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

Håstad's Switching Lemma and its Implications for Parity

Author: Atul Ganju

1 Introduction

In the following literature, we define a relationship between boolean functions and boolean circuits. There has been a lot of work in the field over the years and we hope to paint a picture of the general intuition for studying this relationship by going over some of the more fundamental results.

2 Basic definitions

To discuss boolean circuits we must first define boolean functions.

Definition 2.1 (Boolean Function)**.** *Consider the finite field $\mathbb{Z}_2^n$, then a boolean function (also referred to as a n-ary boolean function) is any function that takes the form $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$.*

In words, a boolean function is a function that maps bit strings of length n to either 0 or 1. Three trivial examples of such functions are logic gates: $\wedge$, $\vee$, and $\neg$.

Definition 2.2 (Basis Ω)**.** *A set of boolean functions.*

With this, we can define the boolean circuit.

Definition 2.3 (Boolean Circuit)**.** *A boolean circuit is a directed, acyclic graph such that for each vertex v in the graph, either (1) v has in-degree 0 and is labeled as an input which takes either a constant value of 0 or 1 or is a variable x_i for some $i \in \mathbb{N}$ or (2) v has in-degree $k > 0$ and is labeled as a gate which is a k-ary boolean function $g \in \Omega$ with an ordering on the incoming edges to v. Vertices with an out-degree of 0 are referred to as the labeled the circuit.*

The in-degree of a gate is its fanin and its out-degree is its fanout. The depth of a circuit is the longest path from an input to the output gate. The size of a circuit is the number of gates it contains.

A basis Ω is complete if every binary function can be computed by a circuit over Ω. The standard basis, denoted Ω_0, is the set $\{\wedge, \vee, \neg\}$ in which $\wedge$ and $\vee$ have fanin 2. Later we will show the standard basis is complete. Another example of a complete basis is Ω_{16}, the complete list of boolean functions on 2 variables (there are 16 such functions). For simplicity, when we refer to a boolean circuit below, we will assume it to be over the standard basis.

A particularly useful type of boolean circuit is the alternating circuit.

Definition 2.4 (Alternating Circuit)**.** *Boolean circuits where all $\wedge$ and $\vee$, gates are organized into alternating levels with edges only between adjacent levels. For circuits, the depth is the maximal length of a path from an input gate to the output gate.*

The two distinct kinds of alternating circuits of depth 2 are CNFs and DNFs.

Definition 2.5 (Disjunctive Normal Form)**.** *An s-DNF is an OR (disjunction) of clauses of width at most s, where each clause is an AND (conjunction) of literals. The following is an example of a*

2-DNF over 4 literals:

$$(x_1 \wedge x_3) \vee (x_2 \wedge x_4) \vee (x_3 \wedge x_4)$$

The size of the DNF is the number of clauses, and the width of the DNF is the maximum number of literals in a term.

Definition 2.6 (Conjunctive Normal Form)**.** *A t-CNF is an AND of clauses of width at most t, each of which is an OR of literals. The following is an example of a 3-CNF over 5 literals:*

$$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_5) \wedge (x_3 \vee x_4 \vee x_5)$$

Its size and width are defined in the same way.

Interestingly, one can convert any CNF to a DNF and vice versa by applying the results of the following two statements referred to as DeMorgan's Laws:

$$\neg(p_1 \vee \ldots \vee p_k) = (\neg p_1) \wedge \ldots \wedge (\neg p_k) \qquad \neg(p_1 \wedge \ldots \wedge p_k) = (\neg p_1) \vee \ldots \vee (\neg p_k)$$

Notice for any depth $d > 0$ there are exactly two types of alternating circuits. By repeatedly applying DeMorgan's Laws, one can see that any depth d alternating circuit of the first type can be converted to a depth d circuit of the second type and vice versa. Now, we have enough background to see the connection between boolean functions and boolean circuits.

