Title
A satisfiability algorithm for constant depth boolean circuits with unbounded fan-in gates

Author
Matthews, William Grant

Publication Date
2011

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, SAN DIEGO

A Satisfiability Algorithm for Constant Depth Boolean Circuits with Unbounded Fan-In Gates

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

William G. Matthews

Committee in charge:

Professor Russell Impagliazzo, Co-Chair
Professor Ramamohan Paturi, Co-Chair
Professor Samuel R. Buss
Professor Daniele Micciancio
Professor Jacques Verstraete

2011

The dissertation of William G. Matthews is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____
Co-Chair

_____
Co-Chair

University of California, San Diego

2011

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGEMENTS

I would like to thank my advisors Russell Impagliazzo and Ramamohan Paturi for endless ideas, advice, support, discussions, and quite often counter examples to some of my overly ambitious ideas.

I am very grateful to my parents, John and Margaret Matthews, my siblings Michael and Elizabeth, and to Chessa Scullin for their support and encouragement.

Chapters 1–4 in part, and Chapters 5 and 7 in full are based on joint work with Russell Impagliazzo and Ramamohan Paturi, submitted to the 2012 ACM-SIAM Symposium on Discrete Algorithms. Chapter 6 is joint work with Ramamohan Paturi and appeared in *SAT*, 369-374, Lecture Notes in Computer Science 6175, 2010.

VITA

2004	B. S. in Computer Science, University of New Mexico
2005-2011	Graduate Research & Teaching Assistant, University of California, San Diego
2011	Ph. D. in Computer Science, University of California, San Diego

PUBLICATIONS

R. Impagliazzo, W. Matthews, R. Paturi, "A Satisfiability Algorithm for $\mathbf{AC}^0$", submitted, 2011.

W. Matthews, R. Paturi, "Uniquely Satisfiable k-SAT Instances with Almost Minimal Occurrences of Each Variable", *SAT*, 369-374, LNCS 6175, 2010

S. Arora, R. Impagliazzo, W. Matthews, and D. Steurer, "Improved Algorithms for Unique Games via Divide and Conquer", *ECCC*, Report TR10-041, 2010

ABSTRACT OF THE DISSERTATION

A Satisfiability Algorithm for Constant Depth Boolean Circuits with Unbounded Fan-In Gates

by

William G. Matthews

Doctor of Philosophy in Computer Science

University of California, San Diego, 2011

Professor Russell Impagliazzo, Co-Chair
Professor Ramamohan Paturi, Co-Chair

We consider the problem of efficiently enumerating the satisfying assignments to $\mathbf{AC}^0$ circuits. $\mathbf{AC}^0$ circuits are Boolean circuits with n inputs and their negations, one output, $m = poly(n)$ total gates, and constant depth, and consist of unbounded fan-in AND and OR gates. The primary technical tool we use is a new algorithmic approach for efficiently simplifying restricted classes of circuits. This approach is based on a new extended version of Håstad's Switching Lemma.

As the main result, we present a Las Vegas algorithm which takes an $\mathbf{AC}^0$ circuit as input and outputs a set of restrictions (assignments to subsets of the inputs) which partition $\{0,1\}^n$ such that under each restriction the out-

put of the circuit is constant. With high probability, the algorithm runs in time $poly(m, n) \cdot 2^{n(1-\mu)}$ and outputs at most $2^{n(1-\mu)}$ restrictions, where $\mu = 1/O(\lg \frac{m}{n} + d \lg d)^{d-1}$. This is optimal up to the constants in the big-O for enumerating solutions with restrictions. This also implies the best known algorithm for $\mathbf{AC}^0$ circuit satisfiability and for counting satisfying assignments.

Using similar techniques, we also give an algorithm for enumerating the solutions to a k-CNF, but where $\mu = 1/O(k)$. Previously, algorithms with similar savings μ were known for finding a single satisfying assignment to a k-CNF, but not for counting or enumerating satisfying assignments.

These results have some interesting applications to circuit lower bounds. We prove a new bound on the correlation of $\mathbf{AC}^0$ circuits with parity which is optimal up to constants, and show how several classic $\mathbf{AC}^0$ circuit lower bounds follow straightforwardly from our algorithm. Then, we use a powerful theorem due to Williams to show how a minor improvement in the running time for finding a single satisfying assignment for an $\mathbf{AC}^0$ circuit would imply $\mathbf{NEXP} \not\subseteq \mathbf{NC}^1$.

Chapter 1

Introduction

Circuit satisfiability, deciding whether a Boolean circuit has an assignment to its inputs where it evaluates to true, is essentially the original **NP**-complete problem [Coo71, Lev73], and in many ways, the canonical **NP**-complete problem. It is particularly important both theoretically and pragmatically since search problems reduce to circuit satisfiability without increasing the size of the search space. In contrast, reductions to special cases such as k-SAT often increases the number of variables dramatically. Despite this downside, k-SAT, and to a lesser extent CNF-SAT, have many algorithms which have been analyzed theoretically and implemented practically with astounding success. On the other hand, fairly little is known either in theory or in practice about the difficulty of satisfiability problems for circuits even slightly more general than CNFs, much less for arbitrary circuits.

This raises several natural questions: For which classes of circuits are there non-trivial (faster than enumerating the entire search space) algorithms for satisfiability? How does the complexity of satisfiability algorithms change as increasingly general classes of circuits are considered? Formally, for an algorithm that decides satisfiability for circuits in a class $\mathcal{C}$, we express its worst-case running time as $|C| \cdot poly(n) \cdot 2^{n(1-\mu)}$ where $C \in \mathcal{C}$ is a circuit with n inputs. We say that μ is the savings over exhaustive search and write it in terms of n and the parameters of the class $\mathcal{C}$, for example, the ratio of gates or wires to the number of inputs, and depth.

When can we exploit the structural properties of $\mathcal{C}$ to obtain savings over

exhaustive search? What are the best savings for various circuit classes? How is the expressive power of a circuit class related to the amount of savings for its satisfiability problem?

We also consider many of the same questions with respect to algorithms for the canonical #**P**-complete problems of counting the number of satisfying assignments for the same classes of circuits. Like in the satisfiability case, #k-SAT has been studied to some extent both theoretically and empirically whereas very little is know about #CIRCUIT-SAT for more general classes of circuits.

Both circuit satisfiability and counting satisfying assignments have numerous practical applications such as planning, model-checking, and Bayesian reasoning. Most of the effort has been concentrated on CNFs (and in particular k-CNFs), and for some very structured families of problems, state-of-the-art programs can handle CNFs with hundreds of thousands of variables and millions of clauses [MZ09]. Circuit satisfiability, rather than just CNF-SAT, is important because these underlying problems (model-checking, etc.) reduce to circuit satisfiability without increasing the search space, in stark contrast to the reductions to CNFs. Less is known about satisfiability for general circuits, but there have been some promising empirical results taking advantage of both the added structure given by circuits and the reduced search space [TBW04, WLLH07].

We approach these problems from a theoretical rather than empirical point of view. The main result we show is an algorithm which enumerates satisfying assignments to $\mathbf{AC}^0$ circuits. $\mathbf{AC}^0$ is the class of constant depth circuits consisting of unbounded fan-in AND and OR gates. Since circuits may have $\Omega\left(2^n\right)$ satisfying assignments, we construct a set of restrictions which partition $\{0,1\}^n$ such that under each restriction the value of the circuit is constant. It turns out that partitioning the input space this way gives an efficient representation for all the satisfying (and unsatisfying) assignments. The savings of this algorithm are the best known, and naturally generalize the savings of many of the best known satisfiability algorithms for more restricted classes of circuits, and in addition solve the corresponding counting problems with the same savings. For CNF-SATs and k-SATs our algorithms are the best known for counting satisfying assignments, and

Table 1.1: Some of the best known satisfiability and counting results. Throughout the table, n denotes the number of variables, m denotes the number of gates or clauses, $c = m/n$ is the ratio of the size of the circuit to the number of inputs, k bounds the fan-in of bottom level gates, and d denotes the depth of the circuit.

Problem	Savings	Source
3-SAT	0.614	[PPSZ05, Her11]
k-SAT	$1.227/(k-1)$	[PPSZ05]
k-SAT	$1/O(k)$	[PPSZ05, PPZ99, Sch99, LP11], Our algorithm [IMP11]
#2-SAT	0.671	[FK07]
#3-SAT	0.2569	[DJW05]
#k-SAT	$1/O\left(2^k\right)$	[LPI01]
#k-SAT	$1/O(k)$	Our algorithm [IMP11]
CNF-SAT	$1/O(\lg m)$	[Sch05]
CNF-SAT	$1/O(\lg c)$	[CIP06]
#CNF-SAT	$1/O(\lg c)$	Our algorithm [IMP11]
Boolean formulas, satisfiability and counting	$1/poly(c)$	[San10]
$\mathbf{AC}^0$ satisfiability	$1/O(c^{2^{d-2}-1}\lg^{3\cdot 2^{d-2}-2} c)$	[CIP09]
$\mathbf{AC}^0$, satisfiability and counting	$1/O(\lg c + d\lg d)^{d-1}$	Our algorithm [IMP11]
$\mathbf{ACC}^0$ satisfiability	$\Theta\left(\frac{n^{2^{-\Theta(d)}-\lg m}}{n}\right)$	[Wil11]

have similar asymptotic savings as the best known algorithms for satisfiability.

Often, the same ideas which are used to give improved satisfiability algorithms can also be used to prove circuit lower bounds. In our case, this connection holds in both directions: We used ideas from circuit lower bounds to design our algorithms, and we used our algorithm as a tool to reprove many classic $\mathbf{AC}^0$ lower bounds and give a new, essentially optimal, lower bound on the correlation of $\mathbf{AC}^0$ circuits with parity.

Table 1.1 summarizes our algorithmic results in comparison to some of the previous and best known results. In Section 1.1 we give more details on the previous results, and in Section 1.2 we formalize our main theorems.

1.1 Previous Work

Among the satisfiability problems for various circuit classes, the k-SAT problem has attracted the attention of several researchers since Monien and Speckenmeyer [MS85] first showed that it can be computed in less than 2^n time in the worst-case, that is, in time $|F|2^{n(1-\mu_k)}$ for $\mu_k > 0$ where F is a k-CNF. Researchers have obtained a series of improvements for μ_k, with particular attention to the case $k = 3$. Currently, the best known savings for 3-SAT is about 0.614 for randomized algorithms [PPSZ05, Her11] and about 0.4157 for deterministic algorithms [Sch99, MS11]. The best known μ_k is of the form $\Theta(1/k)$ [PPZ99, Sch99]. For counting the solutions to a k-CNF ($\#k$-SAT), less is known. [FK07, DJW05] give the best known algorithms for #2-SAT and #3-SAT with savings 0.671 and 0.2569 respectively, and [LPI01] give an algorithm for general $\#k$-SAT with savings $1/O\left(2^k\right)$.

Much less is known about the nature of savings for satisfiability problems for more expressive circuit classes. Schuler [Sch05] has shown that the satisfiability of an m-clause CNF F can be determined in time $|F|2^{n(1-\frac{1}{1+\lg m})}$. Using a bit more careful analysis, Calabro et al. [CIP06] have shown that the satisfiability of a cn-clause CNF F in n variables can be checked in time $|F|2^{n(1-\frac{1}{\Theta(\lg c)})}$. Subsequently, Calabro et al. [CIP09] have considered the question of satisfiability of bounded-depth unbounded fan-in circuits over standard basis ($\mathbf{AC}^0$ circuits) and shown that the satisfiability of $\mathbf{AC}^0$ circuits C of size cn and depth d can be decided in time $|C|2^{n(1-\mu)}$ where $\mu \geq 1/O(c^{2^{d-2}-1} \lg^{3\cdot 2^{d-2}-2} c)$. Santhanam [San10] has considered cn-size formulas F with no depth restriction and showed that the satisfiability problem for such formulas can be solved in time $|F|2^{n\left(1-\frac{1}{poly(c)}\right)}$. Like our results, Santhanam's also generalize to counting and enumerating satisfying assignments. More recently, Williams [Wil11] has shown that the satisfiability of $\mathbf{ACC}^0$ circuits C of size s can be solved in time $|C|2^{n\left(1-\Theta\left(\frac{n^{2^{-\Theta(d)}-\lg s}}{n}\right)\right)}$.

In this paper, we return to $\mathbf{AC}^0$ circuits and seek further improvement in the savings for the satisfiability algorithm for $\mathbf{AC}^0$ circuits of size cn and depth d. While the algorithm in [CIP09] obtains savings in terms of c and d independent of

n, its double exponentially small savings diminish rapidly to zero for $d > 2$ when c grows as a function of n. While Williams' algorithm [Wil11] provides nontrivial savings even when c grows sufficiently large as a function of n, its savings decreases with n even when c and d are constants. Furthermore, it leaves open the question whether one can obtain better savings for $\mathbf{AC}^0$ circuits.

Another independent motivation is the natural connection to proving lower bounds. Paturi et al. [PPZ99] have observed that the analysis that led to an improved upper bound for k-SAT can also be used to prove lower bounds for depth-3 $\mathbf{AC}^0$ circuits. Using this approach, they obtain a tight lower bound of $\Omega(n^{\frac{1}{4}}2^{\sqrt{n}})$ on the number of gates required to compute the parity function with depth-3 circuits. Subsequently, [PPSZ05] proposed a resolution-based k-SAT algorithm and obtained an improved savings (by a constant factor) using a sophisticated analysis. Using the same analysis, they construct a fairly simple function (checking whether the input binary string is a codeword of a certain code) which requires at least $2^{1.282\sqrt{n}}$ size for any depth-3 $\mathbf{AC}^0$ circuit. This is the best-known lower bound for depth-3 circuits for any function. Although depth reduction techniques based on Switching Lemma [Hås86a, Raz93, Bea94] and especially the top-down technique of [HJP95] for depth-3 circuits prove lower bounds that are close to $2^{\sqrt{n}}$, it is not clear that these techniques by themselves would yield a lower bound of $2^{c\sqrt{n}}$ for $c > 1$.

Recently, Williams [Wil10] established a formal connection between satisfiability upper bounds and circuit lower bounds. More precisely, he has shown that if there exists a deterministic algorithm for the CIRCUIT-SAT problem for polynomial size circuits which achieves $\omega(\lg n/n)$ savings, then **NEXP** does not have polynomial size nonuniform circuits. Subsequently, using this connection, he has obtained the first ever exponential size lower bound for $\mathbf{ACC}^0$ circuits for computing a function in **NEXP**. An important ingredient in this proof is the earlier mentioned satisfiability algorithm for **ACC** circuits which achieves nontrivial savings. The results of Williams show an improved satisfiability savings leads to stronger size lower bounds.

It has been a long open problem to prove $2^{\omega(n^{\frac{1}{d-1}})}$ size lower bounds for $\mathbf{AC}^0$ circuits of depth d. Since the breakthrough results of Yao and Håstad in the

mid 1980's, there have been only modest improvements [Yao85, Hås86a] and only for for depth-3 circuits [HJP95, PPSZ05]. It is tantalizing to prove better lower bounds by exploiting the connection between $\mathbf{AC}^0$ satisfiability upper bounds and lower bounds. Unfortunately, we do not have any ideas for proving strong enough upper bounds for $\mathbf{AC}^0$ satisfiability that would imply better lower bounds. Our modest goal is to obtain a satisfiability algorithm for $\mathbf{AC}^0$ with savings sufficient enough to imply some of the best-known lower bounds.

1.2 Main Results and Techniques

Our main result is an algorithm for $\mathbf{AC}^0$ satisfiability with the best known savings.

Theorem 1.1. *There is a Las Vegas algorithm for deciding the satisfiability of circuits with cn gates and depth d whose expected time has savings at least* $\frac{1}{O(\lg c + d \lg d)^{d-1}}$.

The above algorithm immediately follows from the existence of an algorithm that enumerates all satisfying assignments by partitioning the space into sub-cubes where the circuit is constant.

Theorem 1.2. *There exists a Las Vegas algorithm which, on input a cn gate, depth d circuit C in n variables, produces a set of restrictions* $\{\rho_i\}_i$ *which partition* $\{0,1\}^n$ *and such that for each i,* $C|_{\rho_i}$ *is constant. The expected time and number of restrictions are both* $poly(n)|C|2^{n(1-\mu_{c,d})}$*, where the savings* $\mu_{c,d}$ *is at least* $\frac{1}{O(\lg c + d \lg d)^{d-1}}$.

We discuss the algorithm for enumerating satisfying assignments to $\mathbf{AC}^0$ circuits in Chapter 5.

In Chapter 4, we consider the more restricted case of depth 2 circuits (CNFs and DNFs). This chapter both provides background in k-SAT algorithms, techniques for the $\mathbf{AC}^0$ algorithm, and provides new results for enumerating and counting the solutions to depth 2 circuits.

Theorem 1.3. *There exists a randomized algorithm which takes a k-CNF or k-DNF F in n variables and outputs a set of restrictions which partition $\{0,1\}^n$ and make F constant. The algorithm outputs at most $2^{n(1-\frac{1}{O(k)})}$ restrictions and runs in time $|F| \cdot poly(n) \cdot 2^{n(1-\frac{1}{O(k)})}$ with probability at least $1-2^{-n}$.*

While algorithms for k-SAT with savings $1/k$ were already known, previous results do not extend to enumerating solutions. In particular, the following corollary is new and answers affirmatively an open question of [LPI01].

Corollary 1.4. *There exists a randomized algorithm for #k-SAT with savings $1/O(k)$.*

As a corollary to Theorem 1.2, we obtain the following size and depth bounds for computing parity by $\mathbf{AC}^0$ circuits, which are comparable to the best known [Hås86a].

Theorem 1.5. *Any $poly(n)$ size $\mathbf{AC}^0$ which computes parity must have depth at least $\frac{\lg n}{\lg\lg n} - O\left(\frac{\lg n}{\lg^2 \lg n}\right)$. Any depth d $\mathbf{AC}^0$ circuit which computes parity must have size at least $2^{\Omega(n)^{\frac{1}{d-1}}}$.*

This is not surprising, because we use a version of the Håstad Switching Lemma that was used to prove these lower bounds. However, our results also imply new, very tight, bounds on how well constant-depth circuits can approximate the parity. As another corollary to Theorem 1.2, we obtain the following bounds on correlation between $\mathbf{AC}^0$ circuits and the parity function. Recently and independently from this work, Håstad [Hås11] achieved a similar bound on correlation (although our result is better for small values of m). We discuss these and other lower bound results in Chapter 7

Theorem 1.6. *The correlation of parity with any $\mathbf{AC}^0$ circuit of size cn and depth d is at most $2^{-\mu_{c,d}n} = 2^{-n/O(\log c + d\log d)^{d-1}}$.*

Overall, our results are particularly interesting in several regimes:

- For linear sized families of circuits, with c and d constants independent of n, we obtain constant savings μ and a $2^{-\Omega(n)}$ bound on the correlation with parity.

Calabro, Impagliazzo, and Paturi [CIP09] also give an algorithm with constant savings for satisfiability in this case, but our savings are more than exponentially better ($\approx 1/O\left(\lg c\right)^d$ versus $\approx 1/O\left(c^{2^d}\right)$). In addition our algorithm also enumerates solutions and solves the counting problem.

- We obtain non-trivial savings and correlations bounds for circuits up to size $m = 2^{O(n)^{\frac{1}{d-1}}}$.

  Since parity can be computed exactly by circuits of size $m = 2^{\Omega(n)^{\frac{1}{d-1}}}$, this is the essentially the best possible for correlation bounds and for enumerating satisfying assignments. Håstad [Hås11] independently gives a similar correlation bound. Beame, Impagliazzo, and Srinivasan [BIS11] also give an improved algorithm and correlation bound, but only for quasi-polynomial size circuits.

- For k-CNFs and cn clause CNFs, we extend the best known savings $1/O\left(k\right)$ and $1/O\left(\lg c\right)$ respectively for satisfiability to also include counting and enumerating satisfying assignments.

  Previously, the best known savings for #k-SAT, due to Littman, Pitassi, and Impagliazzo [LPI01], were exponentially small in k.

- For k-CNFs, CNFs, and $\mathbf{AC}^0$ circuits, our algorithms for covering the set of solutions by sub-cubes (enumerating solutions) are optimal up to a constant factor in the savings μ. In addition, any algorithm for $\mathbf{AC}^0$ satisfiability with savings μ more than polynomially better than ours would imply $\mathbf{NEXP} \not\subseteq \mathbf{NC}^1$ via Williams' technique [Wil10, Wil11].

A key ingredient in our analysis is an extended switching lemma. Previous switching lemmas showed that with high probability applying an appropriate random restriction to a single k-CNF yields a circuit which has a small height decision tree. Specifically, the probability that the k-CNF fails to have a height h decision tree decreases exponentially in h, so the expected height is constant. We would like to show a "concentration bound" on the height of decision trees when the same random restriction is applied to a set of k-CNFs. Ideally, it would be nice

to show that the sum of the heights of the decision trees for the set of k-CNFs is concentrated around (or upper bounded with high probability by) what previous switching lemmas would predict for independent restrictions. However, if the k-CNFs are very similar, the heights of the decision trees will be highly correlated, so with moderate probability the sum will be very large. But in this case, since the k-CNFs are very similar, the variables involved in their decision trees will be similar as well. The following lemma intuitively shows that this is the only way the distribution of the sum can be "unconcentrated." With high probability a set of k-CNFs won't have more than about the expected number of k-CNFs with large decision trees on *different sets of variables*. We will prove these results and give additional background on switching lemmas in Chapter 3.

Lemma 1.7 (Extended Switching Lemma)**.** *Let $\phi_1, \ldots, \phi_m$ be a sequence of k-CNFs and/or k-DNFs in the same n variables. For any $p \leq 1/13$, let $\rho \sim \mathcal{R}_n^p$. The expected number of nodes at distance t from the root in tree$((\phi_1, \ldots, \phi_m)|_\rho)$ such that the each ϕ_i contributes at least one variable to the path to the node is at most $\left(\frac{12pk}{1-p}\right)^t \leq (13pk)^t$.*

Using the observation that any path in the canonical decision tree for $(\phi_1, \ldots, \phi_m)|_\rho$ is also a path in the canonical decision tree for the subset of k-CNFs or k-DNFs which contribute nodes to the path, and using a union bound, we get the following corollary.

Corollary 1.8. *Let $\phi_1, \ldots, \phi_m$ be a sequence of k-CNFs and/or k-DNFs in the same n variables. For any $p \leq 1/13$, let $\rho \sim \mathcal{R}_n^p$. The expected number of nodes at distance t from the root in tree$((\phi_1, \ldots, \phi_m)|_\rho)$ is at most $\sum_{i=1}^{\min(t,m)} \binom{m}{i} \left(\frac{12pk}{1-p}\right)^t \leq (13pk)^t$.*

1.3 Acknowledgements

Portions of this chapter are joint work with Russell Impagliazzo and Ramamohan Paturi and appear in [IMP11].

Chapter 2

Preliminaries

2.1 Circuits and Satisfiability

We consider Boolean circuits consisting of alternating layers of AND and OR gates. The inputs are n variables $x_1, \ldots, x_n$ and their negations $\neg x_1, \ldots, \neg x_n$. Unless otherwise stated, we assume that circuits have a single output. We also assume that all wires are between adjacent layers. The *depth* of a circuit is the number of layers of AND and OR gates, and the *size* of a circuit is the total number of gates. In general, we consider families of circuits, one for each number of inputs n in which case we view the depth and the size of the circuits as functions of n. The *fan-in* of a gate is the number of inputs to the gate. We number the layers from top to bottom. Layer 1 consists of the top or the output gate. We regard the inputs as layer $d + 1$ (or bottom) gates. We assume that inputs of a gate as well as the gates in a layer are ordered.

