steps can be represented as polynomial length strings. At the bottom of the graph we have acceptance or rejection leaves, which can be taken as literals. For a concatenation of polynomially many configuration strings y, let $R^A(y,x)$ be the relation which is true iff the path in the configuration graph outlined by y leads to a terminal accepting state. Clearly with the help of A this can be decided in time polynomial in $|x|$. It follows that we can construct a $\boldsymbol{PSPACE}^A$ expression ϕ such that $x \in L(M^A)$ iff $\phi(x)$. Since $U \in \boldsymbol{PH}^A$, there is for some $i \in \omega$, and some poly-time computable relation $S^A(y,x)$, an expression $\psi(n,x) = \exists^p_{x_1} \forall^p_{x_2} \ldots Q^p_{x_i} S^A(y,n,x)$ (y the concatenation of the quantified strings) such that $(n,x) \in U \iff \psi(n,x)$. Let m be the integer indexing the $\boldsymbol{PSPACE}^A$ expression ϕ. It is clear that $(m,x) \in U$ iff $\phi(x)$. Thus $\psi(m,x)$ is a $\boldsymbol{PH}^A$ expression for $L(M^A)$. Thus $\boldsymbol{PSPACE}^A \subseteq \boldsymbol{PH}^A$, as desired.

$\square$