3 Representing boolean functions as alternating circuits

The most intuitive relationship between boolean functions and boolean circuits is representing the first in terms of the second. Take $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ and denote the set of inputs for which f evaluates to 1 as $\mathbf{X}$. Then, for every $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbf{X}$ and $T_{\mathbf{x}} = \{i : x_i = 1\}$, we can write the following clause:

$$\mathbf{x} = \left(\bigwedge_{i \in T_{\mathbf{x}}} x_i \right) \wedge \left(\bigwedge_{i \notin T_{\mathbf{x}}} \neg x_i \right).$$

This gives us a boolean circuit of depth 1 for our fixed point $\mathbf{x}$ and allows us to write the following DNF which computes our function:

$$f := \bigvee_{\mathbf{x} \in \mathbf{X}} \left[\left(\bigwedge_{i \in T_{\mathbf{x}}} x_i \right) \wedge \left(\bigwedge_{i \notin T_{\mathbf{x}}} \neg x_i \right) \right].$$

This circuit representation of f has $n - 1$ gates per clause and 2^n clauses giving us $\mathcal{O}(n2^n)$ gates. Furthermore, as we stated before, since any DNF can be converted to a CNF using DeMorgan's Laws, we have shown that alternating circuits of depth 2 that contain an exponential number of gates have the power to compute any boolean function over n variables. Using more sophisticated techniques, one can optimize this result to give a tight bound on the size of the circuit needed to compute the most complex boolean function. What we have shown is a simple result that shows the relationship between boolean functions and boolean circuits. In the following sections, we delve deeper into this relationship.

4 Tight bounds on computing boolean functions

We just observed the power of depth 2 alternating circuits with polynomially bounded fanin. However the above construction, does not necessarily return the minimal circuit for a boolean function, just one that computes it. Let's define the set of boolean functions over n variables to be $\mathbf{B}_n$ and the set which contains the minimal fanin 2 circuits for every $b \in \mathbf{B}_n$ to be $\mathbf{C}_n$. Then, below we give a lower bound for the size of the most of the elements of $\mathbf{C}_n$.

Theorem 4.1 ([Mul56])**.** *Almost every boolean function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ requires fanin 2 circuits of size $\Omega(2^n/n)$.*

Proof. We start by proving the former. We will use a counting argument to show that the number of circuits of size $(1-\epsilon)\frac{2^n}{n}$ that compute a unique function is significantly less than the number of boolean functions over n variables in the limit where $n \to \infty$.

First notice that $|\mathbf{B}_n| = 2^{2^n}$ since a boolean function can map every element of its domain, the set of the binary strings of length n, to either 0 or 1. Therefore, we are showing that the number of circuits of size $(1-\epsilon)\frac{2^n}{n}$ that compute a unique function is much less than 2^{2^n}.

Suppose we wanted to count the number of circuits of size at most s that compute a unique function. We know that if a function is computable by a circuit of size less than s, by adding useless gates to the circuit we can increase its size to s. So the number of circuits of size at most s that compute a unique function is bounded above by the number of circuits of size exactly s. A circuit having s gates is defined by the following properties:

1. For each internal gate, its type and its two predecessor gates
2. The output gate

Now, if each gate can take input from either one of the n boolean variables or the output of some gate, there are at most $(n+s)^2$ ways to choose inputs for each gate (where we choose with replacement for the sake of simplicity). Thus, since as mentioned before we only consider the $\wedge, \vee$, and $\neg$ gates, we have that there are at most $3^s(n+s)^2 s$ different ways to assign the type and predecessor gates for each of the s gates in the circuit. Furthermore, since we can assume that every circuit computes a unique function (because if it did not we could delete all copies of it and rewire our circuit) we can permute the gates in $s!$ different ways. Finally, since we can assign any of the s gates to be the output, we have that, for sufficiently large n, the total number of circuits of size s is bounded above by:

$$\frac{s \cdot (3 \cdot (n+s)^2)^s}{s!} \leq \frac{s \cdot (3 \cdot (2s)^2)^s}{s!} = \frac{s \cdot (12 \cdot s^2)^s}{s!} \leq \frac{s \cdot 36^s \cdot s^{2s}}{s^s} = s \cdot (36 \cdot s)^s \leq (36 \cdot s)^{s+1}$$