We further categorize circuits. The simplest are depth two circuits with unbounded fan-in gates, CNFs and DNFs. CNFs are circuits in *conjunctive normal form*, which consist of an AND of ORs of inputs. We refer to each OR in a CNF as a *clause*. DNFs are are circuits in *disjunctive normal form*, which consist of an OR of ANDs of inputs, and each AND in a DNF is referred to as a term. k-CNFs and k-DNFs are CNFs and DNFs respectively where the clauses/terms have fan-in at most k (the output gate still has unbounded fan-in).

The next larger class of circuits we consider is $\mathbf{AC}^0$. $\mathbf{AC}^0$ circuits have

constant depth, polynomial size, and consist of unbounded fan-in AND and OR gates. More generally, $\mathbf{AC}^i$ generalizes $\mathbf{AC}^0$ to depth $O\left(\lg^i n\right)$. The class $\mathbf{NC}^i$ refers to depth $O\left(\lg^i n\right)$, polynomial size circuits consisting of fan-in 2 AND and OR gates. Note that $\mathbf{NC}^i \subseteq \mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$. Since $\mathbf{NC}^0$ circuits can only compute a function of a constant number of inputs, $\mathbf{NC}^0 \subset \mathbf{AC}^0$. It is known (and we will reprove in Chapter 7) that $\mathbf{AC}^0$ circuits cannot compute parity whereas $\mathbf{NC}^1$ can, so $\mathbf{AC}^0 \subset \mathbf{NC}^1$. Beyond $\mathbf{NC}^1$ it is not known if the inclusions are proper.

Other authors define $\mathbf{AC}^i$ and $\mathbf{NC}^i$ to allow NOT gates throughout the circuit and allow wires between non-adjacent layers. All NOT gates can be moved to the inputs using De Morgan's laws, and this at most doubles the size of the circuit. Wires between non-adjacent layers can be replaced by sequences of fan-in 1 gates, one for each intermediate layer. This increases the size of the circuit by at most a factor d. Since these changes only increase the size of the circuits by a $2d$ factor, we use our more restrictive definition which will make future results simpler to state.

For arbitrary constants $m_1, \ldots, m_\ell$, the class $\mathbf{ACC}^0(m_1, \ldots, m_\ell)$ generalizes $\mathbf{AC}^0$ by allowing unbounded fan-in $MOD_{m_1}, \ldots, MOD_{m_\ell}$ and NOT gates, in addition to AND and OR gates. Here, as before, we could also assume that the NOT gates are just at the inputs, but for our purposes it won't make a difference. MOD_m gates output 0 if the sum of its inputs is 0 modulo m and 1 otherwise. The class $\mathbf{ACC}^0$ is the union of $\mathbf{ACC}^0(m_1, \ldots, m_\ell)$ over all finite ℓ and $m_1, \ldots, m_\ell$.

For technical reasons, we parameterize $\mathbf{AC}^0$ circuits further. For circuits on n inputs, we parameterize them by m, which is the number of gates per layer and d, the number of layers. We call such circuits (n, m, d)-*circuits.* where m and d could be functions of n. For technical reasons we are also interested in a slight variant of (n, m, d)-circuits where we only require that each gate at level d has fan-in bounded by k (rather than limiting the number of gates level d), for some k. All other layers are still required to have at most m gates. We refer to these circuits as (n, m, d, k)-*circuits.*

Given a circuit C, the satisfiability problem is to find an assignment to the variables which causes the circuit to output 1 (a satisfying assignment). When

the circuit is a CNF or k-CNF, we refer to the satisfiability problem as CNF-SAT or k-SAT respectively. We refer to the satisfiability problem for arbitrary circuits as CIRCUIT-SAT. In addition to finding a single satisfying assignment for a circuit, we are also interested in counting the number of satisfying assignments. The counting problems for CNFs, k-CNFs, and arbitrary circuits are referred to as #CNF-SAT, #k-SAT, and #CIRCUIT-SAT respectively. By the Cook–Levin Theorem [Coo71, Lev73], all of these satisfiability problems except for 2-SAT are **NP**-complete, and the corresponding counting problems are #**P**-complete. In addition, Valiant [Val79a, Val79b] showed that #2-SAT is #**P**-complete (2-SAT is in **P**).

2.1.1 Subexponential Size $\mathbf{AC}^0$ Circuits for More General Classes

Since many of our results remain non-trivial even for subexponential size circuits, we give some sense of how subexponential size $\mathbf{AC}^0$ circuits relate to other classes of circuits. The following lemma, which generally seems to be regarded as folklore, gives a simple transformation from bounded fan-in circuits (such as $\mathbf{NC}^1$) to constant depth circuits with subexponential size.

Lemma 2.1. *Any depth $D \geq 1$ boolean circuit C where every gate has fan-in at most $k \geq 2$ can be converted to an equivalent depth $d \geq 2$ size $k^D 2^{k^{\frac{D}{d-1}}}$ $\mathbf{AC}^0$ circuit C' with either an AND or an OR output gate. The gates in C may compute arbitrary functions of their inputs, not just AND or OR.*

The idea is to divide the D layers of C into $d-1$ groups of $\frac{D}{d-1}$ consecutive layers, and then replace each group of layers with a depth 2 circuit.

Proof. We will prove a slightly stronger size bound than Lemma 2.1. The size of the resulting circuit will be at most $\frac{k^D-1}{k^{\frac{D}{d-1}}-1} 2^{k^{\frac{D}{d-1}}}$. We prove this by induction on d. If $d = 2$ then we can construct either an AND of ORs or an OR of ANDs of size 2^{k^D} equivalent to C since C depends on at most k^D inputs.

If $d > 2$ then let $\phi_1, \ldots, \phi_\ell$ denote the subcircuits of C rooted at depth $\frac{D}{d-1} + 1$, and let ψ denote the top $\frac{D}{d-1}$ layers of C. By induction, we can convert

$\phi_1, \ldots, \phi_\ell$ to equivalent depth $d-1$ size $\frac{k^{D-\frac{D}{d-1}}-1}{k^{\frac{D}{d-1}}-1} 2^{k^{\frac{D}{d-1}}}$ $\mathbf{AC}^0$ circuits $\phi'_1, \ldots, \phi_\ell$. Assume without loss of generality that we wish to construct C' with an AND output gate (the OR case is symmetric). Construct $\phi'_1, \ldots, \phi'_\ell$ with OR output gates. Construct a depth 2 AND of ORs ψ' equivalent to ψ. Replace the inputs to ψ' with $\phi'_1, \ldots, \phi'_\ell$ and combine the adjacent layers of OR gates to get C'.

All that remains is to bound the size of C'. Since each gate in ψ has fan-in at most k and ψ has depth $\frac{D}{d-1}$, it has at most $k^{\frac{D}{d-1}}$ inputs. These inputs correspond to the circuits $\phi_1, \ldots, \phi_\ell$. The size of C' is at most $k^{\frac{D}{d-1}}$ times the size of each ϕ'_i plus the size of ψ', which is at most

$$k^{\frac{D}{d-1}} \cdot \frac{k^{D-\frac{D}{d-1}}-1}{k^{\frac{D}{d-1}}-1} 2^{k^{\frac{D}{d-1}}} + 2^{k^{\frac{D}{d-1}}} = \frac{k^D-1}{k^{\frac{D}{d-1}}-1} 2^{k^{\frac{D}{d-1}}}.$$

□

We use Lemma 2.1 to get two useful corollaries. The first using the definition of $\mathbf{NC}^1$, and the second since the parity of n inputs can be computed by a depth $\lg n$ circuit consisting of fan-in 2 parity gates.

Corollary 2.2. *Any depth $c \lg n$ $\mathbf{NC}^1$ circuit C can be converted into an equivalent $\mathbf{AC}^0$ circuit of size $n^c 2^{n^{\frac{c}{d-1}}}$.*

Corollary 2.3. *The parity of n inputs can be computed by a depth d size $n2^{n^{\frac{1}{d-1}}}$ $\mathbf{AC}^0$ circuit.*

Allender, Hellerstein, McCabe, Pitassi, and Saks [AHM+06] give, among other things, a much more general variant of Lemma 2.1.

Lemma 2.4. *For every $\mathbf{SAC}^1$ circuit C in n variables with depth $c \lg n$ and every $\epsilon > 0$, there exists a d depending on c and ϵ such that C can be converted to an equivalent depth d size 2^{n^ϵ} $\mathbf{AC}^0$ circuit.*

$\mathbf{SAC}^1$ is the class of depth $O(\lg n)$ circuits consisting of unbounded fan-in OR gates and fan-in 2 AND gates (or vice versa since $\mathbf{SAC}^1$ is closed under complement [BCD+89]). Clearly $\mathbf{NC}^1 \subseteq \mathbf{SAC}^1 \subseteq \mathbf{AC}^1$. Furthermore, $\mathbf{SAC}^1 = \mathbf{LOGCFL}/poly$ [Ven87] and $\mathbf{NL}/poly \subseteq \mathbf{LOGCFL}/poly$. $\mathbf{LOGCFL}/poly$ is the

class of languages reducible in non-uniform log space to context-free languages and $\mathbf{NL}/poly$ is the class of languages decidable by non-uniform nondeterministic log space Turing Machines. We do not particularly consider these less common classes of circuits and languages, but we present them as evidence of the power of 2^{n^ϵ} size $\mathbf{AC}^0$ circuits.

2.1.2 Block Parity Circuits

Parity is one of the canonical functions which *cannot* be computed by $\mathbf{AC}^0$ circuits. As we show in Section 7.2.2 parity cannot even be well approximated by $\mathbf{AC}^0$ circuits. In this section, we construct a family of circuits which are somewhat "parity-like," yet can be constructed in the restricted classes of circuits we consider (k-CNFs, CNFs, and $\mathbf{AC}^0$ circuits). The same general family of circuits was also used by Håstad [Hås86b] as "small" circuits that approximate parity. These circuits serve as a worst case example for many of our results, and in some cases show that our results are essentially optimal.

These circuits, which we call "block parity" circuits can be defined for a variety of classes of circuits. Fix a class of circuits (k-CNFs, depth d $\mathbf{AC}^0$, etc.). Take the n inputs and divide them into n/ℓ groups of ℓ inputs. Construct a circuit with an AND output gate to compute the parity of each group of ℓ inputs, and then construct the AND of of these circuits. For each class of circuits, we choose ℓ as large as possible such that the resulting circuit remains in the desired class. For example, the block parity circuit for k-CNFs sets $\ell = k$, divides the inputs into n/k groups of k, and constructs 2^{k-1} clauses for each group to compute the parity. For size m depth d $\mathbf{AC}^0$ circuits, we can compute the parity on ℓ inputs in depth d with at most $\ell 2^{\ell^{\frac{1}{d-1}}}$ gates by Corollary 2.3, and the additional AND gate can be combined with the top level gates of the parity circuits. Choose $\ell = (\lg \frac{m}{n})^{d-1}$ so that $\frac{n}{\ell} \cdot \ell 2^{\ell^{\frac{1}{d-1}}} \leq m$.

2.2 Restrictions and Regions

Formally, a restriction ρ on a set of variables V is a map $\rho : V \to \{0, 1, *\}$. Applying the restriction to a function f, denoted $f|_\rho$, results in a function where $\rho(v)$ is substituted for each input v when $\rho(v) \neq *$. We say that the variables v where $\rho(v) = *$ are *unset*. After applying a restriction, $f|_\rho$ is a function in the unset variables in the natural way. A restriction may also be applied to a circuit C. Replace each input v with $\rho(v)$ when $\rho(v) \neq *$. Then simplify gates with constant inputs: If an AND gate has a 0 input, replace the gate with 0; if it has a 1 input, remove that input. If an OR gate has a 0 input, remove that input; if it has a 1 input, replace the gate with 1. Repeat this process until either the circuit is a single constant, or no constants remain. For our purposes, we will also view restrictions ρ as functions $\hat{\rho} : \{0, 1\}^n \to \{0, 1\}$ where $\hat{\rho}(x_1, \ldots, x_n) = 1$ if and only if $\rho(i) \in \{x_i, *\}$ for all i.

We say that a set of functions $\phi_1, \ldots, \phi_m : \{0, 1\}^n \to \{0, 1\}$ partitions $\{0, 1\}^n$ if for every $x \in \{0, 1\}^n$ there exists exactly one i such that $\phi_i(x) = 1$. We say that the region $\{x \in \{0, 1\}^n | \phi_i(x) = 1\}$ is defined by ϕ_i. In particular, we are interested in regions defined by functions of the form $R \wedge \rho$, where R is a k-CNF and ρ is a restriction. In this case, we will say that $\mathcal{R} = (R, \rho)$ defines the region. Unless otherwise noted, regions will always be defined this way and the value of k will be clear from the context. In a slight abuse of terminology, we will often refer to $\mathcal{R}$ as the region.

We say that two circuits, C and D, are equivalent in a region $\mathcal{R}$ if $\mathcal{R} \implies (C \equiv D)$.

We define regions this way because in several algorithms we will need to branch on disjunctions of k variables. This corresponds to partitioning the current region. In one branch, when the disjunction is true, we add the disjunction as a clause to R. In the other branch, when the disjunction is false, we set the value of the k variables in the restriction. In each branch, we will then simplify the current circuit we are considering based on the branch.

Definition 2.5. Let C be a circuit in n variables. We say that a set $\mathcal{P} = \{(\mathcal{R}_i = (R_i, \rho_i), C_i)\}_i$ is a *partitioning for* C if $\{\mathcal{R}_i\}_i$ defines a set of regions which partition

$\{0,1\}^n$ and for every i, C is equivalent to C_i in the region $\mathcal{R}_i$. We will say that each C_i is the *circuit associated with* the region $\mathcal{R}_i$.

We generalize this definition to sequences of circuits in the natural way.

Definition 2.6. Let $\Phi = (\phi_1, \ldots, \phi_m)$ be a sequence of circuits in the same n variables (or equivalently, a circuit with n inputs and m outputs). We say that a set $\mathcal{P} = \{(\mathcal{R}_i = (R_i, \rho_i), \Psi_i = (\psi_{i,1}, \ldots, \psi_{i,m}))\}_i$ is a *partitioning for* Φ if $\{\mathcal{R}_i\}_i$ defines a set of regions which partition $\{0,1\}^n$ and for every i and j, ϕ_j is equivalent to $\psi_{i,j}$ in the region $\mathcal{R}_i$.

2.3 Canonical Decision Trees

In several places we will use the notion of the *canonical decision tree* [Bea94] for a CNF or DNF ϕ, $tree(\phi)$. For a CNF ϕ, write ϕ as $C \wedge \phi'$ where C is the first clause of ϕ and ϕ' is the rest of ϕ. Construct $tree(\phi)$ by first constructing the complete binary tree that queries all of the variables in C in order. Label the leaf that falsifies C with 0 and replace each of the other leaves with $tree(\phi'|_\rho)$ where ρ is the path in the tree to the leaf. When ρ satisfies ϕ' or $\phi'|_\rho$ has an unsatisfied clause, the tree is 1 or 0 accordingly, otherwise it is constructed recursively. For a DNF ϕ, we do the analogous process, labeling the leaf that satisfies the first term with 1 and recursively construct the other leaves.

We also similarly define the decision tree for a tuple of CNFs and/or DNFs $\Phi = (\phi_1, \ldots, \phi_\ell)$. First construct $tree(\phi_1)$ and then replace each leaf with $tree((\phi_2, \ldots, \phi_\ell)|_\rho)$ where ρ is the restriction corresponding to the path to the leaf. Label the leaves of the resulting tree with the tuples of leaves from the original trees.

When we construct a decision tree for a circuit ϕ by exploring it clause by clause, each encounter of a variable which corresponds to a particular occurrence of it ϕ results in a branching node in the decision tree. We say that the branching node is *contributed by* the particular occurrence of the variable. Similarly for any subcircuit ψ of ϕ, we regard the set of nodes contributed by all the occurrences of variables in ψ as the set of nodes contributed by Ψ. Define the height of a decision

tree T, $height(T)$, as the length of the longest path. We will view paths in decision trees as restrictions in the natural way.

2.4 Acknowledgements

Portions of this chapter are joint work with Russell Impagliazzo and Ramamohan Paturi and appear in [IMP11].

Chapter 3

Switching Lemmas

At a very general level, a switching lemma says that with high probability a circuit of one type of circuit can be converted to an equivalent circuit of another type after applying a random restriction from a particular distribution. Perhaps the most common switching lemmas convert from k-CNFs to k'-DNFs and conversely. Furst, Saxe, and Sipser [FSS84] and independently Ajtai [Ajt83] first used switching lemmas in this context to prove that parity cannot be computed by polynomial size $\mathbf{AC}^0$ circuits. Suppose that a $poly(n)$ size, depth d $\mathbf{AC}^0$ circuit C existed which computed parity. Choose a random restriction and apply it to all of the subcircuits of C rooted at depth $d-1$ and switch them from CNFs to DNFs (or vice versa). Then repeat this process $d-2$ times to get a depth 2 circuit. However, any depth two circuit computing the parity of n inputs requires size $\Omega(2^n)$. They show that sufficiently many variables remain unset to arrive at a contradiction.

Yao [Yao85] improved the lower bound for the size of $\mathbf{AC}^0$ circuits computing parity to exponential. Cai [Cai89] used similar techniques to show that $\mathbf{PSPACE}^A \neq \mathbf{PH}^A$ with probability 1 for a random oracle A. This result was built upon a proof that not only can parity not be computed by $\mathbf{AC}^0$ circuits, it can't even be approximated with probability much more than 1/2. We will give nearly optimal bounds on how well parity can be approximated by $\mathbf{AC}^0$ circuits in Chapter 7.

Then Håstad [Hås86a] gave a switching lemma (Lemma 3.1) which is optimal except perhaps for constants. He then used this switching lemma to prove

that any depth d $\mathbf{AC}^0$ circuit requires size $2^{\Omega(n)^{\frac{1}{d-1}}}$, which is also optimal up to constants. We will give a different proof of this same result in Chapter 7. Håstad also gets similar lower bounds for $\mathbf{AC}^0$ circuits computing majority, MOD_p, and other functions.

Razborov [Raz93] gives a new proof of a switching lemma qualitatively equivalent to Håstad's (but quantitatively slightly weaker). Razborov's proof is quite amenable to modification and will be the basis for our extended switching lemma.

Many switching lemmas of this form can actually be stated in a stronger way: After applying an appropriate random restriction, any k-CNF or k-DNF has a decision tree of height at most k' with high probability. This was first observed by Cai [Cai89], and subsequently shown to hold for Håstad's and Razborov's Switching Lemmas. In particular, it holds for the canonical decision tree (Section 2.3). Given a decision tree T of height at most k' for a function f, we can construct a k'-DNF for f by constructing a term for each path in T ending at a leaf labeled 1. Similarly, we can construct a k'-CNF for f by constructing a clause corresponding to the complement of each path in T ending at a leaf labeled 0 (equivalently, by constructing the negation of the k'-DNF for the negation of T). Thus, we see that stating switching lemmas in terms of decision trees is at least as powerful as stating them in terms of k'-DNFs. Furthermore, we will use the fact that the canonical decision tree is constructive to design algorithms. Håstad's and Razborov's Switching Lemmas bound the probability that a "long" path exists in the canonical decision tree by bounding the expected number of such paths. Since the bound on the expected number of "long" paths will be useful to us, we will prove these switching lemmas in this form.

More generally, switching lemmas have been used extensively to prove a variety of lower bounds in a variety of contexts, for exmaple [Ajt83, FSS84, Yao85, Hås86a, Lyn86, Cai89, BH89, Bea90, Raz93, BIK+92, SBI02] and many others.

In the remainder of this chapter, we will make precise Håstad's Switching Lemma, Razborov's Switching Lemma, and our extended switching lemma, and give proofs for the second two.

3.1 Håstad's Switching Lemma

The results in this section are due to Håstad [Hås86a] and presented in greater detail in [Hås86b]. Let $\mathcal{R}'^p_n$ denote the distribution on restrictions on n variables where each variable is left unset with probability p, set to 0 with probability $(1-p)/2$, and set to 1 with probability $(1-p)/2$.

In each of the following three lemmas, let $\alpha < \frac{2pk}{(1+p)\ln\frac{1+\sqrt{5}}{2}} \approx \frac{4.16pk}{1+p}$ denote the unique positive solution to

$$\left(1+\frac{4p}{1+p}\frac{1}{\alpha}\right)^k = \left(1+\frac{2p}{1+p}\frac{1}{\alpha}\right)^k + 1.$$

To see that $\alpha < \frac{2pk}{(1+p)\ln\frac{1+\sqrt{5}}{2}}$, let $\beta = \alpha\frac{1+p}{2p}$. β is a root of $(1+2/\beta)^k-(1+1/\beta)^k-1 = 0$ if and only if α is a root of $\left(1+\frac{4p}{1+p}\frac{1}{\alpha}\right)^k = \left(1+\frac{2p}{1+p}\frac{1}{\alpha}\right)^k + 1$. We will show that $(1+2/\beta)^k-(1+1/\beta)^k-1 = 0$ has a root $1 < \beta < k/\ln\frac{1+\sqrt{5}}{2}$ for all $k \geq 2$. Substituting 1 for β in $(1+2/\beta)^k-(1+1/\beta)^k-1$ gives $3^k-2^k-1 > 0$ (assuming $k \geq 2$). On the other hand,

$$(1+2/\beta)^k-(1+1/\beta)^k-1 < e^{2k/\beta}-e^{k/\beta}-1 = \left(e^{k/\beta}-\frac{1+\sqrt{5}}{2}\right)\left(e^{k/\beta}-\frac{1-\sqrt{5}}{2}\right)$$

and substituting $k/\ln\frac{1+\sqrt{5}}{2}$ for β in $\left(e^{k/\beta}-\frac{1+\sqrt{5}}{2}\right)$ gives 0. Thus, since $(1+2/\beta)^k-(1+1/\beta)^k-1$ is continuous when $\beta > 0$, it must have a root $1 < \beta < k/\ln\frac{1+\sqrt{5}}{2}$.

Lemma 3.1 (Håstad's Switching Lemma, [Hås86a]). *For any k-CNF F and any $0 < p < 1$, let $\rho \sim \mathcal{R}'^p_n$. The probability that $F|_\rho$ cannot be written as a k'-DNF is at most $\alpha^{k'+1}$.*

Lemma 3.2 (Håstad's Switching Lemma, Decision Tree Version). *For any k-CNF or k-DNF F and any $0 < p < 1$, let $\rho \sim \mathcal{R}'^p_n$. The probability that tree$(F|_\rho)$ has height greater than k' is at most $\alpha^{k'+1}$.*

Lemma 3.3 (Håstad's Switching Lemma, Expected Number of Paths Version). *For any k-CNF or k-DNF F and any $0 < p < 1$, let $\rho \sim \mathcal{R}'^p_n$. The expected number of nodes at distance k' from the root in the canonical decision tree for tree$(F|_\rho)$ is at most $\alpha^{k'}$.*

Since Lemma 3.1 and Lemma 3.2 follow from Lemma 3.3, we will just sketch the proof of Lemma 3.3. We actually prove a slightly stronger statement: Lemma 3.3 hold even if we condition on ρ setting an arbitrary function to true.