where we get the third inequality using Stirling's bound $q! \geq q^q/e^q \leq q^q/3^q$. Finally, setting $s = (1-\epsilon)\frac{2^n}{n}$, we have:

$$\frac{s \cdot (3 \cdot (n+s)^2)^s}{s!} \leq (36 \cdot s)^{s+1} = \left(36 \cdot (1-\epsilon)\frac{2^n}{n}\right)^{(1-\epsilon)2^n/n+1} \leq (2^n)^{(1-\epsilon)2^n/n+1} = 2^{(1-\epsilon)2^n+n}$$

where for any ϵ, we can pick an n large enough such that this quantity is much less than 2^{2^n}, the total number of boolean functions on n variables. Therefore, almost every boolean function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ requires fanin 2 circuits of size $\Omega(2^n/n)$. □

This proof can be reformulated in the same way to show that the result holds for B_2 circuits. So, we have shown that most functions have large minimal circuits; however, this proof does not give an explicit construction of such a function. In fact, it is a long standing open problem to show that some function has a minimial circuit that is exponential in size. Currently, the best known lower bound on circuit size for an explicit function is only $5n$ [IM02].

Interesting enough, we also that the same bound is also an asymptotically tight upper bound for the largest element of $\mathbf{C}_n$.

Theorem 4.2 ([Mul56])**.** *Every boolean function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ can be computed by a fanin 2 circuit of size $\mathcal{O}(2^n/n)$*

Proof. Take an arbitrary boolean function f on n variables. First we recognize that we can find a circuit for f of size $\mathcal{O}(2^n)$ using the following recurrence:

$$f(x_1, \ldots, x_n) = (x_1 \wedge f(1, x_2, \ldots, x_n)) \vee (\overline{x_1} \wedge f(0, x_2, \ldots, x_n)).$$

This gives us a representation of f in terms of partial functions on fewer bits which are obtained by fixing the first few input bits. We can see this circuit has size $\mathcal{O}(2^n)$ since there exists a recurrence for a bound on the size of the circuit over n variables which we can solve:

$$\begin{aligned} s(n) \leq 2s(n-1) + 4 &\implies s(n) \leq 2^n(2 + 4/(2-1)) - 4/(2-1) = 6 \cdot 2^n - 4 \\ &\implies s(n) = \mathcal{O}(2^n) \end{aligned}$$

where we can make the first implication since we know the solution to recurrences of the form $a_n = ka_{n-1} + r$, are $a_n = k^n(a_0 + r/(k-1)) - r/(k-1)$.

To get the circuit of size $\mathcal{O}(2^n/n)$, we stop the recursion after fixing t variables. By stopping after t steps, we have that computing f is equivalent to computing $f(b_1, \ldots, b_t, x_{t+1}, \ldots, x_n)$ for all $(b_1, \ldots, b_t) \in \{0,1\}^t$. We now built a circuit C for f in the following way:

1. For every $b_1, \ldots, b_t \in \{0,1\}^t$, construct a circuit for $f(b_1, \ldots, b_t, x_{t+1}, \ldots, x_n)$ and denote them $C_1, \ldots, C_{2^t}$.
2. Combine the outputs of the C_i using the structure of the recurrence relation.

Then the size of C is equal to sum of the number of gates needed to compute all of the C_i and the number of gates needed to emulate the recursion. The former term can be bounded by product of the number of functions on $n-t$ variables, which is $2^{2^{n-t}}$, and the maximum size of a circuit needed to compute such a function which from above we know is $\mathcal{O}(2^{n-t})$. The latter comes from recognizing that using the analysis above we have $\mathcal{O}(2^t)$ as a bound on the number of gates t levels of the recursion will require. Therefore, we have:

$$\text{size}(C) = \mathcal{O}(2^{2^{n-t}} \cdot 2^{n-t} + 2^t)$$

Depending on what t is, our circuit for f has varying size. If we take $t = n$, then our circuit is very clearly of size $\mathcal{O}(2^n)$. However, if we take $t = n - (\log_2(n) - 1)$, we have:

$$\text{size}(C) = \mathcal{O}(2^{2^{n-t}} \cdot 2^{n-t} + 2^t) = \mathcal{O}(2^{2^{\log_2 n - 1}} \cdot 2^{\log_2 n - 1} + 2^{n - (\log_2 n - 1)}) = \mathcal{O}(2^n/n)$$

so we can construct a circuit of size $\mathcal{O}(2^n/n)$ for any boolean function f on n variables. □

Finally, since the circuit approximating f is a fanin 2 circuit the number of wires in the circuit is also $\mathcal{O}(2^n/n)$.

5 Constant depth circuits

Above we show that most boolean functions require large circuits. This result by itself is interesting; however, if we could show that there exists a function $f \in \mathbf{NP}$ which requires super-polynomial size circuits, that would imply $\mathbf{P} \neq \mathbf{NP}$. One logical way of trying to approach such a problem would be to think of a lower bound on the minimial circuit required to compute a specific function. Both of these questions have, after over 70 years of research, proven to be quite difficult to answer. As a result, theorists have thought of restricting their models of computation in an attempt to prove meaningful lower bounds.

5.1 Håstad's Switching Lemma

In this section, we discuss some lower bounds for the size of constant depth circuits that compute boolean functions. Since we restrict the depth of our circuit to a constant we allow for each gate to have unbounded fanin. In order to discuss the techniques used to achieve these lower bounds, we start with some definitions.

Definition 5.1 (Restriction ρ)**.** *Let $X = x_1, \dots, x_n$ be the input variables of a circuit C computing a function f on n boolean variables. A restriction ρ is a map from the input variables to the set $\{0, 1, *\}^n$. Take $stars(\rho)$ to be the number of x_i which ρ maps to $*$. Then, we write $f|\rho$ for the restricted function $\{0,1\}^{stars(\rho)} \to \{0,1\}$. Furthermore, a random restriction $\rho \in \mathcal{R}_p$ satisfies:*

- $\rho(x_i) = 0$ *with probability* $\frac{1-p}{2}$
- $\rho(x_i) = 1$ *with probability* $\frac{1-p}{2}$
- $\rho(x_i) = *$ *with probability* p

The set of restrictions we discuss have the probability of fixing a literal to 0 equal to the probability that it is fixed to 1; however, there is a more general class of restriction used in to prove circuit lower bounds in which the probability of 0 and 1 are specified by an additional parameter.

Definition 5.2 ($\mathcal{D}(f)$)**.** *The smallest $s \geq 0$ such that f can be expressed as a DNF formula that satisfies the following two properties:*

- *Each clause has size at most s;*
- *The clauses all accept disjoint sets of points (i.e. there is no $x \in \{0,1\}^n$ that satisfies multiple clauses).*

The general intuition for restricting a function f is to decomplexify the circuit computing it. Notice, that if even one variable in a DNF clause is fixed to 0, then the entire clause evaluates to 0. Similarly, if even one variable in a CNF clause is fixed to 1, then the entire clause evaluates to 1. If ρ fixes many variables, the number of clauses in the DNF that computes f will be small which implies the minimal circuit computing $f|\rho$ will be small. Finally, we can prove the main lemma of this section.

Lemma 5.3 (Håstad Switching Lemma)**.** *Let f be given by a s-DNF formula over n variables. Choose a random restriction ρ by setting every variable independently to $*$ with probability p, and to 0 and 1 each with probability $(p-1)/2$. Then:*

$$\Pr_{\rho \sim \mathcal{R}_p} [\mathcal{D}(f|\rho) > t] \leq (10ps)^t.$$

Proof. Take an arbitrary function f which can be written as a s-DNF formula over n variables. We can define some ordering $C_1, C_2, C_3 \dots$ on its terms. Now, suppose we hit f with the restriction ρ. We want to show the probability the $f|_\rho$ can't be rewritten as an depth t decision tree is bounded above by $(10ps)^t$. Say we had an algorithm for which we can construct a decision tree for $f|_\rho$. Then the probability that $f|_\rho$ can't be rewritten as an depth t decision tree is bounded above by the probability this algorithm returns a decision tree for $f|_\rho$ which has depth greater than t. Below we define such an algorithm $\mathcal{T}(f, \rho)$ for constructing a decision tree for $f|_\rho$. Then the remainder of our proof will show this construction rarely has depth greater than t.