Lemma 3.4. *Let $G = \wedge_{i=1}^{m} G_i$ be a k-CNF. Let F be an arbitrary function and let $\rho \sim \mathcal{R}'^{p}_{n}$. Then for each s, let $N_s(G|_\rho)$ denote the number of nodes in the canonical decision tree for $G|_\rho$ at distance s from the root. $\mathbf{E}_\rho [N_s(G|_\rho) \mid F|_\rho \equiv 1] \leq \alpha^s$, where α satisfies*

$$\left(1 + \frac{4p}{1+p}\frac{1}{\alpha}\right)^k - \left(1 + \frac{2p}{1+p}\frac{1}{\alpha}\right)^k - 1 = 0.$$

This proof follow's Håstad's proof [Hås86b] closely. We just rephrase parts in terms of the expected number of nodes in the decision tree rather than minterms.

Proof. We prove this lemma by induction on m. If $m = 0$, $G = 1$ and the canonical decision tree consists of only the root and the lemma hold trivially.

Otherwise, consider the first clause G_1, and whether ρ satisfies G_1. Let $G' = \wedge_{i=2}^{m} G_i$ denote the rest of G.

$$\mathop{\mathbf{E}}_{\rho}[N_s(G|_\rho) \mid F|_\rho \equiv 1] =$$
$$\max(\mathop{\mathbf{E}}_{\rho}[N_s(G|_\rho) \mid F|_\rho \equiv 1 \wedge G_1|_\rho = 1], \mathop{\mathbf{E}}_{\rho}[N_s(G|_\rho) \mid F|_\rho \equiv 1 \wedge G_1|_\rho \neq 1])$$

.

The first term, $\mathbf{E}_\rho [N_s(G|_\rho) \mid F|_\rho \equiv 1 \wedge G_1|_\rho = 1]$ is equivalent to $\mathbf{E}_\rho [N_s(G'|_\rho) \mid (F \wedge G_1)|_\rho = 1]$ which is at most α^s by induction.

Now consider the second term, $\mathbf{E}_\rho [N_s(G|_\rho) \mid F|_\rho \equiv 1 \wedge G_1|_\rho \neq 1]$. Let T denote the set of variables in G_1, and write ρ as $\rho_1\rho_2$ where ρ_1 is a restriction on the variables in T and ρ_2 is a restriction on the remaining variables. The condition $G_1|_\rho \neq 1$ is equivalent to saying ρ_1 does not satisfy the clause G_1. Either ρ_1 falsifies G_1 in which case $G|_\rho = 0$ and the canonical decision tree consist of only the root, or ρ_1 leaves some variables in T unset.

In this last case, the canonical decision tree for $G|_\rho$ begins by constructing the canonical decision tree for $G_1|_\rho$, and then recursively constructs the canonical decision tree for G' at each leaf. The size of the canonical decision tree for $G_1|_\rho$

depends on the number of unset variables in ρ_1. We use the following lemma of Håstad.

Lemma 3.5 ([Hås86b, Lemma 4.3])**.** *For any set $Y \subseteq T$ of variables,*

$$\mathbf{Pr}\left[\rho(Y) = * \mid F|_\rho \equiv 1 \wedge G_1|_\rho \neq 1\right] \leq \left(\frac{2p}{1+p}\right)^{|Y|}.$$

Without the conditioning $F|_\rho \equiv 1$ the probability would be exactly $\left(\frac{2p}{1+p}\right)^{|Y|}$. Intuitively, adding the conditioning cannot make $*$s any more likely.

Given this lemma, it is fairly straightforward to bound the expected number of leaves by first summing over all possible numbers of unset variables in ρ_1, then summing over all paths in $tree(G_1|_\rho)$.

$$\begin{aligned}
&\mathop{\mathbf{E}}_{\rho}\left[N_s(G|_\rho) \mid F|_\rho \equiv 1 \wedge G_1|_\rho \neq 1\right] \\
&\leq \sum_{t=0}^{k} \mathop{\mathbf{Pr}}_{\rho}\left[\rho \text{ has } t \text{ unset variables in } T \mid F|_\rho \equiv 1 \wedge G_1|_\rho \neq 1\right] \\
&\qquad \sum_{\substack{\rho' \in tree(G_1|_\rho) \\ G_1|_{\rho'} \neq 0}} \mathop{\mathbf{E}}_{\rho}\left[N_{s-t}(G'|_\rho) \mid F|_\rho \equiv 1 \wedge \rho \text{ consistent with } \rho'\right] \\
&\leq \sum_{t=0}^{k} \mathop{\mathbf{Pr}}_{\rho}\left[\rho \text{ has } t \text{ unset variables in } T \mid F|_\rho \equiv 1 \wedge G_1|_\rho \neq 1\right](2^t - 1)\alpha^t \\
&\leq \sum_{t=0}^{k} \binom{k}{t}\left(\frac{2p}{1+p}\right)^t (2^t - 1)\alpha^{s-t} \\
&= \alpha^s \sum_{t=0}^{k} \binom{k}{t}\left(\left(\frac{4p}{\alpha(1+p)}\right)^t - \left(\frac{2p}{\alpha(1+p)}\right)^t\right) \\
&= \alpha^s \left[\left(1 + \frac{4p}{\alpha(1+p)}\right)^k - \left(1 + \frac{2p}{\alpha(1+p)}\right)^k\right] = \alpha^s
\end{aligned}$$

□

3.2 Razborov's Switching Lemma

Let $\mathcal{R}_n^\ell$ denote the set of restrictions on n variables which leave exactly ℓ variables unset.

Lemma 3.6 (Razborov's Switching Lemma [Raz93, Section E.4]). *For any k-CNF F and any $0 < p < 1/9$, let $\rho \sim \mathcal{R}_n^{pn}$. The probability that $F|_\rho$ cannot be written as a k'-DNF is at most $\left(\frac{8pk}{1-p}\right)^{k'+1} \leq (9pk)^{k'+1}$.*

Lemma 3.7 (Razborov's Switching Lemma, Decision Tree Version). *For any k-CNF or k-DNF F and any $0 < p < 1/9$, let $\rho \sim \mathcal{R}_n^{pn}$. The probability that $tree(F|_\rho)$ has height greater than k' is at most $\left(\frac{8pk}{1-p}\right)^{k'+1} \leq (9pk)^{k'+1}$.*

Lemma 3.8 (Razborov's Switching Lemma, Expected Number of Paths Version). *For any k-CNF or k-DNF F and any $0 < p < 1/9$, let $\rho \sim \mathcal{R}_n^{pn}$. The expected number of nodes at distance t from the root in the canonical decision tree for $tree(F|_\rho)$ is at most $\left(\frac{8pk}{1-p}\right)^{t} \leq (9pk)^{t}$.*

Let $\beta \approx k/\ln\frac{1+\sqrt{5}}{2} \approx 2.08k$ be the unique positive root of $(1+2/\beta)^k = (1+1/\beta)^k + 1$. The probability bound may be improved to $\left(\frac{2p\beta}{1-p}\right)^{k'+1}$ (for comparison, we may write the probability bound in Håstad's Switching Lemma as $\left(\frac{2p\beta}{1+p}\right)^{k'+1}$) at the expense of making the proof more complicated. For our purposes, the simpler bound will be sufficient. See [Bea94] for details.

Before we give a formal proof of Razborov's Switching Lemma, we will give some intuition and then an outline. Intuitively, either a k-CNF only depends on a very small fraction of its variables in any significant way, or with very high probability a random restriction falsifies a clause in the formula. In the extreme case, if clauses are on disjoint sets of variables (or in some more general sense "independent"), the a random restriction falsifies the formula with all but exponentially small probability. In either case, the fact that under particular restriction a k-CNF has a large canonical decision tree (also implying that no clause in the k-CNF is falsified) conveys a great deal of information about the restriction. Razborov's proof shows how this fact, combined with a small amount of additional information, can be "decoded" to give the original restriction. Since a small amount of information can be decoded to get any "bad" restriction, there cannot be too many "bad" restrictions.

First, we assume that ρ does not falsify any clause in F, since if it did the decision tree would have height 0. Suppose that $F|_\rho$ has a decision tree of height

$t \geq s+1$, and let P be a path in this decision tree of length t. P corresponds uniquely to a node at distance t in the canonical decision tree for $F|_\rho$. Consider the process of constructing the decision tree for $F|_\rho$ along the path P. Find the first clause C such that $C|_\rho \not\equiv 1$, and for simplicity assume $C = (x_1 \vee x_2 \vee \cdots \vee x_k)$. Since C is neither satisfied or falsified by ρ, we know that $\rho(C) = \{0, *\}$. In particular, let x_i be the first variable in C such that $\rho(x_i) = *$. When constructing the decision tree, the first variable queried is x_i. Let p_i be the value assigned to x_i along P. We the repeat this process to construct the decision tree for $F|_{\rho, x_i = p_i}$.

The idea of Razborov's proof is to encode the pair (ρ, P) using a restriction $\rho' = \rho\sigma$, where σ is a restriction on the variables along the path P, a vector $\vec{i} \in [k]^t$, and a vector $\vec{p} \in [2]^t$. Given $\rho', \vec{i}$, and $\vec{p}$, we wish to figure out the clause C which contributed the first variable along P. Since we know ρ doesn't falsify any of the clauses in F, we may choose σ so that $C|_{\rho\sigma} \equiv 0$. In this case, the first clause falsified by $\rho' = \rho\sigma$ must be C. Let x_1 be the first variable along P. Let v_1 be the index of x_1 in C. Since we can decode which clause C is, we can decode which variable x_1 is. Let p_1 be the value P assigns to x_1. Now change σ by setting $\sigma(x_1) = p_1$. Now if we repeat the process, the first clause falsified by $\rho\sigma$ (using the new σ) will contain the second variable along P with index v_2, and so forth until we have decoded all of P.

The reason we do this encoding is because the set $S = \mathcal{R}_n^{pn-t} \times [k]^t \times [2]^t$ is much smaller than the set of all restrictions that leave pn variables unset. Specifically, $|S|/|\mathcal{R}_n^{pn}| \leq \left(\frac{8pk}{1-p}\right)^t$, and this bounds the probability that for a random $\rho \in \mathcal{R}_n^{pn}$, such a path P exists.

The proof we give of Razborov's Switching Lemma is based on Beame's proof [Bea94], modified to be as similar as reasonable to our proof of our extended switching lemma, at the expense of a slightly poorer constant.

Proof. Let P be a path in $tree(F|_\rho)$ to a node at distance t from the root. Let $x_1, \ldots, x_t$ denote the variables along P and let $p_1, \ldots, p_t$ denote the values that P assigns to $x_1, \ldots, x_t$. Let $C_1, \ldots, C_t$ denote the clauses which contribute $x_1, \ldots, x_t$ respectively (some C_is may refer to the same clauses if these clauses contribute more than one variable to P). Let $index_1, \ldots, index_t$ denote the indices

of $x_1, \ldots, x_t$ in the clauses $C_1, \ldots, C_t$. Let $last_i, 1 \leq i \leq t$ be 1 if x_i is the last variable contributed by C_i along P.

Let $\sigma = \sigma_1 \cdots \sigma_{s+1}$ where σ_i is a restriction where $\sigma_i(x_i) = 0$ if x_i appears positively in C_i and $\sigma_i(x_i) = 1$ otherwise, and $\sigma_i(y) = *$ for all $y \neq x_i$ (σ is constructed to *not satisfy* the clauses $C_1, \ldots, C_{s+1}$). Let π_i be the restriction where $\pi_i(x_i) = p_i$ and $\pi_i(y) = *$ for $y \neq x_i$. Note that $P = \pi_1 \cdots \pi_{s+1}$

We map (ρ, P) to $\rho' = \rho\sigma \in \mathcal{R}_n^{pn-t}, \vec{index} = (index_1, \ldots, index_t) \in [k]^t, \vec{last} = (last_1, \ldots, last_t) \in [2]^t$ and $\vec{p} = (p_1, \ldots, p_t) \in [2]^t$.

We must now show that we can decode (ρ, P) from $\rho', \vec{index}, \vec{last}$ and $\vec{p}$. Let $\rho_i = \rho\pi_1 \cdots \pi_i\sigma_{i+1} \cdots \sigma_{s+1}$, for $0 \leq i \leq s+1$. Note that $\rho_0 = \rho'$ and $\rho_{s+1} = \rho P$.

We will show that for any $i < s+1$ given C_i and ρ_i, we can decode π_{i+1} and therefore ρ_{i+1}. Then by induction, given ρ' we can decode ρ and P. Let $last_0 = 1$. If $last_i = 0$ then we know $C_{i+1} = C_i$, otherwise we claim that C_{i+1} is the first clause not satisfied by ρ_i. Once we identify C_{i+1}, then $index_{i+1}$ is the index of x_{i+1} in this clause and we get π_i using p_i. All that remains is to prove the claim that when $last_i = 1$ then C_{i+1} is first clause in F not satisfied by ρ_i.

When x_{i+1} is queried along the path P when constructing the decision tree for $F|_\rho$, C_{i+1} is the first clause not satisfied by $\rho\pi_1 \cdots \pi_i$ (otherwise a variable in an earlier clause would have been queried instead). Since setting more variables in a restriction cannot change a clause from satisfied to not satisfied, all the clauses in F before C_i are satisfied by ρ_i. All that remains is to show that $\sigma_{i+1} \cdots \sigma_{s+1}$ does not satisfy C_{i+1}. Let j be the largest index such that $C_j = C_{i+1}$, or equivalently x_j is the last variable from C_{i+1} along P. By the construction of $\sigma_{i+1}, \ldots, \sigma_j$, $\sigma_{i+1} \cdots \sigma_j$ does not satisfy C_{i+1} and either $j = t$ or $\rho\pi_1 \cdots \pi_i\sigma_{i+1} \cdots \sigma_j$ sets all of the variables in C_{i+1} (because of the way we construct decision trees). In either case, no $\sigma_\ell, \ell > j$ can satisfy C_{i+1} so we conclude that ρ_i does not satisfy C_{i+1}.

All that remains is to calculate $\mathbf{E}_\rho$[number of nodes at distance t from the root in $tree(F|_\rho)$]. Each node corresponds to a path P, and each pair (ρ, P) can be encoded by an element of $|\mathcal{R}_n^{pn-t} \times [k]^t \times [2]^t \times [2]^t|$. Thus the expected number

of such paths is at most

$$\leq \frac{|\mathcal{R}_n^t \times [k]^t \times [2]^t \times [2]^t|}{|\mathcal{R}_n^{pn}|} = \frac{\binom{n}{pn-t} 2^{n-pn+t} (4k)^t}{\binom{n}{pn} 2^{n-pn}}$$
$$= \frac{(pn)!(n-pn)!}{(pn-t)!(n-pn+t)!} (8k)^t$$
$$= \left(\frac{pn}{n-pn+t}\right)\left(\frac{pn-1}{n-pn+t-1}\right)\cdots\left(\frac{pn-t+1}{n-pn+1}\right)(8k)^t$$
$$\leq \left(\frac{pn}{n-pn+t}\right)^t (8k)^t \leq \left(\frac{8pk}{1-p}\right)^t.$$

□

3.3 Extended Switching Lemma

The primary use of switching lemmas is to reduce the depth of circuits by "switching" the subcircuits rooted at depth $d-1$ from k-CNFs to k-DNFs or vice versa. For this to work, after we apply a random restriction, we have to be able to "switch" all of these gates. For his lower bounds, Håstad chose his parameters such that a union bound sufficed ($k \approx \lg |C|$, see Chapter 7 for more details). For our $\mathbf{AC}^0$ satisfiability algorithm, we would like to keep k constant at least when the size of the circuit is linear in n. When k is constant, however, we expect a constant fraction of the subcircuits to not "switch". The algorithm will be able to handle some subcircuits not "switching", but we need to bound the probability that too many fail to "switch".

In some ways, our Extended Switching Lemma can be viewed as a concentration bound on the number of subcircuits that fail to "switch". However, there is one catch. Suppose we apply a random restriction to a set of CNFs that are all (essentially) the same. Then either all the CNFs will "switch" or the will all fail to "switch". Our Extended Switching Lemma and the subsequent corollary say that either a set of the CNFs fail to "switch" for essentially the same reason (the same set of variables) or whether or not they "switch" acts approximately independently.

Lemma (Extended Switching Lemma, Lemma 1.7, restated)**.** *Let $\phi_1, \ldots, \phi_m$ be a*

sequence of k-CNFs and/or k-DNFs in the same n variables. For any $p \leq 1/13$, let $\rho \sim \mathcal{R}_n^p$. The expected number of nodes at distance t from the root in $tree((\phi_1, \ldots, \phi_m)|_\rho)$ such that the each ϕ_i contributes at least one variable to the path to the node is at most $\left(\frac{12pk}{1-p}\right)^t \leq (13pk)^t$.

Corollary (Corollary 1.8, restated)**.** *Let $\phi_1, \ldots, \phi_m$ be a sequence of k-CNFs and/or k-DNFs in the same n variables. For any $p \leq 1/13$, let $\rho \sim \mathcal{R}_n^p$. The expected number of nodes at distance t from the root in $tree((\phi_1, \ldots, \phi_m)|_\rho)$ is at most $\sum_{i=1}^{\min(t,m)} \binom{m}{i} \left(\frac{12pk}{1-p}\right)^t \leq (13pk)^t$.*

This corollary follows from the observation that any path P in $tree((\phi_1, \ldots, \phi_m)|_\rho)$ is also a path in $tree(S|_\rho)$ where S is the subsequence of $\phi_1, \ldots, \phi_m$ of circuits that contribute variables to P, and then summing Lemma 1.7 over all subsequences of length at most t.

The proof of Lemma 1.7 is basically the same as the proof of Razborov's Switching Lemma in Section 3.2 with one exception. There, we used the variable $last_i$ to indicate when we had finished decoding all of the variables in a clause. Here we will also use $last_i$ to indicate when we finish decoding all of the variables in a formula.

Proof. Let P be a path in $tree((\phi_1, \ldots, \phi_m)|_\rho)$ of length t where each ϕ_i contributes at least one variable to the path. Let $x_1, \ldots, x_t$ denote the variables along P and let $p_1, \ldots, p_t$ denote the values that P assigns to $x_1, \ldots, x_t$. Let $C_1, \ldots, C_t$ and $F_1, \ldots, F_t$ denote the clauses and formulae respectively which contribute $x_1, \ldots, x_t$ (some C_is and F_is may refer to the same clauses or formulae if they contribute more than one variable to P). Let $index_1, \ldots, index_t$ denote the indices of $x_1, \ldots, x_t$ in the clauses $C_1, \ldots, C_t$. Let $last_i, 1 \leq i \leq t$ be 2 if x_i is the last variable contributed by F_i along P; be 1 if x_i is the last variable contributed by C_i (but not by F_i) along P; and be 0 otherwise.

Let $\sigma = \sigma_1 \cdots \sigma_{s+1}$ where σ_i is a restriction where $\sigma_i(x_i) = 0$ if x_i appears positively in C_i and $\sigma_i(x_i) = 1$ otherwise, and $\sigma_i(y) = *$ for all $y \neq x_i$ (σ is constructed to *not satisfy* the clauses $C_1, \ldots, C_{s+1}$). Let π_i be the restriction where $\pi_i(x_i) = p_i$ and $\pi_i(y) = *$ for $y \neq x_i$. Note that $P = \pi_1 \cdots \pi_{s+1}$

We map (ρ, P) to $\rho' = \rho\sigma \in \mathcal{R}_n^{pn-t}, \vec{index} = (index_1, \ldots, index_t) \in [k]^t, \vec{last} = (last_1, \ldots, last_t) \in [3]^t$ and $\vec{p} = (p_1, \ldots, p_t) \in [2]^t$.

We must now show that we can decode (ρ, P) from $\rho', \vec{index}, \vec{last}$ and $\vec{p}$. Let $\rho_i = \rho\pi_1 \cdots \pi_i\sigma_{i+1} \cdots \sigma_{s+1}$, for $0 \leq i \leq s+1$. Note that $\rho_0 = \rho'$ and $\rho_{s+1} = \rho P$.

We will show that for any $i < s+1$ given C_i, F_i and ρ_i, we can decode π_{i+1} and therefore ρ_{i+1}. Then by induction, given ρ' we can decode ρ and P. Let $last_0 = 1$ and let $F_0 = \phi_1$. First we identify F_{i+1}. If $last_i = 2$ then $F_{i+1} = \phi_{q+1}$ where q is the index such that $F_i = \phi_q$, and otherwise $F_{i+1} = F_i$. If $last_i = 0$ then we know $C_{i+1} = C_i$, otherwise we claim that C_{i+1} is the first clause not satisfied by ρ_i. Once we identify C_{i+1}, then $index_{i+1}$ is the index of x_{i+1} in this clause and we get π_i using p_i. All that remains is to prove the claim that when $last_i = 1$ then C_{i+1} is first clause in F not satisfied by ρ_i.

When x_{i+1} is queried along the path P when constructing the decision tree for $F|_\rho$, C_{i+1} is the first clause not satisfied by $\rho\pi_1 \cdots \pi_i$ (otherwise a variable in an earlier clause would have been queried instead). Since setting more variables in a restriction cannot change a clause from satisfied to not satisfied, all the clauses in F before C_i are satisfied by ρ_i. All that remains is to show that $\sigma_{i+1} \cdots \sigma_{s+1}$ does not satisfy C_{i+1}. Let j be the largest index such that $C_j = C_{i+1}$, or equivalently x_j is the last variable from C_{i+1} along P. By the construction of $\sigma_{i+1}, \ldots, \sigma_j$, $\sigma_{i+1} \cdots \sigma_j$ does not satisfy C_{i+1} and either $j = t$ or $\rho\pi_1 \cdots \pi_i\sigma_{i+1} \cdots \sigma_j$ sets all of the variables in C_{i+1} (because of the way we construct decision trees). In either case, no $\sigma_\ell, \ell > j$ can satisfy C_{i+1} so we conclude that ρ_i does not satisfy C_{i+1}.

All that remains is to calculate $\mathbf{E}_\rho$[number of nodes at distance t from the root in $tree((\phi_1, \ldots, \phi_m)|_\rho)$ such that the each ϕ_i contributes at least one variable to the path to the node]. Each pair (ρ, P) can be encoded by an element of

$|\mathcal{R}_n^{pn-t} \times [k]^t \times [3]^t \times [2]^t|$. Thus the expected number of such paths is at most

$$\begin{aligned}
&\leq \frac{|\mathcal{R}_n^t \times [k]^t \times [3]^t \times [2]^t|}{|\mathcal{R}_n^{pn}|} = \frac{\binom{n}{pn-t} 2^{n-pn+t} (6k)^t}{\binom{n}{pn} 2^{n-pn}} \\
&= \frac{(pn)!(n-pn)!}{(pn-t)!(n-pn+t)!} (12k)^t \\
&= \left(\frac{pn}{n-pn+t}\right)\left(\frac{pn-1}{n-pn+t-1}\right)\cdots\left(\frac{pn-t+1}{n-pn+1}\right)(12k)^t \\
&\leq \left(\frac{pn}{n-pn+t}\right)^t (12k)^t \leq \left(\frac{12pk}{1-p}\right)^t.
\end{aligned}$$

□

3.4 Acknowledgements

Material in Section 3.3 is joint work with Russell Impagliazzo and Ramamohan Paturi and appears in [IMP11].