$\mathcal{T}(f, \rho)$:

1. Define the subsequence $C_{i_1}, C_{i_2}, \dots$ to be the terms of f which ρ does not kill when applied.
2. Take C_{i_1}, which say reduces to R_1 over s_1 variables after ρ is applied, and make a depth s_1 decision tree over the variables in R_1 (querying them in order of indices).
3. Denote γ_1 to be the unique path in the decision tree of R_1 for which $C_{i_1}|_\rho$ evaluates to 1 and make it a leaf of the tree with value 1.
4. For all other paths ρ' in the decision tree for R_1, recursively add on $\mathcal{T}(f, (\rho, \rho'))$ where $f|_{\rho,\rho'}$ is the restriction of f to ρ and ρ' (if this restriction results in a constant function, add the constant as a leaf to the tree).

Note that when recursively adding on the tree $\mathcal{T}(f, (\rho, \rho'))$, each time the C_{i_1} depends on the first term which is not killed by the restrictions ρ and ρ'.

It remains to show that this construction rarely produces a large depth decision tree for $f|_\rho$. For our function f, denote the set of restrictions for which $\mathcal{T}(f, \rho)$ produces a decision tree of depth greater than t as $\mathcal{B}$ (and casually refer to these restrictions as "bad" restrictions). Then, we can prove our claim by bounding $|\mathcal{B}|/|\mathcal{R}_p|$, the proportion of restrictions which are bad, by the same $(10ps)^t$. Notice that the number of restrictions that fix σ variables is specifically $\binom{n}{\sigma}2^\sigma$ since we choose σ of the n variables to be fixed and set each one either to 0 or 1. Therefore, for some $p' < p$ and integer k small enough, if we show there exists an injection from $\mathcal{B}$ into $\mathcal{R}_{p'} \times \{0,1\}^k$ we will have shown what we are trying to prove. We specifically take $p' = p - t/n$ and $k = t \log s + 2t$. Then, our injection would imply the following result:

$$|\mathcal{B}| \leq |\mathcal{R}_{p-t/n} \times \{0,1\}^{t \log s + 2t}| = \binom{n}{np - t} 2^{n-np-t} \cdot (4s)^t.$$

So, the probability our restriction is bad is given by:

$$\frac{|\mathcal{B}|}{|\mathcal{R}_p|} = \frac{\binom{n}{np-t} 2^{n-np-t} \cdot (4s)^t}{\binom{n}{np} 2^{n-np}} \leq \left(\frac{np}{n - np - t}\right)^t (8s)^t \leq \left(\frac{p}{1-p}\right)^t (8s)^t \leq (10ps)^t$$

where in the last step we use the fact that $p \leq 1/5$. Now, it remains to show that there exists an injection from $\mathcal{B}$ to the set $\mathcal{R}_{p-t/n} \times \{0,1\}^{t \log s + 2t}$.

Such an injection is equivalent to the following: given some $\beta \in \mathcal{B}$, there being a unique member of $\mathcal{R}_{p-t/n} \times \{0,1\}^{t \log s + 2t}$ which we can map it to. Now, since our assumption is that β is a bad restriction, we are able to say that the depth of $\mathcal{T}(f, \beta)$ is greater than t and therefore can define π to be the lexicographically earliest path of depth exceeding t in this decision tree for $f|_\beta$ (where we trim it such that it fixes exactly t variables). Then, by the definition of π, even if we subject f to both the restrictions β and π, it will still not be a constant function.

As stated before, γ_1 is the setting to the s_1 variables in R_1 such that the first clause not killed by β outputs 1. We can define a counterpart to γ_1, call it π_1, to be the portion of π which sets these specific variables. Then assuming π_1 is not all of π, by further restriction f to both β and π_1, we kill C_{i_1} (and potentially many more clauses) and still have a function $f|_{\beta,\pi_1}$, over only $np - s_1$ free variables, which is not constant. We can then repeat this process. First, we find the earliest term in our ordering that is not killed by both β and π_1 and refer to its restriction under β and π_1 as R_2 where say it has s_2 free variables. Then, if we define γ_2 to be the setting to the variables in R_2 such that the first clause not killed by β outputs 1, we can define π_2 to be the part of π which sets the variables in R_2. Finally, again assuming that π_2 is not all of π, we can subject f to the ρ, π_1, and π_2, and still have a nonconstant function. Specifically, $f|_{\beta,\pi_1,\pi_2}$ is a nonconstant function over $np - s_1 - s_2$ variables. This process we can repeat until π_l gets the last bit of π. At this point, we can trim γ_l to just set the variables π_l sets (note $f_{\beta,pi_1,\ldots,\pi_{l-1},\gamma_l}$ is not constant but γ_l is still the unique assignment to the variables in R_l which satisfies them). Note that the combination of the restrictions $\beta, \gamma_1, \ldots, \gamma_l$, call it Σ is a member of the set the set $\mathcal{R}_{p-t/n}$.

We will use show that we can reverse engineer our β from the Σ and our function f with the help of $t \log s + 2t$ bits of information. In doing so, we will have shown that each β maps to a unique element in the set $\mathcal{R}_{p-t/n} \times \{0,1\}^{t \log s + 2t}$ which implies our injection. We reverse engineer β in the following way. Since an ordering was defined on the clauses of our DNF, we can pinpoint exactly which clause γ_1 applies to by finding the first term in the ordering which Σ fixes to 1. We can then encode the locations of the originally free variables in C_{i_1} using $s_1(\log s)$ bits. One can use encode the way π_1 set those bits using s_1 variables (and therefore know what the next clause which was unfixed by β was). One can encode the number of fixed variables in C_{i_1} with $\log s_1$ bits. Using the above facts, one can flip the s_1 unfixed bits in Σ to their setting in π_1 and look for the next clause which evaluates to 1 and repeat the process till we convert Σ to π. Then, by unfixing the variables we changed in Σ to get π, we have found our bad restriction β. It is not hard to see this iterative procedure will then require an encoding which uses $\sum_{j=1}^{l} s_j(\log s) + s_j + \log s_j \leq \sum_{j=1}^{l} s_j(\log s) + 2s_j = t(\log s) + 2t$ bits. Therefore, we have that β maps to a unique element in the set $\mathcal{R}_{p-t/n} \times \{0,1\}^{t \log s + 2t}$ which implies our injection. □

Informally, the switching says that if you have a DNF f with small width, s and hit it with a random restriction ρ such that there are very few remaining free variables, then with high probability $f|\rho$ has a small decision tree. Now, since a depth t decision tree can be imagined as a t-CNF in the trivial way, we can re-imagine the switching lemma in terms of CNFs in the following way: if you have a DNF f with small width, and hit it with a random restriction ρ such that there are very few remaining free variables, then with high probability $f|\rho$ can be written as a small width CNF.

It is not hard to see that the switching lemma can be rewritten with DNF switched with CNF and vice versa, where this proof is left to the reader.

5.2 Implications of the Switching Lemma

We will now see a classic application of Håstad's Switching Lemma: an asymptotically tight lower bound on computing the parity function with a depth d circuit. But first, we need to define parity and prove some basic results.

Definition 5.4 (Parity of n variables)**.** *The parity function $\oplus_n : \{0,1\}^n \to \{0,1\}$ maps binary strings with an odd number of* 1*s to* 1 *and an even number of* 1*s to* 0*. Equivalently,*

$$\oplus_n(x_1, \ldots, x_n) = x_1 \oplus \cdots \oplus x_n.$$

In this section, we will in fact show that parity over n variables, for n sufficiently large, is "hard" to compute with a constant depth circuit. Moreover we show that we have a tight lower bound on the size of a circuit computing parity over n variables.