Chapter 4

Algorithms for CNF Satisfiablity

Recall that we say a satisfiability algorithm has savings μ if it runs in (expected) time $s \cdot poly(n) \cdot 2^{n(1-\mu)}$ on circuits of size s in n variables. We will discuss four different techniques used in k-SAT algorithms that all achieve savings $1/O(k)$.

Currently, the fastest knows algorithm for k-SAT is due to Paturi, Pudlák, Saks, and Zane [PPSZ05]. The PPSZ algorithm has savings $\frac{\mu_k}{k-1}$ where $\mu_k = \sum_{j=1}^{\infty} \frac{1}{j(j+\frac{1}{k-1})}$ (μ_k is an increasing function in k, with $\mu_3 = 4 - 4\ln 2 \approx 1.227$ and $\lim_{k\to\infty} \mu_k = \frac{\pi^2}{6} \approx 1.645$). For $k = 3, 4$ these bounds were proved by [Her11]. The PPSZ algorithm is based on an earlier algorithm of Paturi, Pudlák, and Zane [PPZ99]. The PPSZ algorithm performs a limited amount of resolution and then runs the PPZ algorithm. The PPZ algorithm by itself solves k-SAT with savings $\frac{1}{k}$.

Schöning [Sch99] gives a different algorithm for k-SAT based on local search, and Lokshtanov and Paturi give an algorithm based on approximating k-CNFs by low degree polynomials and then using a fast zeta transform. Both of these algorithm also achieves savings $1/O(k)$.

Using different techniques from all of these results, we also give an algorithm for k-SAT with savings $\frac{1}{O(k)}$, however our algorithm not only finds a satisfying assignment if one exists, it enumerates all satisfying assignments in the same amount of time. We will discuss the PPZ algorithm in Section 4.1, Schöning's algorithm in Section 4.2, the Lokshtanov–Paturi algorithm in Section 4.3, and our algorithm

in Section 4.4.

Schuler [Sch05] gives a Turing reduction from CNF-SAT with cn clauses to k-SAT which may be composed with an algorithm for k-SAT with savings δ_k to get an algorithm for CNF-SAT with cn clauses with savings approximately $\delta_k - c2^{-k}$. Choosing $k \approx \lg c$ and composing Schuler's reduction with PPZ or PPSZ gives an algorithm for CNF-SAT with cn clauses with savings $\frac{1}{\lg c}$, and composing Schuler's reduction with our algorithm enumerates the solutions to a cn clause CNF with savings $\frac{1}{O(\lg c)}$. We present these results in Section 4.5.

4.1 PPZ Algorithm

Algorithm 4.1: PPZ(F)

```
   Data: F is a k-CNF in n variables, x_1, ..., x_n
   Result: 1 with high probability if F is satisfiable, 0 otherwise
 1 begin
 2     repeat poly(n)2^{n(1-1/k)} times
 3         let π be a permutation of [n] chosen uniformly at random
 4         for i ← 1, ..., n do
 5             if x_{π(i)} is in a unit clause C in F then
 6                 set x_{π(i)} to satisfy C
 7             else
 8                 set x_{π(i)} uniformly at random
 9             end
10             if F is satisfied then
11                 return 1
12             end
13         end
14     end
15     return 0
16 end
```

Algorithm 4.1 gives the PPZ k-SAT algorithm [PPZ99]. It chooses a random order for the variables, and then assigns values to the variables in that order. If a variable is in a unit clause when it is its turn to have a value assigned, the value is chosen to satisfy the clause. Otherwise the value is chosen randomly. They show that each iteration of this process succeeds with probability $2^{-n(1-\frac{1}{k})}$, so by

repeating this process $poly(n)2^{n(1-\frac{1}{k})}$ times it will find a satisfying assignment if one exists with high probability.

Theorem 4.1. *If a k-CNF F is satisfiable, the PPZ algorithm returns 1 with high probability, and if F is unsatisfiable the PPZ algorithm always returns 0.*

To see why this works, assume for now that F has a unique satisfying assignment, and assume that each time a variable is assigned a random value it gets the value that is consistent with the unique satisfying assignment. A unique satisfying assignment means that for every variable x, there must be a clause that is satisfied by only x (otherwise negating x would also give a satisfying assignment). Say that such a clause is *critical* for x. If a variable x occurs in the permutation π after all the other variables in its critical clause then the critical clause will be a unit clause when x is assigned a value. In this case, say that x is *forced*. Since π is chosen uniformly at random, we expect a $1/k$ fraction of the variables to occur last in their critical clause and be forced. Than means that the algorithm only needs to randomly guess the assignment for $n(1-1/k)$ variables. Thus, each iteration succeeds in finding a unique satisfying assignment with probability $2^{-n(1-1/k)}$.

In general, we cannot assume that F has a unique satisfying assignment, much less anything about the solution space. However, as an intermediate step, say that a satisfying assignment is *isolated* if changing the assignment for any single variable changes the assignment to non-satisfying. If F only has isolated satisfying assignments then the previous intuition still holds since each variable will still have at least one critical clause with respect to each isolated satisfying assignment. If F has non-isolated satisfying assignments, then consider an assignment and a variable x such that the assignment satisfies F regardless of how x is set. In this case, it doesn't matter how x is set when it is assigned a random value. Extending this intuition, for any satisfying assignment either a variable has a critical clause and is forced with probability $1/k$, or it doesn't matter what value it is assigned.

[PPZ99] formalize this intuition and prove Theorem 4.1. [CIKP08a] take this intuition further and show that having multiple satisfying assignments not only doesn't hurt the performance of the PPZ algorithm, it strictly helps. Formally, they prove that if a k-CNF F has s satisfying assignments then each iteration of the PPZ

algorithm returns 1 with probability at least $\left(\frac{s}{2^n}\right)^{1-1/k}$.

Why do we care so much about the PPZ algorithm when the PPSZ algorithm performs better? For small k, PPSZ gives a significantly faster running time. For example, for 3-SAT PPSZ runs in time $\approx 2^{0.387n}$ whereas PPZ runs in time $\approx 2^{0.667n}$. However asymptotically the PPSZ algorithm only gives a small improvement over PPZ as k increases (both have savings $1/O(k)$). Furthermore PPSZ is based on PPZ, so understanding PPZ would seem to be a prerequisite. The PPZ algorithm itself is quite simple and elegant and its analysis is, for the most part, straightforward and intuitive. PPSZ adds complexity to the algorithm and is less intuitive. Finally, in general, we are more interested in the asymptotic behavior of the savings for large k and n. In this regard, PPZ, PPSZ, and Schöning's algorithm (which we will discuss next) are equivalent.

4.2 Schöning's k-sat Algorithm

Schöning [Sch99] gives a very different algorithm (Algorithm 4.2) for k-SAT based on local search. Assume that a k-CNF F is satisfiable and that $a^* \in \{0,1\}^n$ is a satisfying assignment. Choose a random assignment a. We will keep track of the Hamming distance between a^* and a. If the Hamming distance reaches 0 then $a = a^*$ and we've found a satisfying assignment. We may find a satisfying assignment sooner, but this can only help us.

As long as a isn't a satisfying assignment, let C be a clause in F not satisfied by a. Choose a variable i uniformly from C. Changing the setting of i in the assignment a decreases the Hamming distance between a^* and a by one with probability at least $1/k$ (a falsifies all the literals in C, and a^* satisfies at least one of them, and we randomly choose this one with probability $1/k$). View how this process changes the Hamming distance between a^* and a as a random walk on $0, \ldots, n$ which ends when it reaches 0.

The initial Hamming distance between a and a^* is j with probability $2^{-n}\binom{n}{j}$. For each j, consider random walks that take i steps in the "wrong" direction and $i+j$ steps in the "right" direction. The algorithm will consider all $0 \leq i \leq j \leq n$,

but for the sake of intuition just consider $i = j/(k-2)$. The probability that the algorithm takes i steps in the wrong direction and $i+j$ steps in the right direction, given that the initial Hamming distance was j is

$$\binom{j+2i}{i}\left(\frac{k-1}{k}\right)^i\left(\frac{1}{k}\right)^{i+j} = \binom{j\left(\frac{k}{k-2}\right)}{j\left(\frac{1}{k-2}\right)}\left(\frac{k-1}{k}\right)^i\left(\frac{1}{k}\right)^{i+j}$$
$$\approx k^i(k-1)^i\left(\frac{1}{k}\right)^{j+2i} \approx \left(\frac{1}{k}\right)^j$$

using the fact that $\binom{n}{k} \approx \left(\frac{n}{k}\right)^k$. Now summing over j we get an overall success probability of

$$\sum_{j=0}^{n} 2^{-n}\binom{n}{j}\left(\frac{i}{k}\right)^j = \left(\frac{1}{2}\left(1+\frac{1}{k}\right)\right)^n.$$

Using somewhat more careful calculations, Schöning shows that the process succeeds with probability at least $(\frac{1}{2}(1+\frac{1}{k-1}))^n \geq 2^{-n\left(1-\frac{1}{k\ln 2}\right)}$. The choice of $3n$ allows the random walk to take up to n steps in the "wrong" direction and leaves $2n$ steps in the right direction.

By running this process $poly(n)2^{n\left(1-\frac{1}{k\ln 2}\right)}$ times independently, we get a k-SAT algorithm with savings $\frac{1}{k\ln 2}$ that succeeds with high probability.

Algorithm 4.2: Schöning (F)

```
   Data: F is a k-CNF in n variables
 1 begin
 2     repeat poly(n)2^{n(1-1/(k ln 2))} times
 3         choose an assignment a ∈ {0,1}^n uniformly at random
 4         repeat 3n times
 5             if F(a) = 1 then
 6                 return 1
 7             end
 8             let C be an arbitrary clause in F such that C(a) = 0
 9             choose a variable i ∈ C uniformly at random
10             a ← a ⊕ i                    // change the setting for i in a
11         end
12     end
13     return 0                             // probably not satisfiable
14 end
```

4.3 Lokshtanov–Paturi Algorithm

Lokshtanov and Paturi [LP11] give an algorithm for k-SAT with savings $1/O(k)$ based a modification of Williams' [Wil11] $\mathbf{ACC}^0$ satisfiability algorithm. The idea of Williams' algorithm is to take an $\mathbf{ACC}^0$ circuit C in n variables, choose a parameter $m \ll n$ and the construct the circuit $C'(x_1, \ldots, x_{n-m}) = \vee_{x_{n-m+1},\ldots,x_n} C(x_1, \ldots, x_n)$ which computes the disjunction of the circuits where the last m variables are set in all possible ways. Then Williams constructs a polynomial $\tilde{C}'(x_1, \ldots, x_{n-m})$ such that it's easy to compute $C'(x_1, \ldots, x_{n-m})$ given $\tilde{C}'(x_1, \ldots, x_{n-m})$. Finally, he uses Yates' [Yat37] dynamic programming algorithm for the zeta transform to compute the value of $\tilde{C}'(x_1, \ldots, x_{n-m})$ on all 2^{n-m} inputs. (Originally, Williams used a different approach to evaluate $\tilde{C}'$. The zeta transform and Yates' Algorithm were suggested to him by Andreas Björklund.)

In this section, we will always view polynomials as a sum of monomials (rather than, say, arithmetic circuits) and say that the size of a polynomial is simply the number of monomials. Yates' algorithm will evaluate a polynomial represented this way on all 2^{n-m} inputs in time $poly(n)2^{n-m}$, even if the size of the polynomial is $\Omega(2^{n-m})$ (note that the naive evaluation of this size polynomial would require time $\Omega(2^{2(n-m)})$). The key to Williams' algorithm is to choose m and the structure of $\tilde{C}'$ such that the polynomial can be constructed in time $poly(n)2^{n-m}$ as well. This is the step which requires particular ingenuity, and where Lokshtanov and Paturi's algorithm will differ from Williams'.

Lokshtanov and Paturi convert a k-CNF gate by gate into a polynomial in the same set of variables over $GF(2)$. It is straightforward to see that $\neg x$ can be represented by the polynomial $(1 - x)$ and $x_1 \wedge x_2 \wedge \cdots \wedge x_\ell$ can be represented by $x_1 x_2 \cdots x_\ell$. By De Morgan's laws we get that $x_1 \vee x_2 \vee \cdots \vee x_\ell$ corresponds to $1 - (1 - x_1)(1 - x_2) \cdots (1 - x_\ell)$. The problem with this naive approach is that it will take time $\Omega(2^n)$ just to convert the k-CNF into a polynomial, and even longer if they first set m variables in all possible ways. Instead, they will construct a polynomial which only approximates the k-CNF but which has sufficiently low degree that they can bound the size of the polynomial and the time required to construct it. They do this with a technique due to Razborov [Raz87].

Razborov gives a method of converting a circuit into a low degree polynomial which approximates the circuit, and then uses this approach to show that majority cannot be computed by $\mathbf{ACC}^0(2)$ circuits ($\mathbf{AC}^0$ with parity gates). Smolensky [Smo87] generalizes this technique to show that MOD_q cannot by computed by $\mathbf{ACC}^0(p)$ circuits where p and q are distinct primes (or powers of primes).

To approximate an OR $f(x_1, \ldots, x_\ell) = x_1 \vee x_2 \vee \cdots \vee x_\ell$, choose a random subset $S \subseteq \{1, \ldots, \ell\}$ where each i is in S with probability $1/2$. The sum $f_S(x_1, \ldots, x_\ell) = \sum_{i \in S} x_i$ (over $GF(2)$) is equal to $f(x_1, \ldots, x_\ell)$ when $f(x_1, \ldots, x_\ell) = 0$ and when $f(x_1, \ldots, x_\ell) = 1$, $f_S(x_1, \ldots, x_\ell) = 1$ with probability exactly $1/2$. To see this, fix an x_j which equals 1. Then the sum $\sum_{i \in S \setminus \{j\}} x_i$ may be either 0 or 1, but i is in S with probability exactly $1/2$ so the output of f_S is equally likely to be 0 or 1. We may improve this probability to $1 - 2^{-t}$ by choosing t sets $S_1, \ldots, S_t \subseteq \{1, \ldots, \ell\}$ and then let $\tilde{f}(x_1, \ldots, x_\ell) = f_{S_1}(x_1, \ldots, x_\ell) \vee \cdots \vee f_{S_t}(x_1, \ldots, x_\ell)$, where the OR is computed by the naive formula. By De Morgan's laws, we get a similar formula for AND.

Now we will give the overview of the Lokshtanov–Paturi Algorithm. Let $F = \wedge_{i=1}^{\ell} C_i$ be a k-CNF in n variables and ℓ clauses. Let t and m be parameters which we will optimize shortly. Write each clause C_i as a degree (at most) k polynomial $\tilde{C}_i$ using the naive formula. Use Razborov's approximation with t random sets $S_1, \ldots, S_t \subseteq \{1, \ldots, \ell\}$ followed by the naive formula for AND to get a degree tk polynomial $\tilde{F}$ which approximates F. When $F(\vec{x}) = 1$ then $\tilde{F}(\vec{x}) = 1$, and when $F(\vec{x}) = 0$ then $\tilde{F}(\vec{x}) = 0$ with probability $1 - 2^{-t}$.

Let $G = \vee_{x_{n-m+1}, \ldots, x_n} F(x_1, \ldots, x_n)$ be the circuit the computes the disjunction of F where the last m variables are set in all possible ways. Let $\tilde{G}$ be the result of applying Razborov's approximation to the OR gate of G and using $\tilde{F}$ instead of F. Note that $\tilde{G}$ is a polynomial in $n - m$ variables. For any assignment $\vec{x}$, let $\vec{x}'$ denote the corresponding assignment to the $n - m$ variables remaining in $\tilde{G}$. If $F(\vec{x}) = 1$ then $\tilde{F}(\vec{x}) = 1$, and then $\tilde{G}(\vec{x}') = 1$ with probability $1/2$; and if $F(\vec{x}) = 0$ then $\tilde{F}(\vec{x}) = 0$ with probability at least $1 - 2^{-t}$, and then by a union bound $\tilde{G}(\vec{x}') = 0$ with probability at least $1 - \frac{1}{2} 2^m 2^{-t}$. Use the fast zeta transform to evaluate $\tilde{G}$ on all 2^{n-m} inputs. As long as $t > m$, we have at least

a constant gap between $\mathbf{Pr}\left[\tilde{G}(\vec{x}') = 1 \mid F(\vec{x}) = 1\right]$ and $\mathbf{Pr}\left[\tilde{G}(\vec{x}') = 1 \mid F(\vec{x}) = 0\right]$ so by repeating the process $poly(n)$ times we can identify a satisfiable k-CNF with all but exponentially small probability.

The remaining piece is to choose values for m and t. The degree of $\tilde{G}$ will be at most kt, so it will contain at most $\binom{n}{kt}$ monomials (and can be constructed in the same amount of time, ignoring $poly(n)$ factors), so in this regard, we want t small. On the other hand, the zeta transform will take time 2^{n-m} (ignoring $poly(n)$ factors), so we want m large. But we require $m < t$. We will set $m = t - 1$ and then approximately balance $\binom{n}{kt}$ and 2^{n-t+1}. By choosing $t = \frac{n}{4k} + 1$, we get

$$\binom{n}{kt} = \binom{n}{n/4} \leq (4e)^{n/4} < 2^{0.9n} < 2^{n-m} = 2^{n(1-\frac{1}{4k})}$$

giving savings $\frac{1}{4k}$. This can be optimized somewhat using tighter inequalities, but it will still only give savings $1/O(k)$.

4.4 Our k-sat Algorithm

Our k-SAT algorithm is a straightforward application of any of the decision tree versions of the switching lemmas in Chapter 3. The switching lemmas say that after applying a random restriction which sets a $1 - 1/O(k)$ fraction of variables, any k-CNF or k-DNF has a small canonical decision tree with high probability. In particular, the probability that any path in the canonical decision tree involves more than half of the unset variables is exponentially small. We first build a complete decision tree on a random set of $n(1 - 1/O(k))$ variables. Then, we view each path in this tree as a random restriction, and replace each leaf in this tree with the canonical decision tree for the k-CNF under the corresponding restriction. By the switching lemma, most of these canonical decision trees are small, so the overall decision tree doesn't grow very much when we do this. We can then iterate over the leaves of this tree to solve k-SAT, $\#k$-SAT, and to enumerate the solutions to a k-CNF.

In comparison to PPZ, PPSZ, and Schöning's algorithm, this algorithm has a somewhat poorer constant in the savings (we will show savings $\frac{1}{20k}$ versus $\approx \frac{1}{k}$),

but has the advantage that it enumerates solutions rather than just finding one, if one exists. Given the switching lemma, this algorithm is simple and intuitive like PPZ. However, the switching lemma is fairly complicated so PPZ has the advantage that it is more self-contained.

More precisely, let F be a k-CNF and let $\rho \sim \mathcal{R}_n^p$ for $p = \frac{1}{20k}$. By Lemma 3.8, $\mathbf{E}_\rho[\text{number of nodes in } tree(F|_\rho) \text{ at depth } t] \leq (9pk)^t = \left(\frac{9}{20}\right)^t$. Summing over t, the expected size of the canonical decision tree is at most $\mathbf{E}_\rho[size(tree(F|_\rho))] \leq \sum_{t=0}^{\infty}\left(\frac{9}{20}\right)^t < 2$. The algorithm chooses a random set of $n(1-\frac{1}{20k})$ variables and repeats this process for the 2^{n-pn} restrictions ρ which set this set of variables in all possible ways, to construct a decision tree of expected size at most $2 \cdot 2^{n-pn}$ for the original k-CNF. This is formalized in Algorithm 4.3 and Lemma 4.2.

Algorithm 4.3: KCNFEnumerate(F)

```
Data: F is a k-CNF or k-DNF in n variables
Result: a decision tree for F
begin
    let p = 1/(20k)
    choose a set V' of n(1 − p) variables uniformly at random
    let T be a complete decision tree on the variables in V'
    foreach path ρ in T do
        replace the leaf at the end of the path ρ in T with tree(F|_ρ)
    end
    return T
end
```

Lemma 4.2. *Let F be a k-CNF or k-DNF in n variables. The expected size of the decision tree constructed by $KCNFEnumerate(F)$ (Algorithm 4.3) is at most $2 \cdot 2^{n(1-\frac{1}{20k})}$ and the expected running time of the algorithm is at most $|F|poly(n)2^{n(1-\frac{1}{20k})}$.*

Corollary 4.3. *The problems k-SAT, #k-SAT, and enumerating the solutions to a k-CNF all have algorithms with savings at least $\frac{1}{20k}$.*

The algorithm for enumerating solutions to a k-CNF is optimal up to constants in the savings since the corresponding block parity circuit (Section 2.1.2) requires $2^{n-n/\Omega(k)}$ restrictions to partition $\{0,1\}^n$ and make the circuit constant.

KCNFEnumerate() naturally generalizes to construct a decision tree for a sequence $F_1, \ldots, F_m$ of k-CNFs and/or k-DNFs by using Corollary 1.8 in place Lemma 3.8 and $p = \frac{1}{30k}$, the rest of the algorithm and argument remain the same and we get the following lemma.

Lemma 4.4. *Let $F_1, \ldots, F_m$ be a sequence of k-CNFs and/or k-DNFs in the same n variables. There exists a randomized algorithm which constructs a decision tree for $(F_1, \ldots, F_m)$ of expected size at most $O\left((2^m - 1)2^{n(1-\frac{1}{30k})}\right)$.*

4.4.1 Optimizing the Constant

Algorithm 4.3 gives the best known approach for $\#k$-SAT for general k. In this section we will optimize the parameters to get the best known constant. This same constant also holds for k-SAT but this is less interesting since it is somewhat poorer than the best known.

Lemma 4.5. *There exists a Las Vegas algorithm for $\#k$-SAT with expected savings at least*

$$\frac{\ln \frac{1+\sqrt{5}}{2}}{2k} \approx \frac{1}{4.1562k}.$$

Proof. Use Algorithm 4.3 with $p = \frac{\ln \frac{1+\sqrt{5}}{2}}{2k}$. The proof is essentially the same as the proof of Lemma 4.2, however it is complicated slightly by the fact that we will use Håstad's Switching Lemma (Lemma 3.3) rather than Razborov's Switching Lemma (Lemma 3.8). Håstad's Switching Lemma says that after applying a random restriction from $\mathcal{R}'^p_n$ (where each variable is independently 0 with probability $(1-p)/2$, 1 with probability $(1-p)/2$, and $*$ with probability p), the expected number of nodes in the canonical decision tree at depth h is at most α^h where $\alpha^{\frac{2pk}{(1+p)\ln\frac{1+\sqrt{5}}{2}}}$ is the unique positive root of $\left(1 + \frac{4p}{1+p}\frac{1}{\alpha}\right)^k = \left(1 + \frac{2p}{1+p}\frac{1}{\alpha}\right)^k + 1$.

We need to calculate $\mathbf{E}_{\rho\sim\mathcal{R}'^p_n}\left[size(tree(F|_\rho)) \mid \rho \text{ has exactly } pn \text{ }*\text{s}\right]$. However, since p is a constant and the number of $*$s is distributed binomially, $\mathbf{Pr}_{\rho\sim\mathcal{R}'^p_n}\left[\rho \text{ has exactly } pn \text{ }*\text{s}\right] = \Omega\left(1/\sqrt{n}\right)$ so we may ignore the conditioning at the expense of at most an additional $O\left(\sqrt{n}\right)$ factor in the size of the tree.

The choice of $p = \frac{\ln \frac{1+\sqrt{5}}{2}}{2k}$ gives $\alpha < 1$. By the same reasoning as the proof of Lemma 4.2 $\mathbf{E}_{\rho \sim \mathcal{R}'^p_n}[size(tree(F|_\rho))] \leq O(\sqrt{n}) \sum_{t=0}^{\infty} \alpha^t = O(\sqrt{n})$. Thus we get savings $p = \frac{\ln \frac{1+\sqrt{5}}{2}}{2k} \approx \frac{1}{4.1562k}$. □

4.5 Schuler's Algorithm

Schuler [Sch05] gives a Turing reduction from CNF-SAT to k-SAT, where the running time of the reduction will depend on the parameter k, the number of variables n and the number of clauses m. We will be able to choose $k = O\left(\lg \frac{m}{n}\right)$ such that the overhead from Schuler's reduction is much less than the $\approx \frac{1}{k}$ savings from any of the previous k-SAT algorithms.

Let F be a CNF. The idea is, as long as F contains a clause C of width greater than k, to branch on the disjunction C' of the first k literals in C. If C' is true, then we replace C with C', and if C' is false, then we know that all the literals in C' must be false so we reduce the number of variables by k. This process is repeated until F is a k-CNF.

The same idea may be applied to a DNF or even an arbitrary collection of conjunctions (clauses) and disjunctions (terms). When considering a term T of width greater than k, branch on the conjunction T' of the first k literals and proceed accordingly.

Consider the execution of Schuler's reduction as a tree where each node corresponds to branching on some disjunction (or conjunction) of k literals. To use this reduction its fullest extent, we will keep track of the number of variables remaining at each leaf in this tree.

Schuler's reduction outputs a set of k-CNFs with varying numbers of remaining variables. In order to use Schuler's reduction efficiently, we need to bound the number of k-CNFs output and the number of variables remaining in each. View Schuler's reduction as constructing a tree where each leaf of the tree has a k-CNF. The following theorem bounds the number of k-CNFs generated by Schuler's reduction.

Theorem 4.6. *Let F be a* CNF *with m clauses and $k > 0$ be a parameter. For*

Algorithm 4.4: Schuler(F, k)

```
Data: F is a CNF and k ≥ 1 is a parameter
begin
    if F is a k-CNF then
        output F
    else
        let C be the first clause in F of width > k
        let C' be the conjunction of the first k literals in C
        Schuler(F ∪ {C'} \ {C}, k)                    // C' is true
        Schuler(F|_{C'=0}, k)                         // C' is false
    end
end
```

each $0 \leq f \leq n/k$, the computation tree of Schuler's algorithm has at most $\binom{m+f}{f}$ leaves with at most $n - fk$ variables remaining.

Proof. For any path in the computation tree, let f refer to the number of branches where the conjunction C' is false. Each of these branches sets k variables, so $f \leq n/k$. Each branch where C' is true reduces the number of clauses of width $> k$ by one, so there can be at most m such branches. Thus, the length of the path is at most $m + f$. For each f, we can bound the number of paths with f false branches by $\binom{m+f}{f}$. □

Now we'll compose Schuler's reduction and the PPZ algorithm (using PPSZ would improve the constants slightly, but they end up hidden in the big O anyway). Recall that PPZ runs on a k-CNF F in n variables in time $|F|poly(n)2^{n(1-\frac{1}{k})}$.

Theorem 4.7. *Let F be a* CNF *in n variables with m clauses. The composition of Schuler's reduction and PPZ can compute the satisfiability of F in expected time $2^{n\left(1-\frac{1}{O(\lg(m/n))}\right)}$.*

Proof. These calculations are due to [CIP06] and give slightly better bounds than Schuler's original calculations [Sch05]. Let k be a parameter that we'll optimize for. We bound the running time by summing over all the paths in the computation

tree.

$$\sum_{f=0}^{n/k} \binom{m+f}{f} 2^{(n-fk)(1-\frac{1}{k})} \leq 2^{n(1-\frac{1}{k})} \sum_{f=0}^{m+\lfloor n/k \rfloor} \binom{m+\lfloor n/k \rfloor}{f} 2^{-fk(1-\frac{1}{k})}$$
$$= 2^{n(1-\frac{1}{k})} \left(1+2^{-(k-1)}\right)^{m+\lfloor n/k \rfloor} \leq 2^{n(1-\frac{1}{k})} e^{2^{-(k-1)}(m+n/k)}$$
$$\leq 2^{n(1-\frac{1}{k})+4\cdot 2^{-(k-1)}\max(m,n/k)}$$

choose $k = \Theta\left(\max(\lg \frac{m}{n}, 1)\right)$ such that $4 \cdot 2^{-(k-1)} \max(m, n/k) \leq \frac{1}{2k}$

$$\leq 2^{n(1-\frac{1}{2k})} = 2^{n\left(1-\frac{1}{O(1+\lg \frac{m}{n})}\right)}$$

□

We can compose Schuler's reduction (Algorithm 4.4, Theorem 4.6) and our algorithm for enumerating solutions to a k-CNF (Algorithm 4.3, Lemma 4.2) in exactly the same way to get an algorithm for CNF-SAT with m clauses running in expected time $2^{n\left(1-\frac{1}{O(\lg(m/n))}\right)}$. However, since Algorithm 4.3 can also be used to enumerate the satisfying assignments to a k-CNF, we would like to use Schuler's reduction with our algorithm to enumerate the satisfying assignments to a CNF with m clauses. This requires a little bit more work.

First, lets see why the naive composition doesn't work as well as we would like when enumerating solutions. We would like to output a set of restrictions which partition $\{0,1\}^n$ such that under each restriction the CNF is constant. Consider the single clause CNF $(x_1 \vee x_2 \vee x_3 \vee x_4)$ and let $k = 2$. Schuler's algorithm will first branch on $(x_1 \vee x_2)$. When $(x_1 \vee x_2)$ is true, the CNF is equivalent to just $(x_1 \vee x_2)$. Now by constructing the decision tree for this, we would output among other things, the restriction $x_1 = 0, x_2 = 0, x_3 = *, x_4 = *$ since this makes the CNF $(x_1 \vee x_2)$ constant. However, it *doesn't* make the original CNF constant.

In general, when the naive composition outputs a restriction that makes the k-CNF 1, it will also make the original CNF 1 (this is why the composition works for satisfiability). However, when the k-CNF is 0, it can be 0 because the original CNF is 0 or because of the path taken in the computation tree of Schuler's reduction.

We could solve these problems by first running the combined algorithm on F to get a set of restrictions (paths from the decision tree) which set F to 1, and then run the algorithm on $\neg F$ to get a set of restrictions which set $\neg F$ to 1 (equivalently set F to 0), and output these, but we will instead use a different technique which will also be useful in Chapter 5.

Think of each branch in Schuler's reduction as partitioning the space $\{0,1\}^n$ of inputs into smaller and smaller regions. Recall from Section 2.2, a region $\mathcal{R} = (R, \rho)$ where R is a k-CNF and ρ is a restriction is the subset of $\{0,1\}^n$ which satisfies R and is consistent with ρ. In the branch in Schuler's reduction where C' is true, we add C' to R, and in the branch where C' is false, we add $C' = 0$ to ρ. This way, we maintain the invariant that at any point in Schuler's reduction in the current region F is equivalent to the original CNF. When F becomes a k-CNF, we use Algorithm 4.3 to construct a decision tree for the pair (F, R). Now, we can simply output the subset of paths where $R \equiv 1$ since these correspond exactly to the current region. This is formalized in Algorithm 4.5.

Algorithm 4.5: SchulerEnumerate$(F, k, \mathcal{R} = (R, \rho))$

Data: F is a CNF and $k \geq 1$ is a parameter, $\mathcal{R}$ is a region initially all of $\{0,1\}^n$
Result: A set of restrictions which partition $\mathcal{R}$ and make F constant

```
1  begin
2      if F is a k-CNF then
3          foreach path σ in KCNFEnumerate(F, R) do
4              if R|_σ ≡ 1 then
5                  output ρσ
6              end
7          end
8      else
9          let C be the first clause in F of width > k
10         let C' be the conjunction of the first k literals in C
11         SchulerEnumerate(F ∪ {C'} \ {C}, k, (R ∧ C', ρ))   // C' is true
12         SchulerEnumerate(F|_{C'=0}, k, (R, ρ(C' = 0)))     // C' is false
13     end
14 end
```

Lemma 4.8. *Let F be a* CNF *in n variables with m clauses. We may choose k such*

that SchulerEnumerate(F, k) outputs at most s restrictions which partition $\{0,1\}^n$ and make F constant and such that SchulerEnumerate(F, k) runs in expected time $|F|poly(n)s$ where $s \leq 2^{n(1-1/O\left(\lg \frac{m}{n}\right))+O(1)}$.

The algorithm for enumerating solutions to a size m CNF is optimal up to constants in the savings since the corresponding block parity circuit (Section 2.1.2) requires $2^{n-n/\Omega\left(\lg \frac{m}{n}\right)}$ restrictions to partition $\{0,1\}^n$ and make the circuit constant.

Proof. Let F^0 denote the original CNF. We maintain the invariant at every recursive call that $\mathcal{R} \implies (F \equiv F^0)$. This follows by induction from inspection of the two recursive calls. The calculations bounding the number of branches in the computation tree of Schuler's reduction are identical to the proof of Theorem 4.7. □

4.6 Acknowledgments

The content of Section 4.4 and parts of Section 4.5 is joint work with Russell Impagliazzo and Ramamohan Paturi and appears in [IMP11].

Chapter 5

A Satisfiability Algorithm for $\mathbf{AC}^0$

Schuler [Sch05] (Section 4.5) gives an algorithm for CNF-SAT with savings $O(1/\log c)$. He does this by reducing the CNF to a moderately exponential sized set of $O(\log c)$-CNFs through a case analysis, then using known k-SAT algorithms (Chapter 4) to get overall savings $1/O(\log c)$. Like [CIP09], our algorithm can be thought of as generalizing Schuler's approach to larger depths, where each step performs a case analysis to reduce a single circuit of depth d to a moderate-sized collection of circuits which are all depth $d-1$. Calabro et. al. use a complex, but local, form of Schuler's case analysis, branching on large constant sized sub-formulas of the input formula. Here, we use a more global case analysis, based on Håstad's Switching Lemma. This global treatment was in part inspired by an algorithm of Santhanam [San10] for formula satisfiability, that constructs a decision tree for the formula whose paths are short on average, although there might not be a small depth decision tree for the formula.

Using a Switching Lemma to convert depth d circuits to depth $d-1$ circuits is standard. However, to achieve the claimed savings with Switching Lemmas is not at all straight-forward. Assume that we have converted the bottom two levels of our circuit to k-CNFs. The main idea is that after a random restriction setting all but about n/k variables, with high probability, each of these sub-circuits is equivalent to a small depth decision tree. View the random restriction as first picking the set of variables to restrict, then the values. For a randomly chosen set of variables to restrict, our algorithm will perform an exhaustive search over all

settings of the values. If for a certain setting, the sub-formulas all become decision trees of depth k', we can write them all as k'-DNFs and combine with the level of OR gates above. Then we can hope to get the recursive savings on these branches for depth $d-1$ circuits on $n' = \Omega(n/k)$ variables.

So there are two factors that limit our savings with this approach: Even if the formula became constant after every restriction, we cannot get savings more than $1/k$, since we use exhaustive search on $n - n/k$ variables.. On the other hand, there is a failure probability (exponentially small in k') where our sub-formulas might not become small depth decision trees. For these branches , we might not get any savings. So our savings is also bounded by roughly k'/n, the log of the failure probability as a fraction of n. Since our value of k' in this depth becomes the new value of k, it is hard to see how to get savings more than $1/\sqrt{n}$ by this type of argument, even for depth 3.

We get around this by taking a more error-tolerant approach. We don't assume that all of our sub-formulas become small depth decision trees, just most of them. This is still problematic, because the sub-formulas might be identical or close, so the events that they become small depth decision trees might be highly correlated. To handle this, we look not at the decision tree complexity of a single sub-formula but at sets of sub-formulas, where the decision tree has to compute the value of each one. If several formulas become high depth for essentially the same reason, once we've evaluated one of them, the others should become small depth, so the combined decision tree will still have relatively small depth. We give an extended switching lemma that proves that it is (almost) exponentially unlikely that there is a large set of sub-formulas who all contribute many variables to their joint decision tree. Thus, intuitively, with extremely high probability, either only a few sub-formulas remain complex, or the ones that do all involve the same moderate sized subset of variables.

Our algorithm then does a case analysis over which of the sub-formulas are in this set of mutually complex ones, and over all paths in their joint decision tree. For each, the other sub-formulas are equivalent to k-DNFs by definition. Because we are not insisting that all sub-formulas become small, we can pick k relatively

small and still have an extremely small chance of failure. k affects the overhead for the case analysis (we'll have to branch on which set of at most n/k of the m sub-formulas stay complex, which is relatively small for $k = O(\log c)$), which is determined by the length of the joint decision tree. We can let this be a constant fraction of the remaining variables, and so get a failure probability exponentially small in n/k rather than k.

In the next section, we formalize the overall algorithm. In Section 5.2 we describe the algorithm for converting one level of CNFs to DNFs and vice versa, which uses our Extended Switching Lemma, Lemma 1.7 proved in Section 3.3.

5.1 Algorithm Details

The main algorithm consists of a sequence of steps, each of which takes a circuit and computes a partitioning for that circuit where the circuits associated with each region are simpler (either the depth or the bottom fan-in is reduced). The first step reduces the bottom fan-in to k (for some k which will eventually depend on m/n and d). It takes an (n, m, d)-circuit C and outputs a partition for C into regions with associated (n', m, d, k)-circuits. In each subsequent step, we reduce the depth of the circuit by further partitioning. In each such step, an we construct a partition for a (n, m, i, k)-circuit into regions with associated $(\epsilon' n, m, i-1, k)$-circuits. This continues until the depth becomes two at which point we run the algorithm of Lemma 4.2. Each step increases the size of the partition by a factor of $2^{(1-\epsilon)n}$ but decreases the number of variables by a factor ϵ' (where ϵ and ϵ' depend on m, d, and k and will be made precise below.) Both the final step where we handle depth 2 circuits, and the depth reduction step require the circuit to have bounded bottom fan-in. Hence why we needed the first step.

We construct a partition for an (n, m, d)-circuit into regions and associated (n', m, d, k)-circuits by using a technique of Schuler [Sch05] (Section 4.5).

Lemma 5.1 (BottomFaninReduction)**.** *Let C be a (n, m, d)-circuit and let $k \geq 1$ be a parameter. There exists an algorithm which outputs a partitioning $\mathcal{P} = \cup_{0 \leq f \leq n/k} \mathcal{P}_f$ for C, where the sets $\mathcal{P}_f$ are disjoint and for each f, $\mathcal{P}_f$ contains at*

Algorithm 5.1: BottomFanInReduction(C, k)

Data: C is a (n, m, d)-circuit and $k \geq 1$ is a parameter
// Assume that the bottom level gates are OR gates (the AND gate case is symmetric.)

```
1  begin
2      let R be an empty (true) k-CNF and let ρ be a restriction where all
       variables are unset
3      while there exists a bottom level gate φ with fan-in greater than k
       do
4          let φ' denote the disjunction of the first k inputs to φ
5          branch on φ'
6          if φ' = 1 then
7              replace φ with 1 in C
8              R ← R ∧ φ'
9          else
10             remove the first k inputs from φ in C
11             ρ ← ρ ∧ ¬φ'
12         end
13     end
14     output (R = (R, ρ), C)
15 end
```

most $\binom{m+f}{f}$ *regions with associated* $(n - fk, m, d, k)$*-circuits. The algorithm runs in time* $poly(n) \cdot |C| \cdot |\mathcal{P}|$.

The proof of Lemma 5.1 is essentially the same as for Lemma 4.8.

Next, we repeatedly reduce the depth of (n, m, i, k)-circuits by one until it reaches depth 2 (either a k-CNF or a k-DNF). Let Φ be the sequence of subcircuits of an (n, m, i, k)-circuit at depth $i - 1$. Assume without loss of generality that the subcircuits are k-DNFs. The main technical ingredient for depth reduction is an algorithm which constructs a partition which allows us to transform a sequence of k-DNFs into a sequence of equivalent k-CNFs in each region, or vice versa. This algorithm will be described in detail in Section 5.2. We apply this algorithm to transform Φ into sequences of k-CNFs. Since the gates at level $i - 1$ change from $\vee$ to $\wedge$, they may be combined with the gates at level $i - 2$ to reduce the depth by one without increasing the number of gates at any levels $i - 2$ or higher.

Lemma 5.2 (DepthReduction)**.** *Let* C *be a* (n, m, d, k)*-circuit and let* $0 < q \leq 1/2$ *be a parameter. There exists a randomized algorithm which outputs partitioning* $\mathcal{P}$ *for* C *where the circuit associated with each region of* $\mathcal{P}$ *is a* $(\frac{n}{100k}, m, d - 1, k)$*-circuit. With probability at least* $1 - q$, $|\mathcal{P}| \leq s$ *and the algorithm runs in time* $poly(n) \cdot \lg \frac{1}{q} \cdot |C| \cdot s$ *where* $s = \frac{2n}{100k} \cdot 2^{n - \frac{n}{100k} + 3^{-k}m}$.

The depth reduction algorithm follows from Lemma 5.3, given below and proved in Section 5.2.

Lemma 5.3 (SwitchingAlgorithm)**.** *Let* $\Phi = (\phi_1, \ldots, \phi_m)$ *be a sequence of* k-CNF*s in* n *variables and let* $0 < q \leq 1/2$ *be a parameter. There exists a randomized algorithm which takes* Φ *as input and outputs a partitioning* $\mathcal{P}$ *for* Φ *where the circuits associated with each region of* $\mathcal{P}$ *are* k-DNF*s in at most* $\frac{n}{100k}$ *variables. With probability at least* $1 - q$, $|\mathcal{P}| \leq s$ *and the algorithm runs in time* $poly(n) \cdot \lg \frac{1}{q} \cdot |\Phi| \cdot s$ *where* $s \leq \frac{2n}{100k} \cdot 2^{n - \frac{n}{100k} + 3^{-k}m}$.

The case where each ϕ_j is a k-CNF and the circuits in each region of $\mathcal{P}$ are k-DNFs is symmetric.

Proof of Lemma 5.2. By Lemma 5.3, each Ψ_i is a depth 2 circuit in at most $\frac{n}{100k}$ variables. Therefore, each C_i will be a $(\frac{n}{100k}, m, d - 1, k)$-circuit after combining

Algorithm 5.2: DepthReduction(C, q)

Data: C is a (n, m, d, k) and $0 < q \leq 1/2$ is a parameter-circuit

begin
  let $\Phi = (phi_1, \ldots, \phi_m)$ be the sequence of subcircuits rooted at level $d-1$ in C
  foreach $(\mathcal{R}_i, \Psi_i = (\psi_{i,1}, \ldots, \psi_{i,m}))$ output by SwitchingAlgorithm(Φ, q) **do**
    let C_i be the circuit resulting from replacing $\phi_1, \ldots, \phi_m$ with $\psi_{i,1}, \ldots, \psi_{i,m}$ in C and then combining the gates at level $d-2$ and $d-1$ *// these will be the same type of gates*
    output $(\mathcal{R}_i, C_i)$
  end
end

the gates at levels $d-2$ and $d-1$. With probability at least $1-q$, the algorithm of Lemma 5.3 produces $\mathcal{P}$ satisfying $|\mathcal{P}| \leq s$ and runs in time $poly(n) \cdot \lg \frac{1}{q} \cdot |\Phi| \cdot s$ where $s \leq \frac{2n}{100k} \cdot 2^{n - \frac{n}{100k} + 3^{-k}m}$. This algorithm produces a partition of the same size and increases the running time by an additive $O(|C| \cdot |\mathcal{P}|)$. □

This lemma does $d-2$ steps of depth reduction.

Lemma 5.4 (RepeatedDepthReduction)**.** *Let C be a (n, m, d, k)-circuit and let $0 < q \leq 1/2$ be a parameter. There exists a randomized algorithm which outputs a partitioning $\mathcal{P}$ for C where the circuit associated with each region is a $\left(\frac{n}{(100k)^{d-2}}, m, 2, k\right)$-circuit (either a k-CNF or a k-DNF). With probability at least $1-q$, $|\mathcal{P}| \leq s$ and the algorithm runs in time $poly(n) \cdot \lg \frac{1}{q} \cdot |C| \cdot s$ where $s \leq \frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}} 2^{n - \frac{n}{(100k)^{d-2}} + (d-2)3^{-k}m}$.*

Proof. We will prove Lemma 5.4 by induction on d. If $d = 2$, output $((R = 1, \rho = 1), C)$ since C is already a k-CNF or k-DNF and the bounds on $\mathcal{P}$ and the running time holds with probability 1.

If $d > 2$ we assume by induction that we can run this algorithm recursively on circuits of depth $d-1$ and that the recursive call will satisfy the properties of the lemma.

Say that $\mathcal{P}$ is good if $|\mathcal{P}| \leq \frac{2n}{100k} 2^{n(1 - \frac{1}{100k}) + 3^{-k}m}$. By Lemma 5.2, $\mathcal{P}$ is good with probability at least $1 - q/2$. For each i, say that $\mathcal{P}_i$ is good

Algorithm 5.3: RepeatedDepthReduction(C, q)

```
Data: C is a (n, m, d, k)-circuit and 0 < q ≤ 1/2 is a parameter
1  begin
2      if the depth of C is at most 2 then
3          output (R = (R = 1, ρ = 1), C)
4      else
5          let P = {(R_i, C_i)}_i be the set of outputs of
           DepthReduction(C, q/2)
6          foreach (R_i, C_i) ∈ P do
7              let P_i = {(R_{i,j}, C_{i,j})}_j be the set of outputs of
               RepeatedDepthReduction(C_i, q/2^{n+1})
8              foreach {(R_{i,j}, C_{i,j})} ∈ P_i do
9                  output (R_i ∧ R_{i,j}, C_{i,j})
10             end
11         end
12     end
13 end
```

if $|\mathcal{P}_i| \leq \frac{\left(\frac{2n}{100k}\right)^{d-3}}{(100k)^{(d-2)(d-3)/2}} 2^{\left(\frac{n}{100k}\right)(1-\frac{1}{(100k)^{d-3}})+(d-3)3^{-k}m}$ (note that this is the result of a recursive call on a $(\frac{n}{100k}, m, d-1, k)$-circuit). By induction, each $\mathcal{P}_i$ is good independently with probability at least $1 - q/2^{n+1}$. By a union bound, $\mathbf{Pr}\left[\mathcal{P}_i \text{ is good for all } i \mid \mathcal{P} \text{ is good}\right] \geq 1 - q/2$, since if $\mathcal{P}$ is good then the number of $\mathcal{P}_i$s is much less than 2^n. The probability that $\mathcal{P}$ and $\mathcal{P}_i$ for all i are all good is at least $(1 - q/2)(1 - q/2) > 1 - q$. In this case, the total number of outputs is at most

$$\left(\frac{2n}{100k} \cdot 2^{n(1-\frac{1}{100k})+3^{-k}m}\right)\left(\frac{\left(\frac{2n}{100k}\right)^{d-3}}{(100k)^{(d-2)(d-3)/2}} 2^{\left(\frac{n}{100k}\right)(1-\frac{1}{(100k)^{d-3}})+(d-3)3^{-k}m}\right)$$
$$= \frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}} 2^{n(1-\frac{1}{(100k)^{d-2}})+(d-2)3^{-k}m}.$$

□

When we end up with a k-CNF or a k-DNF C in a region defined by a k-CNF R (and a restriction), we run the algorithm of Lemma 4.2 on each (C, R) which outputs a set of restrictions that partition R and make C constant.