Claim 5.5. *We can convert any depth d circuit of size S into one that satisfies the following properties:*

- *All $\neg$ gates are in the bottom layer which contains literals $x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}$.*
- *All gates at depth $s < d$ of the circuit, where $s \in \mathbb{N}$ are the same type of gate.*
- *The inputs of any gate at the ith layer are outputs of gates in the $(i+1)$th layer.*

by only increasing the size of the circuit by a factor of d. A circuit satisfying these properties is said to be in standard form.

Proof. Take an arbitrary circuit of depth d and size S. We can use De Morgan's Laws to force the $\neg$ gates to be in the bottom layer. We can ensure every layer contains the same type gate by combining gates of the same type in adjacent layers. Finally, we can ensure every gate only take input from the layer below it by adding $\wedge$ and $\vee$ gates with fanin 1. Finally, since the first two transformations don't add gates to the circuit and the last can only increase the size of the circuit by a multiplicative factor of d, we have that the size of our circuit is at most dS. □

Notice, standard form really converts any circuit of depth d into an alternating circuit of depth d. This will allow us to apply the switching lemma to our circuit.

Claim 5.6. *If a DNF or CNF computes parity of n variables, then:*

- *Each term includes all n variables*
- *There are at least 2^{n-1} terms*

which implies that a depth 2 circuit that computes parity must be at least of size 2^{n-1}

Proof. Without loss of generality we prove this lemma for CNFs, since the proof for DNF uses nearly identical logic. Take some CNF C that computes the parity of n variables.

- First we show that every clause of C must contain all n variables. Suppose, for the sake of contradiction C has some clause c^* that does not contain one of the literals namely x_i. Then, when all inputs to c^* are 0, our c^* outputs 0 and since C computes parity, we have that C outputs 0. Now, if we just change the value of x_i to 1, then c^* still outputs 0 implying C outputs 0; however, since we only changed one variable the parity is now 1, a contradiction.
- Now, we show that there are at least 2^{n-1} terms in our CNF. The parity function must output 0 if and only if $\sum_i x_i \equiv 0 \pmod 2$. This is true for 2^{n-1} configurations of the x_i. Since we just established every clause has n terms and it is the OR of n literals, a clause will output 0 on exactly one configuration of the x_i. Therefore, there must be 2^{n-1} clauses in our CNF.

Finally, since there are at least n clauses in our CNF of size 2^{n-1}, the size of our CNF which computes parity must at least 2^{n-1} □

Now, we can tackle Håstad's asymptotically tight bound on the minimal circuit of constant depth d that computes parity where this bound is tight up to a constant factor in the exponent. The general intuition is that if there existed a "small" depth k circuit which computes parity over n

variables, we could repeatedly apply Håstad's switching lemma to its standard form analog till we have a "small" depth 2 circuit which computes parity over a large enough subset of the $n$variables such that, using claim 2, we have a contradiction.

Theorem 5.7 ([Hås86]). *For any $d \geq 2$, the minimal circuit of depth d which computes parity on n variables has size at least $S \geq 2^{cn^{1/(d-1)}}$ for $c = 1/21$.*

Proof. Suppose there existed a circuit C of depth d and size $S < 2^{cn^{1/(d-1)}}$, for $c = 1/21$, which computes parity on n variables. We can start by making the assumption that $d > 2$ since, by claim 2, there does not exist a circuit of depth 2 which is smaller than S and computes parity. From above, we know that a depth d circuit can be transformed into one in standard form with a multiplicative d blowup in size. Therefore we can assume our circuit to be in standard form, since we can account for it by marginally increasing the constant c in our bound at the end.

Now, without loss of generality assume the gates on the bottom level to be $\vee$ gates (a symmetric argument applies for the case in which they are $\wedge$ gates). We think of each such gate as a 1-DNF formula where the inside clauses are the inputs themselves. By applying the switching lemma with $p = 1/20, s = \log S, t = 1$, we know that the probability a particular DNF fails, under restriction, to be transformed into a $(\log S)$-CNF is at most:

$$(10pt)^{\log S} = \left(10 \cdot \frac{1}{20}\right)^{\log S} = 2^{-\log S} = \frac{1}{S}.$$