Lemma 5.5 (Depth Two Algorithm)**.** *Let C be a k-CNF or k-DNF and R be a k-CNF each in the same n variables and let $0 < q \leq 1/2$ be a parameter. There*

exists a randomized algorithm which outputs a partitioning $\mathcal{P}$ for C in the region R where each region on $\mathcal{P}$ is defined by a restriction and the circuit associated with each region of $\mathcal{P}$ is either 0 or 1. With probability at least $1-q$, $|\mathcal{P}| \leq s$ and the algorithm runs in time $poly(n) \cdot \lg \frac{1}{q} \cdot (|C|+|R|) \cdot s$ where $s \leq 50 \cdot 2^{n\left(1-\frac{1}{30k}\right)}$.

This follows from running Lemma 4.2 $\lg \frac{1}{q}$ times in parallel with independent randomness and outputting the smallest result.

Algorithm 5.4: AC0Enumerate(C)

```
   Data: C is a (n, m, d)-circuit
 1 begin
 2     let k = Θ(max(lg m/n, d lg d)) as in the proof of Theorem 1.2
 3     foreach (R_i = (R_i, ρ_i), C_i) output by BottomFaninReduction(C, k)
       do
 4         foreach (R_{i,j} = (R_{i,j}, ρ_{i,j}), C_{i,j}) output by
           RepeatedDepthReduction(C_i, 2^{-2n}) do
 5             let T be the smallest tree constructed after running
               KCNFEnumerate(C_{i,j}, R_i ∧ R_{i,j}) poly(n) times with
               independent randomness
 6             foreach path ρ_{i,j,ℓ} ending at a leaf (b_C, b_R) in T do
 7                 if b_R = 1 then
 8                     output ρ_i ρ_{i,j} ρ_{i,j,ℓ}
 9                 end
10             end
11         end
12     end
13 end
```

We compose the preceding algorithms to get our algorithm for $\mathbf{AC}^0$ circuits, Algorithm 5.4, formalized in the following lemma.

Lemma 5.6. *Let C be a (n,m,d)-circuit. Let $k \geq 1$ be a parameter. There exists a randomized algorithm which outputs a partitioning $\mathcal{P}$ for C where each region is defined by a restriction and the circuit associated with each region is either 0 or 1. With probability at least $1-2^{-n}$, $|\mathcal{P}| \leq s$ and the algorithm runs in time at most $poly(n) \cdot |C| \cdot s$ where $s \leq 50 \frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}} 2^{n - \frac{3n}{(100k)^{d-1}} + (d-2)3^{-k}m + 4 \cdot 2^{-k} \max(m, n/k)}$.*

Theorem 1.2 follows straightforwardly from Lemma 5.6.

Theorem (Theorem 1.2, restated)**.** *There exists a randomized algorithm which takes a cn size and depth d $\mathbf{AC}^0$ circuit C in n variables as an input and outputs a set of restrictions which partition $\{0,1\}^n$ and make C constant. The algorithm outputs at most $2^{n(1-\mu_{c,d})}$ restrictions and runs in time $poly(n)|C|2^{n(1-\mu_{c,d})}$ with probability at least $1-2^{-n}$ where savings is at least $\mu_{c,d} \geq \frac{1}{O(\lg c + d \lg d)^{d-1}}$.*

Proof of Theorem 1.2. Let C be a (n,m,d)-circuit. If $m \geq 2^{\Omega(n)^{\frac{1}{d-1}}}$ then output all 2^n restrictions which set all of the variables. Otherwise, we may choose $k = \Theta\left(\max\left(\lg \frac{m}{n}, d \lg d\right)\right)$ such that

$$50\frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}}2^{(d-2)3^{-k}m+4\cdot 2^{-k}\max(m,n/k)} \leq 2^{\frac{2n}{(100k)^{d-1}}}$$

and then use Lemma 5.6. In either case, the algorithm outputs at most $2^{n-\frac{n}{O(\lg \frac{m}{n}+d\lg d)^{d-1}}+O(1)}$ restrictions. □

The algorithm for enumerating solutions to a (n,m,d)-circuit is almost optimal up to constants in the savings since the corresponding block parity circuit (Section 2.1.2) requires $2^{n-n/\Omega\left(\lg \frac{m}{n}\right)^{d-1}}$ restrictions to partition $\{0,1\}^n$ and make the circuit constant.

For the sake of simplifying the calculations in the proof of Lemma 5.6, we first prove the special case where the circuit has bottom fan-in k.

Lemma 5.7. *Let C be a (n,m,d,k)-circuit. There exists a randomized algorithm which outputs a partitioning $\mathcal{P}$ for C where each region is defined by a restriction and the circuit associated with each region is either 0 or 1. With probability at least $1-2^{-2n}$, $|\mathcal{P}| \leq s$ and the algorithm runs in time at most $poly(n)\cdot|C|\cdot s$ where $s \leq 50\frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}}2^{n-\frac{3n}{(100k)^{d-1}}+(d-2)3^{-k}m}$.*

Proof. Run the RepeatedDepthReduction (Lemma 5.4) on $(C, q/2)$ to get a partition $\mathcal{P} = \{((R_i,\rho_i),C_i)\}_i$ for C. Each C_i is either a k-CNF or a k-DNF in $\frac{n}{(100k)^{d-2}}$ variables. Run the Depth Two Algorithm (Lemma 5.5) on $(C_i, R_i, q = q/2^{n+1})$ to get a partition $\mathcal{P}_i = \{(\rho_{i,j}, b_{i,j})\}_j$ for C_i. Output $(\rho_i \wedge \rho_{i,j}, b_{i,j})$.

Say that $\mathcal{P}$ is good if $|\mathcal{P}| \leq \frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}}2^{n(1-\frac{1}{(100k)^{d-2}})+(d-2)3^{-k}m}$. By Lemma 5.4, $\mathbf{Pr}[\mathcal{P} \text{ is good}] \geq 1-q/2$. For each i, say that $\mathcal{P}_i$ is good if $|\mathcal{P}_i| \leq$

$50 \cdot 2^{\frac{n}{(100k)^{d-2}}(1-\frac{1}{30k})}$ (note that the Depth Two Algorithm is run on circuits in $\frac{n}{(100k)^{d-2}}$ variables). By Lemma 5.5, $\mathbf{Pr}[\mathcal{P}_i \text{ is good}] \geq 1 - q/2^{n+1}$. By a union bound, $\mathbf{Pr}[\mathcal{P}_i \text{ is good for all } i \mid \mathcal{P} \text{ is good}] \geq 1 - q/2$, since if $\mathcal{P}$ is good then the number of $\mathcal{P}_i$s is much less than 2^n. The probability that $\mathcal{P}$ and $\mathcal{P}_i$ are all good is at least $(1 - q/2)(1 - q/2) > 1 - q$. In this case, the total number of outputs is at most

$$\left(\frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}} 2^{n(1-\frac{1}{(100k)^{d-2}})+(d-2)3^{-k}m}\right)\left(50 \cdot 2^{\frac{n}{(100k)^{d-2}}(1-\frac{1}{30k})}\right)$$
$$\leq 50\frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}} 2^{n(1-\frac{3}{(100k)^{d-1}})+(d-2)3^{-k}m}.$$

□

Proof of Lemma 5.6. Run the BottomFaninReduction (Lemma 5.1) on (C, k) to get $\mathcal{P} = \cup_{0 \leq f \leq n/k} \mathcal{P}_f$ where $\mathcal{P}_f = \{((R_{f,i}, \rho_{f,i}), C_{f,i})\}_i$. For each f and i, the circuit $C_{f,i}$ is an $(n - fk, m, d, k)$-circuit. Run the algorithm of Lemma 5.7 on $(C_{f,i}, q = 2^{-2n})$ to get a partition $\mathcal{P}_{f,i} = \{(\rho_{f,i,j}), b_{f,i,j})\}_j$ for $C_{f,i}$. Output $(\rho_{f,i} \wedge \rho_{f,i,j}, b_{f,i,j})$.

By Lemma 5.1, the sets $\mathcal{P}_f$ are disjoint and for each f, $|\mathcal{P}_f| \leq \binom{m+f}{f}$ and the circuits associated with each region in $\mathcal{P}_f$ are $(n - fk, m, d, k)$-circuits. For each f and i, say that $\mathcal{P}_{i,f}$ is good if $|\mathcal{P}_{f,i}| \leq 50\frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}} 2^{(n-fk)(1-\frac{3}{(100k)^{d-1}})+(d-2)3^{-k}m}$. By Lemma 5.7, $\mathbf{Pr}[\mathcal{P}_{f,i} \text{ is good}] \geq 1 - 2^{-2n}$, and by a union bound over the at most 2^n pairs f, i, all of the sets $\mathcal{P}_{f,i}$ are simultaneously good with probability at least $1 - 2^{-n}$. In this case, the total number of outputs is at most

$$\sum_{f=0}^{n/k} \binom{m+f}{f} 50\frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}} 2^{(n-fk)(1-\frac{3}{(100k)^{d-1}})+(d-2)3^{-k}m}$$
$$= 50\frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}} 2^{n(1-\frac{3}{(100k)^{d-1}})+(d-2)3^{-k}m}$$
$$\cdot \sum_{f=0}^{n/k} \binom{m+f}{f} 2^{-fk(1-\frac{3}{(100k)^{d-1}})+(d-2)3^{-k}m}$$
$$\leq 50\frac{(2n)^{d-2}}{(100k)^{(d-1)(d-2)/2}} 2^{n(1-\frac{3}{(100k)^{d-1}})+(d-2)3^{-k}m+4\cdot 2^{-k}\max(m,n/k)}$$

since

$$\sum_{f=0}^{n/k} \binom{m+f}{f} 2^{-fk(1-\frac{3}{(100k)^{d-1}})} \leq \sum_{f=0}^{m+n/k} \binom{m+n/k}{f} 2^{-fk} = \left(1+2^{-k}\right)^{m+n/k}$$
$$\leq 2^{(\lg e)2^{-k}(m+n/k)} \leq 2^{4\cdot 2^{-k}\max(m,n/k)}$$

□

5.2 Switching Algorithm

Lemma (Lemma 5.3, restated)**.** *Let $\Phi = (\phi_1, \ldots, \phi_m)$ be a sequence of k-CNFs in n variables and let $0 < q \leq 1/2$ be a parameter. There exists a randomized algorithm which takes Φ as input and outputs a partitioning $\mathcal{P}$ for Φ where the circuits in each region of $\mathcal{P}$ are k-DNFs in at most $\frac{n}{100k}$ variables. With probability at least $1-q$, $|\mathcal{P}| \leq s$ and the algorithm runs in time at most $poly(n) \cdot \lg \frac{1}{q} \cdot |\Phi| \cdot s$ where $s \leq \frac{2n}{100k} 2^{n-\frac{n}{100k}+3^{-k}m}$.*

Let ϕ be a k-CNF. Consider the decision tree $tree(\phi)$ for ϕ. If the height of $tree(\phi)$ is at most k', then we can construct a k'-DNF ϕ' equivalent to ϕ by taking the disjunction of the terms corresponding to paths in $tree(\phi)$ labeled 1.

However, in general, k' will be much larger than k. In this case, we split the paths in $tree(\phi)$ into two categories: "short paths" of length at most k, and "long paths" of length greater than k. Since any assignment to a set variables is consistent with exactly one path in any decision tree on those variables, we can partition $\{0,1\}^n$ by partitioning the paths. We will construct a k-CNF $\mathcal{T}(\phi, k)$ which will define the region corresponding to the set of short paths. Rather than defining a single region for the set of long paths, we further partition the space and define a separate region for each long path (each long path viewed as a restriction defines a region).

Formally, for any k-CNF ϕ and any $k \geq 1$, let $\Sigma' = \{\sigma'_1, \ldots, \sigma'_{\ell'}\}$ be the set of paths of length greater than k in $tree(\phi)$. Let $\Sigma = \{\sigma_1, \ldots, \sigma_\ell\}$ be the set of paths of length k in $tree(\phi)$ that do not end at a leaf (equivalently, Σ consists of the paths in Σ' truncated after k variables). Define $\mathcal{T}(\phi, k) = \neg\sigma_1 \wedge \neg\sigma_2 \wedge \cdots \wedge \neg\sigma_\ell$

Algorithm 5.5: SwitchingAlgorithm($\Phi = (\phi_1, \ldots, \phi_m), q$)

Data: $\phi_1, \ldots, \phi_m$ are k-CNFs in the same n variables and $0 < q \leq 1/2$ is a parameter

begin
  repeat $\lg \frac{1}{q}$ **times** *// output the smallest partition from a single iteration*
    let $p = \frac{1}{100k}$
    choose a set U of pn variables uniformly at random
    foreach restriction ρ_0 which leaves the variables in U unset **do**
      $\rho \leftarrow \rho_0$
      $R \leftarrow 1$
      for $i \leftarrow 1, \ldots, m$ **do**
        let $T = tree(\phi_i|_\rho)$
        branch on $\mathcal{T}(\phi_i|_\rho, k)$ and each path ρ' in T of length $> k$
        if $\mathcal{T}(\phi_i|_\rho, k)$ **then** *// the $\mathcal{T}(\phi_i|_\rho, k)$ branch*
          $\psi_i \leftarrow \mathcal{S}(\phi_i|_\rho, k)$
          $R \leftarrow R \wedge \mathcal{T}(\phi_i|_\rho, k)$
        else *// a ρ' branch*
          $\psi_i \leftarrow \phi_i|_{\rho\rho'}$ *// equivalently, the bit labeling the end of the path ρ' in T*
          $\rho \leftarrow \rho\rho'$
        end
      end
      output $(\mathcal{R} = (R, \rho), \Psi = (\psi_1, \ldots, \psi_m))$
    end
  end
end

where each σ_i is viewed as the conjunction of the literals along the path. Note that $\mathcal{T}(\phi, k)$ is a k-CNF. An assignment is in the region corresponding to short paths if and only if it is not consistent with any σ'_i. Since the paths σ'_i form complete decision trees after the first k variables along each path, an assignment is not consistent with any σ'_i if and only if it is not consistent with any σ_i, and therefore if and only if it satisfies $\mathcal{T}(\phi, k)$.

The algorithm will branch on the region $\mathcal{T}(\phi, k)$ and on the regions corresponding to long paths. In the region $\mathcal{T}(\phi, k)$, we can do as we did before and convert ϕ into an equivalent k-DNF by only considering the short paths ending with 1. Formally, for any k-CNF ϕ and any $k \geq 1$, let $\mathcal{S}(\phi, k) = \tau_1 \vee \tau_2 \vee \cdots \vee \tau_\ell$ where $\tau_1, \ldots, \tau_\ell$ are the paths of the decision tree for ϕ of length at most k with label 1 where each τ_i is viewed as a conjunction of the literals along the path. $\mathcal{S}(\phi, k)$ is a k-DNF and is equivalent to ϕ in the region $\mathcal{T}(\phi, k)$. In the regions correspond to long paths, ϕ is constant and therefore trivially a k-DNF.

We repeat this process for each k-CNF $\phi_1, \ldots, \phi_m$ in order, each time recursively partitioning the current set of regions.

The problem with this approach is that we may well branch more than 2^n times between branching on $\mathcal{T}(\phi, k)$ and branching on each long path. We solve this problem by first applying a random restriction which leaves a small constant fraction of the variables unset, and then do depth reduction as described above. In this case, we can bound the expected number of long paths using Lemma 1.7.

Proof. Let $p = \frac{1}{100k}$. Choose a set U of pn variable to leave unset uniformly at random. Branch on each restriction ρ_0 which leaves the variables in U unset. Let $\rho = \rho_0$ and let R denote an empty (true) k-CNF. For each ϕ_i in order, branch on $\mathcal{T}(\phi_i|_\rho, k)$.

In the region $\rho \wedge \mathcal{T}(\phi_i|_\rho, k)$, ϕ_i is equivalent to $\mathcal{S}(\phi_i|_\rho, k)$ so set $\psi_i = \mathcal{S}(\phi_i|_\rho, k)$. Let $R = R \wedge \mathcal{T}(\phi_i|_\rho, k)$. In this branch, say that ϕ_i is "not targeted."

When $\mathcal{T}(\phi_i|_\rho, k)$ is false, we further branch on each path ρ' in $tree(\phi_i|_\rho)$ of length greater than k. Let $\rho = \rho \wedge \rho'$ and let $\psi_i = b'$ where b' is the label at the end of the path ρ'. In each of these branches, say that ϕ_i is "targeted."

Once we have branched in this fashion for each ϕ_i, output the resulting

$(\mathcal{R} = (R, \rho), \Psi = (\psi_1, \ldots, \psi_m))$.

This algorithm naturally defines a computation tree: First branch on each restriction ρ_0, then for each ϕ_i in order branch on whether ϕ_i is targeted, and if ϕ_i is targeted branch on each long path. Each leaf of this computation tree correspond to a region in partition output. Define the "type" of each leaf as the sequence of targeted k-CNFs along the path to the leaf. We will group the leaves of the computation tree both by type and by parent restriction ρ_0 in order to bound the expected number of leaves.

For any ρ_0 and any type (sequence of targeted k-CNFs) T, let $G_{\rho_0,T}$ denote the set of leaves of type T with parent restriction ρ_0. Consider the decision tree $tree(T|_{\rho_0})$. Let $P_{\rho_0,T}$ denote the set of paths in $tree(T|_{\rho_0})$ where each k-CNF in T contributes at least $k+1$ variables to the path. The set of paths $P_{\rho_0,T}$ correspond exactly to the leaves in $G_{\rho_0,T}$: The path in $P_{\rho_0,T}$ corresponds to which branch ρ' is taken at each targeted k-CNF (each ρ' sets at least $k+1$ variables since only paths of length $> k$ are considered when branching).

By Lemma 1.7, $\mathbf{E}_{U,\rho_0}$ [number of paths in $tree(T|_{\rho_0})$ of length s where each k-CNF in T contributes at least one variable] $\leq (13/100)^s$, which gives $\mathbf{E}_{U,\rho_0}[|P_{\rho_0,T}|] \leq \sum_{s=0}^{pn}(13/100)^s$. Since we only consider paths in $tree(T|_{\rho_0})$ where each $\phi_i \in T$ contributes at least $k+1$ variables, any such path must have length at least $|T|(k+1)$. We bound the expected number of outputs by summing over restrictions ρ_0, then path lengths s and then sets of targeted k-CNFs T of size at

most $s/(k+1)$.

$$\begin{aligned}
\mathop{\mathbf{E}}_{U}[|\mathcal{P}|] &= \sum_{\rho_0} \sum_{T\subseteq\{\phi_1,\ldots,\phi_m\}} \mathop{\mathbf{E}}_{U}[|P_{\rho_0,T}|] = 2^{n-pn} \sum_{T\subseteq\{\phi_1,\ldots,\phi_m\}} \mathop{\mathbf{E}}_{U,\rho_0}[|P_{\rho_0,T}|] \\
&\leq 2^{n-pn} \sum_{T\subseteq\{\phi_1,\ldots,\phi_m\}} \sum_{s=0}^{pn} \mathop{\mathbf{E}}_{U,\rho_0} \begin{bmatrix}\text{number of paths in } tree(T|_{\rho_0}) \text{ of} \\ \text{length } s \text{ where each } \phi_i \in T \text{ con-} \\ \text{tributes} > k \text{ variables}\end{bmatrix} \\
&= 2^{n-pn} \sum_{s=0}^{pn} \sum_{\substack{T\subseteq\{\phi_1,\ldots,\phi_m\} \\ |T|\leq s/(k+1)}} \mathop{\mathbf{E}}_{U,\rho_0} \begin{bmatrix}\text{number of paths in } tree(T|_{\rho_0}) \text{ of} \\ \text{length } s \text{ where each } \phi_i \in T \text{ con-} \\ \text{tributes} > k \text{ variables}\end{bmatrix} \\
&\leq 2^{n-pn} \sum_{s=0}^{pn} \sum_{t=0}^{\lfloor s/k\rfloor} \binom{m}{t} (13/100)^s \leq 2^{n-pn} \sum_{s=0}^{pn} \left\lfloor \frac{s}{k} \right\rfloor \binom{m}{\lfloor s/k\rfloor} (13/100)^s
\end{aligned}$$

grouping terms with the same value of $\lfloor s/k\rfloor$ and replacing $\lfloor s/k\rfloor$ with s'

$$\begin{aligned}
&\leq 2^{n-pn} \sum_{s'=0}^{pn/k} ks' \binom{m}{s'} (13/100)^{ks'} \leq pn 2^{n-pn} \sum_{s'=0}^{m} \binom{m}{s'} (13/100)^{ks'} \\
&= \frac{n}{100k} 2^{n-pn} (1+(13/100)^k)^m \leq \frac{n}{100k} 2^{n-pn+(\lg e)(13/100)^k m}
\end{aligned}$$

with probability at least $1/2$

$$|\mathcal{P}| \leq \frac{2n}{100k} \cdot 2^{n-\frac{n}{100k}+3^{-k}m}$$

We may repeat the algorithm $\lg \frac{1}{q}$ times in parallel with independent choices of U and output the smallest partition. This increases the probability of success to $1-q$. □

5.3 Acknowledgements

The content of this chapter is joint work with Russell Impagliazzo and Ramamohan Paturi and appears in [IMP11].

Chapter 6

Uniquely Satisfiable k-sat Instances with Almost Minimal Occurrences of Each Variable

6.1 Introduction

Let (k, s)-CNF refer to CNF formulas with exactly k distinct literals per clause and at most s occurrences of each variable. Let (k, s)-SAT refer to the family of satisfiability problems restricted to (k, s)-CNF formulas. Since $(2, s)$-SAT is in **P** for all s, we restrict our attention to $k \geq 3$.

Tovey [Tov84] first observed that $(3, 3)$-SAT was trivial since every instance is satisfiable, and showed that $(3, 5)$-SAT was **NP**-hard. This was generalized to larger k by Kratochvíl, Savický and Tuza [KST93] who showed that for each $k \geq 4$ there exists a threshold $f(k)$ such that for all $s \leq f(k)$, (k, s)-SAT is trivial whereas for all $s > f(k)$, (k, s)-SAT is **NP**-hard.

Using Hall's Theorem, Tovey [Tov84] showed that every (k, k)-SAT instance is satisfiable, giving the first lower bound $f(k) \geq k$. This was improved by Kratochvíl, Savický and Tuza [KST93] who used the Lovász local lemma to show that all $(k, \lfloor 2^k/ek \rfloor)$-SAT instances are satisfied by random assignments with positive probability, implying $f(k) \geq \lfloor 2^k/ek \rfloor$.

Trivially, $f(k) < 2^k$ since enumerating all 2^k possible clauses for k variables gives an unsatisfiable formula. Kratochvíl, Savický and Tuza [KST93] proved that $f(k+1) \leq 2f(k)+1$. Combined with the fact that $f(3) = 3$, we get $f(k) \leq 2^{k-1} - 1$ (this may be improved slightly by using a base case of $f(k)$ for larger k). Subsequently, this has been improved to $f(k) = \Theta\left(2^k/k\right)$ [Geb09]. However the exact value of $f(k)$, or even whether $f(k)$ is computable, remains unknown.

Valiant and Vazirani [VV86] showed that deciding whether a SAT formula has zero or one solution is essentially as hard as SAT in general. In particular, they prove the following theorem:

Theorem 6.1 (Valiant–Vazirani Theorem [VV86])**.** *There exists a randomized polynomial time reduction from* SAT *to* UNIQUE-SAT*.*

By the standard parsimonious reduction from SAT to k-SAT, the Valiant–Vazirani Theorem implies the same hardness for k-UNIQUE-SAT. However, what happens when the number of occurrences of each variable is also limited? Specifically, what can be said about (k, s)-UNIQUE-SAT for various values of s?

We give a parsimonious reduction from 3-SAT to (k, s)-SAT, for any $k \geq 3$ and $s \geq f(k) + 2$. Thus, $(k, f(k) + 2)$-UNIQUE-SAT is as hard as UNIQUE-SAT. In contrast, $(k, f(k))$-UNIQUE-SAT is trivial since every formula is satisfiable.

Calabro et al. [CIKP08b] give additional evidence that k-UNIQUE-SAT is no easier than k-SAT, not just for polynomial time algorithms (as shown by Valiant and Vazirani), but for super-polynomial time algorithms. They show that if 3-UNIQUE-SAT is in randomized subexponential time ($\cap_{\epsilon>0}\mathbf{RTIME}[2^{\epsilon n}]$), then so is k-SAT for all $k \geq 3$. Our parsimonious reduction from 3-SAT to (k, s)-SAT combined with their result implies that if (k, s)-UNIQUE-SAT is in randomized subexponential time for some $k \geq 3$ and $s \geq f(k) + 2$, then so is k'-SAT for all $k' \geq 3$. We omit the details which follow fairly straightforwardly from [CIKP08b, IPZ01].

A key component in our reduction is a construction of uniquely satisfiable $(k, s+1)$-CNF formulas from unsatisfiable $(k, s+1)$-SAT formulas. Starting with unsatisfiable $(k, f(k)+1)$-CNF formulas allows us work with uniquely satisfiable formulas with almost the minimum number of occurrences of each variable, and also argue about the transition where uniquely satisfiable formulas first occur.

Since the smallest s we argue about for each k is $f(k)+2$, the questions of whether there exists a uniquely satisfiable $(k, f(k)+1)$-CNF formula and the complexity of $(k, f(k)+1)$-UNIQUE-SAT remain open.

Since our reduction requires the existence of an unsatisfiable (k, s)-CNF formula, we require that k and $s > f(k)$ be constants. In this case we know there exists an unsatisfiable formula of constant size. If we could give an upper bound on the size of this formula in terms of k and s it would imply that $f(k)$ was computable, which would be an independently interesting result.

Let $F_k(n, m)$ refer to a random k-CNF formula with n variables and m clauses. Just as there is a transition in (k, s)-SAT as s increases from trivial to **NP**-hard, there is a similar transition in $F_k(n, rn)$ as r increases from satisfiable with high probability to unsatisfiable w.h.p. (See [AP04] and its references). It is conjectured that for each k the transition occurs at a sharp threshold r_k. Achlioptas and Ricci-Tersenghi [ART06] show that for sufficiently large k and $r < r_k$, w.h.p., $F_k(n, rn)$ has exponentially many, widely separated, small clusters of solutions. In some ways, small, widely separated clusters of solutions are similar to unique solutions. In both cases, they seem to be some of the hardest instances for algorithms. While we don't consider random SAT formulas in this paper, we view the similarities as additional motivation.

6.2 Definitions and Results

Definition 6.2. We let UNIQUE-SAT refer to the promise problem of deciding whether a CNF formula is unsatisfiable or has a unique satisfying assignment. k-UNIQUE-SAT and (k, s)-UNIQUE-SAT are defined similarly.

Definition 6.3 (Valiant and Vazirani [VV86])**.** A *randomized polynomial time reduction* M from a problem A to a problem B is a randomized polynomial time Turing machine such that for all $x \notin A$ we are guaranteed that $M(x) \notin B$, and for all $x \in A$ we get $M(x) \in B$ with probability at least $1/poly(|x|)$.

Definition 6.4. In the context of SAT, a reduction M is said to be *parsimonious* if the formulas x and $M(x)$ have the same number of satisfying assignments.

In particular, parsimonious reductions preserve the existence of unique satisfying assignments.

Definition 6.5 (Kratochvíl, Savický and Tuza [KST93])**.** For each $k \geq 3$, $f(k)$ is defined as the largest value of s such that all (k, s)-SAT instances are satisfiable.

Equivalently, we may think of $f(k) + 1$ as the smallest value of s such that there exist unsatisfiable (k, s)-SAT instances.

Definition 6.6. For each $k \geq 3$, we define $u(k)$ as the smallest value of s such that there exist (k, s)-SAT instances with exactly one satisfying assignment.

Theorem 6.7. *For all $k \geq 3$, $f(k) \leq u(k) \leq f(k) + 2$.*

This theorem follows directly from the following two lemmas:

Lemma 6.8. *For all $k \geq 3$ and $s \geq u(k)$, there exist unsatisfiable $(k, s+1)$-SAT instances.*

Lemma 6.9. *For all $k \geq 3$ and $s \geq f(k) + 1$, there exist uniquely satisfiable $(k, s+1)$-SAT instances.*

To prove that$(k, f(k) + 1)$-SAT is **NP**-hard, Kratochvíl, Savický and Tuza [KST93] give a reduction from k-SAT to (k, s)-SAT for any $s > f(k)$. Combining their proof with Lemma 6.9 we get the following lemma:

Lemma 6.10. *For any constants $k \geq 3$ and $s \geq f(k)+2$, there is a* parsimonious *polynomial time reduction from* 3-SAT *to* (k, s)-SAT*.*

Composing the Valiant–Vazirani Theorem (Theorem 6.1), the standard parsimonious reduction from SAT to 3-SAT, and Lemma 6.10, we get the following:

Corollary 6.11. *For any constants $k \geq 3$ and $s \geq f(k)+2$, there is a randomized polynomial time reduction from* SAT *to* (k, s)-UNIQUE-SAT*.*

6.3 Proofs

Lemma 6.8. Since $s \geq u(k)$, there exists a uniquely satisfiable (k, s)-CNF formula F. Add a single clause to F which is violated by the unique satisfying assignment. We add at most 1 occurrence of each variable, so this gives an unsatisfiable $(k, s+1)$-CNF formula. □

To prove Lemma 6.9, we will construct a uniquely satisfiable $(k, s+1)$-CNF formula in a sequence of steps from an unsatisfiable (k, s)-CNF formula. We classify variables in each of these formulas as either *forced* or *unforced*. If every satisfying assignment for a formula sets a variable to the same value, we say that the variable is forced. Otherwise, we say that the variable is unforced. We will be particularly interested in forced variables that occur exactly once in the formula. Without loss of generality, we will always assume that forced variable must be set to false in all satisfying assignments (otherwise replace every occurrence of the variable with its negation). Note that uniquely satisfiable formulas are equivalent to formulas where every variable is forced.

Our construction can be broken down into 3 steps formalized by the following lemmas: Lemma 6.12 constructs a formula with a few forced variables. Lemma 6.13 increases the number of forced variables without increasing the number of unforced variables. Lemma 6.14 uses the newly created forced variables to force all of the unforced variables.

Lemma 6.12. *We can transform an unsatisfiable* (k, s)-CNF *formula into a satisfiable* (k, s)-CNF *formula with* k *forced variables that only occur once.*

Lemma 6.13. *We can transform a* (k, s)-CNF *with* n *unforced variables and* $t \geq k - 1$ *forced variables that only occur once (and possibly other variables that are forced but occur more than once) into a* (k, s)-CNF *with* n *unforced variables and* $t + (s - k) > t$ *forced variables that only occur once.*

Lemma 6.14. *We can transform a* (k, s)-CNF *with* n *unforced variables and at least* $n + k$ *forced variables that only occur once into a* $(k, s+1)$-CNF *where every variable is forced.*

Lemma 6.12. Let F be a minimal unsatisfiable (k, s)-CNF. (A formula is minimally unsatisfiable if removing any clause would make it satisfiable.) Transform F by renaming variables and replacing variables with their negations so that F can be written as $(x_1 \vee x_2 \vee \cdots \vee x_k) \wedge G$, where G is satisfied by the all-false assignment. Within G, the variables $x_1, \ldots, x_k$ each occur at most $s-1$ times and are forced to false (any satisfying assignment that didn't set them to false would also satisfy F).

Let $G^{(1)}, \ldots, G^{(k-1)}$ be $k-1$ disjoint copies of G. Let $x_1^{(i)}, \ldots, x_k^{(i)}$ denote the copy of $x_1, \ldots, x_k$ occurring in $G^{(i)}$. Return the formula $G^{(1)} \wedge \cdots \wedge G^{(k-1)} \wedge H$, where $H = \bigwedge_{i=1}^{k}(x_i^{(1)} \vee \cdots \vee x_i^{(k-1)} \vee \overline{y_i})$ and $y_1, \ldots, y_k$ are fresh variables. Each variable y_i occurs in exactly 1 clause and must be set to false to satisfy that clause since all of the other variables in the clause are already forced. □

Lemma 6.13. Let G be a (k, s)-CNF with n unforced variables and $t \geq k-1$ forced variables that only occur once. Let $y_1, \ldots, y_{k-1}$ denote $k-1$ of these forced variables. Let $H = \bigwedge_{i=1}^{s-1}(y_1 \vee \cdots \vee y_{k-1} \vee \overline{z_i})$, where $z_1, \ldots z_{s-1}$ are fresh variables. Return the formula $G \wedge H$. Each of the variables $y_1, \ldots, y_{k-1}$ is still forced, but now each occurs s times. In their place, we have $s-1$ new forced variables $z_1, \ldots, z_{s-1}$ which each only occur once, for a total of $t+(s-k)$ such variables. Whether other variables are forced remains unchanged. Note that $s > f(k) \geq k$ since unsatisfiable (k, s)-CNFs exist. □

Lemma 6.14. Let F be a (k, s)-CNF with n unforced variables and $n+k$ forced variables that only occur once. Let $x_1, \ldots, x_n$ denote the unforced variables. Let $y_1, \ldots, y_{n+k}$ denote the forced variables that only occur once. Let $m = \lceil \frac{n}{k-1} \rceil$. Arbitrarily partition the variables $x_1, \ldots, x_n$ into m sets $X_1, \ldots, X_m$ of size $k-1$. Add new variables as needed so that every set contains exactly $k-1$ variables. Arbitrarily partition the variables $y_1, \ldots, y_{(k-1)m}$ into m sets $Y_1, \ldots, Y_m$ of size $k-1$.

For each $1 \leq i \leq m$, we will construct a formula H_i using the variables in sets X_i and Y_i. For simplicity, let $X_i = \{x_1, \ldots, x_{k-1}\}$ and $Y_i = \{y_1, \ldots, y_{k-1}\}$. For each i, let $H_i = \bigwedge_{j=1}^{k-1}(y_1 \vee \cdots \vee y_{k-1} \vee \overline{x_j})$.

Return the formula $F \wedge H_1 \wedge \cdots \wedge H_m$. Each H_i uses the variables in Y_i to force the variables in X_i. Since each variable in Y_i is forced, the variables in X_i must be false to satisfy the clauses in H_i. This adds $k-1$ occurrences for each variable in Y_i and one occurrence for each variable in X_i. Each variable in Y_i now occurs $k < s$ times and each variable in X_i now occurs at most $s+1$ times. □

Lemma 6.9. Since $s \geq f(k)+1$, there exists an unsatisfiable (k,s)-CNFa F. Use Lemma 6.12 to construct a (k,s)-CNF G with k forced variables that only occur once. Let n denote the number of unforced variables in G. Use Lemma 6.13 sufficiently many times starting with G to get a formula H with at least $n+k$ forced variables that only occur once. Note that H still contains only n unforced variables. Using Lemma 6.14 on H gives a uniquely satisfiable $(k,s+1)$-CNF. □

By repeating Lemma 6.13 sufficiently many additional time before using Lemma 6.14, we get the following corollary:

Corollary 6.15. *For any constants $k \geq 3$ and $s \geq f(k)+2$, and any $m \geq 0$, we can construct a uniquely satisfiable (k,s)-CNF with at least m forced variables that only occur once in time polynomial in m.*

The following proof of Lemma 6.10 is the same as the reduction given by Kratochvíl, Savický and Tuza [KST93] to prove that $(k, f(k)+1)$-SAT is **NP**-hard with one exception. We use Corollary 6.15 to supply forced variables whereas they used a $(k, f(k)+1)$-CNF with potentially many satisfying assignments.

Lemma 6.10. For any $k \geq 3$ and $s \geq f(k)+2$, we transform a 3-CNF F parsimoniously into a (k,s)-CNF in 2 steps:

First, we reduce the number of occurrences of each variable to at most s, which introduces additional 2-variable clauses. For each variable x occurring $t > s$ times, replace each occurrence of x with a new variable x_i, $1 \leq i \leq t$. Add clauses $(\overline{x_i} \vee x_{i+1})$ for $1 \leq i \leq t-1$, and $(\overline{x_t} \vee x_1)$. These clauses ensure that in any satisfying assignment all the variables x_i are assigned the same value. Thus, we maintain exactly the same number of satisfying assignments. Each of these new variables occurs exactly $3 < s$ times. Let G denote the resulting formula, and m the number of clauses in G.

Second, we pad each clause with forced variables so that all clauses contain exactly k variables. Using Corollary 6.15, there exists a (k, s)-CNF H with at least mk forced variables that only occur once. For each clause c of length $\ell < k$ in G, replace c with $(c \vee y_1 \vee \cdots \vee y_{k-\ell})$, where $y_1, \ldots, y_{k-\ell}$ are arbitrary forced variables from H occurring fewer than s times. Let G' denote the result of these replacements. Return the formula $G' \wedge H$. Since the only satisfying assignment to H sets all variables to false, the padded clauses in G' are satisfied by exactly the same assignments that satisfy G. Thus, $G' \wedge H$ has exactly the same number of satisfying assignments as G. □

6.4 Acknowledgments

The content of this chapter is joint work with Ramamohan Paturi and appears in [MP10]. With kind permission from Springer Science+Business Media: Theory and Applications of Satisfiability Testing SAT 2010, Uniquely Satisfiable k-SAT Instances with Almost Minimal Occurrences of Each Variable, 6175, 2010, 369–374, W. Matthews and R. Paturi.

Chapter 7

Lower Bounds

One approach to prove $\mathbf{P} \neq \mathbf{NP}$ would be to find a language $L \in \mathbf{NP}$ which requires a family of circuits of size $S(n)$ for each n where $S(n)$ grows faster than any polynomial. This approach seems natural since circuit families may be easier to reason about than Turing Machines. Still, completely arbitrary circuit families may even be too hard to reason about. As a result, researchers have considered restricted classes of circuits in the hopes of proving meaningful results and then gradually relaxing the restrictions.

We will begin with extremely restricted families of circuits and survey the known lower bounds as the restrictions are gradually removed. The simplest standard family of circuits studied is $\mathbf{NC}^0$ which consists of constant depth circuits with bounded fan-in. The result of these circuits may only depend on a constant number of inputs, so it is easy to construct functions that cannot be computed by $\mathbf{NC}^0$ circuits. The next simplest reasonable family of circuits to consider is depth two circuits where the gates have unbounded fan-in (CNFs and DNFs). Any function can be computed by an exponential size CNF or DNF. However, it is straightforward to see that parity and many other simple functions require exponential size CNFs or DNFs. What if we increase the depth, but still keep it constant?

In the early 1980s, Furst, Saxe, and Sipser [FSS84] and independently Ajtai [Ajt83] showed that parity (and other similar functions) cannot be computed by polynomial size $\mathbf{AC}^0$ circuits. Yao [Yao85] improved this by showing that parity requires exponential size $\mathbf{AC}^0$ circuits, and then Håstad [Hås86a] gave essentially

optimal size lower bounds for parity. We will sketch Håstad's proof in Section 7.1 and then in Section 7.2 we give an alternative proof of the same result which follows straightforwardly from our algorithm in Chapter 5.

The natural next step would be to add parity for free. Razborov [Raz87] showed that $\mathbf{ACC}^0(2)$ (constant depth circuits consisting of unbounded fan-in AND, OR, MOD_2, and NOT gates) require exponential size to compute majority. Smolensky [Smo87] showed that for any distinct primes p and q, $\mathbf{ACC}^0(p)$ circuits require exponential size to compute MOD_q.

All of these previous results show that very restricted classes of circuits cannot compute easy functions. On the other hand, there are a few results showing that we can find extremely hard functions where we can prove circuit lower bounds. Buhrman, Fortnow, and Thierauf [BFT98] show that **MA-EXP** $\not\subseteq$ $\mathbf{P}/poly$. **MA-EXP** is the version of **MA** where Arthur runs in randomized exponential time and where the message from Merlin may be exponentially long. Milterson, Vinodchandran, and Watanabe [MVW99] show that there is a language in $\Delta_3^{\mathbf{EXP}}$, the third level of the exponential hierarchy, that requires exponential size circuits.

However, despite all of these results, we still cannot even separate $\mathbf{ACC}^0$ from $\mathbf{NC}^1$, or **P**, or even **PSPACE**, much less separate **P** from **NP**. Indeed, given all of these results, however, it is still plausible that $\mathbf{NEXP} \subseteq \mathbf{ACC}^0(6)$. Recently, Williams [Wil11] made significant progress and showed that at least the last possibility cannot happen. Specifically, there is a language in **NEXP** that does not have polynomial size $\mathbf{ACC}^0$ circuits. In Section 7.3 we sketch Williams' result at a high level, and in Section 7.3.2 we show how a small (but perhaps quite hard to achieve) improvement to our algorithmic result of Chapter 5 combined with Williams' result would imply that $\mathbf{NEXP} \not\subseteq \mathbf{NC}^1$.

7.1 Switching Lemma Based Lower Bounds

In this section we will show how Håstad used his switching lemma to prove essentially optimal lower bound on the size of $\mathbf{AC}^0$ circuits computing parity. In

[Hås86a, Hås86b] he uses similar techniques to also prove similar results for other functions as well.

Theorem 7.1 ([Hås86a])**.** *Any depth d circuit computing the parity of n inputs must have size at least $2^{\Omega(n)^{\frac{1}{d-1}}}$.*

First we will give the intuition behind the proof. Suppose there existed a "small" depth d circuit that computed parity on n inputs, where the bottom level gates all have fan-in at most k (for some reasonably small k). The subcircuits rooted at level $d-1$ can be viewed as a set of k-CNFs (or k-DNFs). If the circuit and k are sufficiently small then we can choose p such that under a random restriction from $\mathcal{R}'^p_n$ (restrictions where each variable is independently set to 0 with probability $(1-p)/2$, to 1 with probability $(1-p)/2$, and to $*$ with probability p) we can simultaneously write all of the subcircuits at level $d-1$ as k-DNFs (or k-CNFs) and such that the number of unset variables is at least pn with positive probability, and in particular such a restriction exists. Then we can combine the gates at levels $d-2$ and $d-1$ since after applying the switching lemma these gates are the same type. Now we have a "small" circuit of depth $d-1$ that computes parity on pn inputs. We repeat this process until we end up with a "small" depth 2 circuit that computes parity. However any depth 2 circuit that computes parity on n inputs must have size at least 2^{n-1}.

We will first prove the theorem in the special case of circuits with bounded bottom fan-in. Then we will use this theorem to prove Theorem 7.1.

Theorem 7.2 ([Hås86a])**.** *For any $d \geq 2$ and sufficiently large n, the parity of n inputs cannot be computed by any (n, m, d, k)-circuit with $m = 2^{\frac{1}{10}n^{\frac{1}{d-1}}}$ and $k = \frac{1}{10}n^{\frac{1}{d-1}}$.*

Proof. We prove this by induction on d. If $d = 2$, then any circuit computing parity must have bottom fan-in n. We prove the inductive case by contradiction. Assume that Theorem 7.2 holds for depth $d-1$, and suppose F is a (n, m, d, k)-circuit circuit computing parity. Let $p = n^{-\frac{1}{d-1}}$ and $\rho \sim \mathcal{R}'^p_n$. We will show that with positive probability either $F|_\rho$ or $\neg F|_\rho$ is a circuit computing parity that doesn't exist.

Without loss of generality, assume that the gates in F at level $d-1$ are $\wedge$ gates. Let $\phi_1, \ldots, \phi_\ell$ denote the subcircuits of F rooted at depth $d-1$. Each of these circuits is a k-CNF and $\ell \leq m$. Let $\psi_i = \phi_i|_\rho$ for $1 \leq i \leq \ell$. The probability that all of the ψ_is cannot be written as k-DNFs is at most $m\alpha^k = (2\alpha)^k$, by Lemma 3.1 with $k' = k$. By the choices of p and k, $2\alpha \leq 1$, so this probability is at most $1/3$ for sufficiently large n. With probability at least $2/3$, each ψ_i can be written as a k-DNF. Replace each ϕ_i in F with ψ_i and collapse the gates at levels $d-2$ and $d-1$ to get a depth $d-1$ circuit. Let F' denote this circuit.

The expected number of unset variables is $pn = n^{\frac{d-2}{d-1}}$. For $n > n_0^d$ for some constant n_0, the probability that the number of unset variables is at least $n^{\frac{d-2}{d-1}}$ is greater than $1/3$. Thus, with positive probability F' is a $\left(n^{\frac{d-2}{d-1}}, 2^{\frac{1}{10}n^{\frac{1}{d-1}}}, d-1, \frac{1}{10}n^{\frac{1}{d-1}}\right)$-circuit, letting $n' = n^{\frac{d-2}{d-1}}$ and $d' = d-1$, we can rewrite this as a $\left(n', 2^{\frac{1}{10}{n'}^{\frac{1}{d'-1}}}, d', \frac{1}{10}{n'}^{\frac{1}{d'-1}}\right)$-circuit. Since applying any restriction to parity gives either parity or the negation of parity, one of F' or $\neg F'$ computes parity, a contradiction. □

The last step to prove Theorem 7.1 is to reduce the bottom fan-in.

Proof of Theorem 7.1. Assume for the sake of contradiction that F is a (n, m, d)-circuit circuit computing parity for $m = 2^{\left(\frac{1}{10}\right)^{\frac{d}{d-1}} n^{\frac{1}{d-1}}}$. Let $p = \frac{1}{10}$ and $\rho \sim \mathcal{R}'^p_n$. View F as a depth $d+1$ circuit with bottom fan-in 1 and apply Lemma 3.1 with $k' = \frac{1}{10}\left(\frac{n}{10}\right)^{\frac{1}{d-1}}$ to the subcircuits of F at depth D. With positive probability $F' = F|_\rho$ is a $(\frac{n}{10}, m, d, k')$-circuit circuit computing either parity or the negation of parity, contradiction Theorem 7.2. □

Håstad also uses his switching lemma to prove lower bounds majority and other functions using similar proofs. Theorem 7.1 is optimal up to constants in the Ω, since parity on n inputs can be computed by depth d circuits of size $2^{O(n)^{\frac{1}{d-1}}}$ (Corollary 2.3).