Now, let's define E_1 to be the event that every 1-DNF formula could be converted to a $(\log S)$-CNF after applying some restriction. We know that there would not be S gates in level d, otherwise C would have depth 1, and its easy to see that depth 1 circuits can't compute parity. Therefore, C must have $T < S$ gates depth d. If we define the event $E_{1,i}$ for $i \in \{1, \ldots, T\}$ to be the event that the i-th DNF at depth d of C is killed by ρ, we can show that there exists a non-zero probability the E_1 occurs using the union bound:

$$\Pr_{\rho \in \mathcal{R}_p}[E_1] = 1 - \Pr_{\rho}[\overline{E_1}] \geq 1 - \sum_{i=1}^{T} \Pr_{\rho}[\overline{E_{1,i}}] = 1 - \sum_{i=1}^{T} \frac{1}{S} > 1 - \sum_{i=1}^{S} \frac{1}{S} = 0 \implies \Pr_{\rho \in \mathcal{R}_p}[E_1] > 0.$$

Therefore there exists some restriction ρ_1 for which, we can convert every 1-DNF to a $(\log S)$-CNF. Finally, we know that we can merge the levels $d-1$ and $d-2$ of C since they contain the same gate type and thus, we can reduce the depth of C to $d-1$.

Now, we have a circuit of depth d such that each bottom fanin is $\log S$ and there are at most S in the bottom level, and the circuit computes parity on at least $n/20$ variables. We can now apply the switching lemma with $p = 1/(20 \log S), s = \log S, t = \log S$ to, using nearly identical logic, collapse level $d-1$ and $d-2$ into one level which computes parity of $n/(20 \cdot (20 \log S))$. With this, every gate at the bottom level has fanin at most $\log S$ and there are at most S gates from level $d-2$ to level 1. Repeatedly applying the switching lemma $d-3$ more times, we end with a circuit of depth 2 with bottom fanin of at most $\log S$ which computes parity of $n/(20 \cdot (20 \log S)^{d-2})$ literals. Finally, by our second claim, we have:

$$S \geq 2^{n/(20(20\log S)^{d-2})} \implies S \geq 2^{c' n^{1/(d-1)}}$$

where $c' = 1/20$. Finally, since $c' > c$ we have the strict bound $S > 2^{cn^{1/(d-1)}}$. Finally, since our bound is strict, we can account for our earlier transformation from a arbitrary circuit to one that is in standard form by taking n sufficiently large. So, we have a contradiction. Therefore, we have proven parity of n variables can't be computed by a circuit of size $S < 2^{cn^{1/(d-1)}}$ for $c = 1/21$. $\square$

Notice that this theorem implies that if we wanted a polynomial sized circuit for parity, then it is necessary that $d \geq \Omega(\log n/(\log\log n))$. This proof follows the following general outline:

1. Prove that a hard function f has a large minimal depth 2 circuit
2. Use switching lemma to prove that a small depth d circuits which computes a function g can be converted to depth $d-1$ circuits which computes f (where f need not equal g).

This is a taste of the meaningful lower bounds that we can get from applying Håstad's Switching Lemma. Below we simply give a several more implications of the switching lemma and its variants.

Theorem 5.8 (Hås86)**.** *Majority requires a size $S = 2^{c^{d/(d-1)}n^{1/(d-1)}}$, where $c = 1/10$, and depth d circuit for sufficiently large n.*

Theorem 5.9 (Hås86)**.** *There are functions f_k^m which have linear size circuits of depth k but require exponential size circuits when their depth is restricted to $k-1$.*

Finally, it is important to note that aside from the explicit application of this theorem to prove lower bounds, Håstad's switching lemma is important for introducing a novel entropy argument as a method of proving circuit lower bounds.

Here, we chose to focus on constant depth circuit lower bounds; however, there has also been a lot of work done on the study of monotone circuits, linear circuits with logarithmic depth, etc. It is hoped that progress on the computational complexity of functions when the model of computation are circuits with such restrictions will hopefully translate to results for the hardness of functions when modeled with unrestricted circuits. As stated before, if we can show there exists a function $f \in \mathbf{NP}$ such that f can't be computed by polynomial sized circuits then we know $\mathbf{P} \neq \mathbf{NP}$.