7.2 $\mathbf{AC}^0$ Algorithm Based Lower Bounds

Let C be a depth d size m $\mathbf{AC}^0$ circuit. Theorem 1.2 implies that there exists a set of at most $O\left(2^{n-n/O(\lg m+d\lg d)^{d-1}}\right)$ restrictions which partition $\{0,1\}^n$ and make C constant. Any function f which requires $\omega\left(2^{n-n/O(\lg m+d\lg d)^{d-1}}\right)$ restrictions to partition $\{0,1\}^n$ and make f constant cannot have depth d size m $\mathbf{AC}^0$ circuits. This line of reasoning allows us to easily reprove many classic lower bounds which were originally proved using switching lemmas (so this isn't a big surprise).

7.2.1 New Proofs of Håstad's Lower Bounds

The results in this section were originally due to Håstad [Hås86a]. We are just giving alternate proofs.

Theorem 7.3. *Any size poly(n) $\mathbf{AC}^0$ circuit which computes parity requires depth at least $\frac{\lg n}{\lg\lg n} - O\left(\frac{\lg n}{\lg^2\lg n}\right)$ and any depth d circuit which computes parity requires size at least $2^{\Omega\left(n^{\frac{1}{d-1}}\right)}$.*

Proof. Parity requires 2^n restrictions to partition $\{0,1\}^n$ and make the parity constant. The two statements in the theorem follow from solving

$$O\left(2^{n-n/O\left(\lg\frac{m}{n}+d\lg d\right)^{d-1}}\right) \geq 2^n$$

for d in terms of n and m and m in terms of d and n respectively. □

For any constant $p \geq 2$, the MOD_p function on n inputs requires at least $2^{n-p+1} = \Omega\left(2^n\right)$ restrictions to partition $\{0,1\}^n$ and make the function constant constant since any restriction which sets fewer than $n-p$ variables clearly cannot make it constant. By the same argument as for Theorem 7.3, we get

Theorem 7.4. *For constant any $p \geq 2$, any size poly(n) $\mathbf{AC}^0$ circuit which computes MOD_p requires depth at least $\frac{\lg n}{\lg\lg n} - O\left(\frac{\lg n}{\lg^2\lg n}\right)$ and any depth d circuit which computes MOD_p requires size at least $2^{\Omega\left(n^{\frac{1}{d-1}}\right)}$.*

To prove the same sort of lower bound for circuits computing the majority function, we need to be able to lower bound the number of restrictions which can partition $\{0,1\}^n$ and make the majority constant. For simplicity, assume that n is odd (the general case will be essentially the same depending whether $n/2$ 1s should output 0 or 1). Let A_0 denote the set of assignments of Hamming weight exactly $\lfloor \frac{n}{2} \rfloor$ and let A_1 denote the set of assignments with Hamming weight exactly $\lceil \frac{n}{2} \rceil$. Every assignment in A_0 causes the majority function to output 0 and every assignment in A_1 causes the majority function to output 1. Furthermore, any set of restrictions partitioning $\{0,1\}^n$ and making the majority constant cannot contain any restrictions which are consistent with two or more assignments in $A_0 \cup A_1$. Suppose there were a restriction ρ consistent with both $a, b \in A_0$ where $a \neq b$ (the $a, b \in A_1$ case is symmetric). Since a and b both have Hamming weight exactly $\lfloor \frac{n}{2} \rfloor$ and $a \neq b$ there must be an index i such that $a_i = 0$ and $b_i = 1$. Thus, we get $\rho(i) = *$. However let $a'_j = a_j$ for all $j \neq i$ and let $a'_i = 1$. The Hamming weight of a' is now $\lceil \frac{n}{2} \rceil$ so the majority function must output 1 but a' is still consistent with ρ. Thus, any such set of restrictions must have size at least $|A_0| + |A_1| = 2\binom{n}{\lfloor n/2 \rfloor} \geq 2^n/n$. Since this bound isn't quite $\Omega(2^n)$, we get a slightly weaker lower bound for majority than Håstad.

Theorem 7.5. *Any size poly(n) $\mathbf{AC}^0$ circuit which computes majority requires depth at least $\frac{\lg n}{\lg \lg n} - O\left(\frac{\lg n}{\lg^2 \lg n}\right)$ and any depth d circuit which computes majority requires size at least $2^{\Omega\left((n/\lg n)^{\frac{1}{d-1}}\right)}$.*

7.2.2 Correlation of $\mathbf{AC}^0$ with Parity

Definition 7.6. Let $\mathcal{C}$ denote a class of circuits. Let $f : \{0,1\}^n \to \{0,1\}$ be a function. The *correlation* of f with circuits from $\mathcal{C}$ is defined as

$$\max_{C \in \mathcal{C}} \left(\Pr_{x \in \{0,1\}^n} [C(x) = f(x)] - \Pr_{x \in \{0,1\}^n} [C(x) \neq f(x)] \right)$$

Not only can we prove that parity cannot be computed exactly by small circuits, we prove that it cannot even be well approximated. Ajtai [Ajt83] showed that for any $\epsilon > 0$, the correlation of parity with any $\mathbf{AC}^0$ circuit is at most $2^{n^{1-\epsilon}}$.

Beame, Impagliazzo, and Srinivasan [BIS11] improve this bound to $2^{-n^{1-O\left(\log^{-1/5} n\right)}}$. Independently of our work, Håstad [Hås11] gives a bound qualitatively equivalent to Theorem 1.6.

Theorem (Theorem 1.6, restated)**.** *The correlation of parity with any* $\mathbf{AC}^0$ *circuit of size* cn *and depth* d *is at most* $2^{-\mu_{c,d}n}$.

Proof of Theorem 1.6. Let $\mathcal{C}$ be the class of depth d size cn $\mathbf{AC}^0$ circuits. We will bound the correlation of $\mathcal{C}$ and the parity function. Let C be an element of $\mathcal{C}$ and consider the partition produce by Theorem 1.2. Each restrictions in this partition which sets fewer than n variables contributes 0 to the correlation with parity. Each restriction which sets all n variables contributes at most 2^{-n} to the correlation with parity. Thus the correlation of $\mathcal{C}$ with parity is at most $2^{n(1-\mu_{c,d})} \cdot n^{-n} = 2^{-\mu_{c,d}n} = 2^{-n/O(\lg c + d \lg d)^{d-1}}$. □

For any n, m, and d, consider the block parity circuit (Section 2.1.2). Each input which sets the circuit to 1 correctly computes parity (assume that the number of groups of inputs is odd). At most half the remaining inputs compute parity correctly. The fraction of inputs which cause the circuit to output one, and therefore the correlation with parity is $2^{-\frac{n}{\Theta(\lg c)^{d-1}}}$. The output of the circuit can always be changed by changing at most one bit per group, or at most $\frac{n}{\Theta\left(\lg \frac{m}{n}\right)^{d-1}}$ bits total. Thus, the number of regions required is at least $2^{n-\frac{n}{\Theta(\lg \frac{m}{n})^{d-1}}}$

7.3 Williams' Approach

Ryan Williams [Wil10, Wil11] gives a powerful connection between circuit satisfiability algorithms and circuit lower bounds. For any constant $c > 0$, there exists a $c' > 0$ such that for any "reasonable" class of circuits $\mathcal{C}$, if there exists a satisfiability algorithm for circuits in $\mathcal{C}$ with $n + c \lg n$ inputs in time $2^n/n^{c'}$ then **NEXP** does not have non-uniform $\mathcal{C}$ circuits. He then gives a satisfiability algorithm for $\mathbf{ACC}^0$ with sufficient savings to prove that $\mathbf{NEXP} \not\subseteq \mathbf{ACC}^0$.

We will give an overview of Williams' approach and then show how a small improvement in our algorithm, from savings $\approx \frac{1}{O(\lg m)^{d-1}}$ to savings $\frac{1}{O(\lg m)^{o(d)}}$, would

imply $\mathbf{NEXP} \not\subseteq \mathbf{NC}^1$ (however, for several reasons, such an improvement may not be easy).

7.3.1 Overview

Let $\mathcal{C}$ be a class of circuits such that $\mathbf{AC}^0 \subseteq \mathcal{C} \subseteq \mathbf{P}/poly$ and such that $\mathcal{C}$ is closed under composition (think of $\mathcal{C}$ as $\mathbf{ACC}^0, \mathbf{TC}^0, \mathbf{NC}^1$, etc.).

Theorem 7.7 (Williams [Wil11])**.** *There exists a constant $c > 0$ such that if there exists an algorithm A to decide the satisfiability of circuits in $\mathcal{C}$ with n inputs in time $2^n/n^c$ then $\mathbf{NEXP} \not\subseteq \mathcal{C}$.*

At a high level, we assume for the sake of contradiction that $\mathbf{NEXP} \subseteq \mathcal{C}$. Then we take a complete language L for $\mathbf{NTIME}[2^n/n^{10}]$ and construct, using the assumption $L \in \mathcal{C}$ and the algorithm A, an algorithm for L in $\mathbf{NTIME}[2^n/n^{20}]$. However, this contradicts the Nondeterministic Time Hierarchy Theorem [Coo73, SFM78, Zák83] so our assumption must be false.

The language SUCCINCT-3-SAT consists of boolean circuits C in n inputs which view the inputs at integers $0 \leq i < 2^n$ and on input i outputs the i^{th} clause of a 3-SAT formula F_C. A circuit C is in SUCCINCT-3-SAT if and only if the corresponding 3-SAT formula F_C is satisfiable. We use the fact every language in $\mathbf{NTIME}[2^n/n^{10}]$ can be reduced to SUCCINCT-3-SAT instances C of size $poly(n)$ with n inputs. The reduction follows from an efficient version of the Cook–Levin Theorem [Coo71, Lev73, Tou01, FLvMV05].

Say that SUCCINCT-3-SAT has *succinct witnesses* if for every $C \in$ SUCCINCT-3-SAT in n inputs there exists a circuit D in n inputs such that for $0 \leq i < 2^n$, $D(i)$ (viewing i as an n bit number in binary) outputs the i^{th} bit of a satisfying assignment to F_C. The assumption $\mathbf{NEXP} \subseteq \mathcal{C}$ implies $\mathbf{NEXP} \subseteq \mathbf{P}/poly$ since we assumed $\mathcal{C} \subseteq \mathbf{P}/poly$. Impagliazzo, Kabanets, and Wigderson [IKW02] show that $\mathbf{NEXP} \subseteq \mathbf{P}/poly$ implies that SUCCINCT-3-SAT has succinct witnesses (and more generally, that every language in **NEXP** has succinct witnesses).

The assumption $\mathbf{NEXP} \subseteq \mathcal{C}$ also implies $\mathbf{P} \subseteq \mathcal{C}$, and in particular circuit evaluation is in $\mathcal{C}$. Thus, for any boolean circuit C (such as the SUCCINCT-3-SAT

instance or its witness), there exists a circuit $C' \in \mathcal{C}$ of size at most $poly(|C|)$, by hard-coding C in the $\mathcal{C}$ circuit for circuit evaluation. Note that just because the circuit C' exists, we may not be able to efficiently find it. Using nondeterminism, however, we can guess C' and check it, but even this is a bit tricky.

Nondeterministically guess circuits $C'_1, \ldots, C'_{|C|} \in \mathcal{C}$ where $C'_j(i)$ is supposed to output the value on the j^{th} gate of $C(i)$. Then for gate j, let $j'_1, \ldots, j'_\ell$ denote the gates corresponding to the inputs to j. Construct a circuit E_j using C'_j and $C'_{j'_1}, \ldots, C'_{j'_\ell}$ such that $E_j(i)$ outputs 0 if and only if the output of $C'_j(i)$ is consistent (in terms of the gate j) with the outputs of $C'_{j'_1}(i), \ldots, C'_{j'_\ell}(i)$. Repeat this process for each gate j. Use the algorithm A to verify that each E_j is *not* satisfiable. After verifying that all these circuits are not satisfiable, let $C' = C'_j$ where j corresponds to the output gate of C.

Now we'll put the pieces together. Let L be a language in $\mathbf{NTIME}[2^n/n^{10}]$ and let x an input such that we want to decide whether $x \in L$. Use the reduction to SUCCINCT-3-SAT to map x to a circuit C. Nondeterministically guess and verify a circuit $C' \in \mathcal{C}$ equivalent to C. Nondeterministically guess a circuit $D \in \mathcal{C}$ which we would like to be a succinct witness for C. Construct a circuit G using C' and D such that $G(i) = 0$ if the assignment encoded by D satisfies the i^{th} clause output by C'. If G is *not* satisfiable then D must encode a satisfying assignment to C' and therefore $x \in L$. However, we can test this by running A on G in time $2^n/n^{20}$, which violates the Nondeterministic Time Hierarchy Theorem. Thus, we can conclude that if $\mathcal{C}$ has an improved satisfiability algorithm, then $\mathbf{NEXP} \not\subseteq \mathcal{C}$.

7.3.2 Towards NEXP $\not\subseteq$ NC^1

Theorem 1.2 (Chapter 5) gives a randomized algorithm that decides the satisfiability of size m depth d $\mathbf{AC}^0$ circuits in n inputs running in time $m \cdot poly(n) \cdot 2^{n\left(1-\frac{1}{O(\lg m)^{d-1}}\right)}$, where the time bound holds with probability at least $1-2^{-n}$. Since Williams' Theorem (Theorem 7.7) uses the satisfiability algorithm in a nondeterministic algorithm to verify that circuits are *not* satisfiable, a co-nondeterministic satisfiability algorithm, and in particular a Las Vegas algorithm (a randomized algorithm that always gives the correct result but where the running time is only

given in expectation) suffices. Our $\mathbf{AC}^0$ satisfiability algorithm (Theorem 1.2) is a Las Vegas algorithm and therefore suffices for use with Williams' Theorem.

In this section, we will show that improving this running time to $m \cdot poly(n) \cdot 2^{n\left(1-\frac{1}{O(\lg m)^{o(d)-1}}\right)}$ (improving the d in the exponent of the savings to $o(d)$) would be sufficient to establish $\mathbf{NEXP} \not\subseteq \mathbf{NC}^1$ via Williams' technique! Note, however, that such an improved satisfiability algorithm must use significantly different techniques since there exists circuits with the above parameters which require $2^{n\left(1-\frac{1}{\Omega(\lg m)^{d-1}}\right)}$ restrictions to enumerate the solutions. This follows from Corollary 2.2 which show that $\mathbf{NC}^1$ circuits have equivalent depth d size $2^{n^{O\left(\frac{1}{d-1}\right)}}$ $\mathbf{AC}^0$ circuits. Combining this corollary with such a hypothetical satisfiability algorithm for $\mathbf{AC}^0$ would give a $2^{n-n^{1-o(1)}}$ time satisfiability algorithm for $\mathbf{NC}^1$. Finally, by Williams' Theorem, this would imply $\mathbf{NEXP} \not\subseteq \mathbf{NC}^1$.

7.4 Acknowledgements

Material in Section 7.2 and parts of Section 7.3 is joint work with Russell Impagliazzo and Ramamohan Paturi and appears in [IMP11].

Bibliography

[AHM+06] Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael Saks, *Minimizing dnf formulas and AC^0_d circuits given a truth table*, Annual IEEE Conference on Computational Complexity **0** (2006), 237–251.

[Ajt83] M. Ajtai, Σ^1_1*-formulae on finite structures*, Annals of Pure and Applied Logic **24** (1983), no. 1, 1 – 48.

[AP04] Dimitris Achlioptas and Yuval Peres, *The threshold for random k-SAT is* $2^k \ln 2 - O(k)$, J. AMS **17** (2004), 947–973.

[ART06] Dimitris Achlioptas and Federico Ricci-Tersenghi, *On the solution-space geometry of random constraint satisfaction problems*, STOC (Jon M. Kleinberg, ed.), ACM, 2006, pp. 130–139.

[BCD+89] A. Borodin, S.A. Cook, P.W. Dymond, W.L. Ruzzo, and M. Tompa, *Two applications of inductive counting for complementation problems*, SIAM Journal on Computing **18** (1989), no. 3, 559–578.

[Bea90] Paul Beame, *Lower bounds for recognizing small cliques on crcw pram's*, Discrete Applied Mathematics **29** (1990), no. 1, 3 – 20.

[Bea94] ———, *A switching lemma primer*, Technical Report, Department of Computer Science and Engineering, University of Washington, 1994.

[BFT98] H. Buhrman, L. Fortnow, and L. Thierauf, *Nonrelativizing separations*, Proceedings of the Thirteenth Annual IEEE Conference on Computational Complexity, 1998, pp. 8–12.

[BH89] Paul Beame and Johan Hastad, *Optimal bounds for decision problems on the crcw pram*, J. ACM **36** (1989), 643–670.

[BIK+92] Paul Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, Pavel Pudlák, and Alan Woods, *Exponential lower bounds for the pigeonhole principle*, Proceedings of the twenty-fourth annual ACM

symposium on Theory of computing (New York, NY, USA), STOC '92, ACM, 1992, pp. 200–220.

[BIS11] Paul Beame, Russell Impagliazzo, and Srikanth Srinivasan, *Approximating ac0 circuits by 'small' height decision trees*, manuscript, March 2011.

[Cai89] Jin-Yi Cai, *With probability one, a random oracle separates pspace from the polynomial-time hierarchy*, Journal of Computer and System Sciences **38** (1989), no. 1, 68 – 85.

[CIKP08a] C. Calabro, R. Impagliazzo, V. Kabanets, and R. Paturi, *The complexity of unique k-*SAT*: An isolation lemma for k-*CNF*s*, Journal of Computer and Systems Sciences **74** (2008), no. 3, 386–393, Preliminary version in *Proceedings of the Eighteenth IEEE Conference on Computational Complexity*, 135-144, 2003.

[CIKP08b] Chris Calabro, Russell Impagliazzo, Valentine Kabanets, and Ramamohan Paturi, *The complexity of unique k-SAT: An isolation lemma for k-CNFs*, J. Comput. Syst. Sci. **74** (2008), no. 3, 386–393.

[CIP06] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi, *A duality between clause width and clause density for* SAT, CCC '06: Proceedings of the 21st Annual IEEE Conference on Computational Complexity (Washington, DC, USA), IEEE Computer Society, 2006, pp. 252–260.

[CIP09] ———, *The complexity of satisfiability of small depth circuits*, Parameterized and Exact Computation: 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers (Berlin, Heidelberg), Springer-Verlag, 2009, pp. 75–85.

[Coo71] S.A. Cook, *The complexity of theorem-proving procedures*, Proceedings of the Third Annual ACM Symposium on Theory of Computing, 1971, pp. 151–158.

[Coo73] Stephen A. Cook, *A hierarchy for nondeterministic time complexity*, Journal of Computer and System Sciences **7** (1973), no. 4, 343 – 353.

[DJW05] Vilhelm Dahllöf, Peter Jonsson, and Magnus Wahlström, *Counting models for 2sat and 3sat formulae*, Theoretical Computer Science **332** (2005), no. 1–3, 265–291.

[FK07] Martin Fürer and Shiva Prasad Kasiviswanathan, *Algorithms for counting 2-sat solutions and colorings with applications*, 2007, pp. 47–57.

[FLvMV05] Lance Fortnow, Richard Lipton, Dieter van Melkebeek, and Anastasios Viglas, *Time-space lower bounds for satisfiability*, J. ACM **52** (2005), 835–865.

[FSS84] Merrick Furst, James B. Saxe, and Michael Sipser, *Parity, circuits, and the polynomial-time hierarchy*, Mathematical Systems Theory **17** (1984), no. 1, 13–27.

[Geb09] Heidi Gebauer, *Disproof of the neighborhood conjecture with implications to SAT*, Algorithms - ESA 2009, vol. 5757/2009, Springer Berlin / Heidelberg, 2009, pp. 764–775.

[Hås86a] J. Håstad, *Almost optimal lower bounds for small depth circuits*, Proceedings of the eighteenth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '86, ACM, 1986, pp. 6–20.

[Hås86b] ———, *Computational limitations for small depth circuits*, Ph.D. thesis, MIT, 1986.

[Hås11] Johan Håstad, *On parity*, Unpublished Manuscript, 2011.

[Her11] Timon Hertli, *3-SAT Faster and Simpler — Unique-SAT Bounds for PPSZ Hold in General*, 2011.

[HJP95] Johan Håstad, Stasys Jukna, and Pavel Pudlák, *Top-down lower bounds for depth-three circuits*, Computational Complexity **5** (1995), no. 2, 99–112.

[IKW02] R. Impagliazzo, V. Kabanets, and A. Wigderson, *In search of an easy witness: Exponential time vs. probabilistic polynomial time*, Journal of Computer and System Sciences **65** (2002), no. 4, 672–694, (preliminary version in CCC'01).

[IMP11] Russell Impagliazzo, William Matthews, and Ramamohan Paturi, *A satisfiability algorithm for AC^0*, Unpublished Manuscript, 2011.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane, *Which problems have strongly exponential complexity?*, J. Comput. Syst. Sci. **63** (2001), no. 4, 512–530.

[KST93] Jan Kratochvíl, Petr Savický, and Zsolt Tuza, *One more occurrence of variables makes satisfiability jump from trivial to NP-complete*, SIAM Journal on Computing **22** (1993), no. 1, 203–210.

[Lev73] L. Levin, *Universal sorting problems*, Problems of Information Transmission **9** (1973), 265–266.

[LP11] Daniel Lokshtanov and Ramamohan Paturi, Personal Communication, 2011.

[LPI01] Michael L. Littman, Toniann Pitassi, and Russell Impagliazzo, *New and old algorithms for belief net inference and counting satisfying assignments*, Unpublished Manuscript, 2001.

[Lyn86] James F. Lynch, *A depth-size tradeoff for boolean circuits with unbounded fan-in*, 1986, pp. 234–248.

[MP10] William Matthews and Ramamohan Paturi, *Uniquely satisfiable k-sat instances with almost minimal occurrences of each variable*, 2010, pp. 369–374.

[MS85] B. Monien and E. Speckenmeyer, *Solving satisfiability in less than 2^n steps*, Discrete Appl. Math. **10** (1985), 287–295.

[MS11] Robin Moser and Domink Scheder, *A Full Derandomization of Schöning's k-SAT Algorithm*, Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, 2011.

[MVW99] P. B. Miltersen, N.V. Vinodchadran, and O. Watanabe, *Super-polynomial versus half-exponential circuit size in the exponential hierarchy*, Proceedings of the Fifth Annual International Conference on Computing and Combinatorics (T. Asano, H. Imai, D.T. Lee, S. Nakano, and T. Tokuyama, eds.), Lecture Notes in Computer Science, vol. 1627, Springer Verlag, 1999, (COCOON'99), pp. 210–220.

[MZ09] Sharad Malik and Lintao Zhang, *Boolean satisfiability from theoretical hardness to practical success*, Commun. ACM **52** (2009), 76–82.

[PPSZ05] R. Paturi, P. Pudlák, M.E. Saks, and F. Zane, *An improved exponential-time algorithm for k-*SAT, Journal of the ACM **52** (2005), no. 3, 337–364, Preliminary version in *39th Annual IEEE Symposium on Foundations of Computer Science*, pages 628–637, 1998.

[PPZ99] R. Paturi, P. Pudlák, and F. Zane, *Satisfiability coding lemma*, Chicago Journal of Theoretical Computer Science (1999), (preliminary version in FOCS'97).

[Raz87] A. A. Razborov, *Lower bounds on the size of bounded depth circuits over a complete basis with logical addition*, Matematicheskie Zametki **41** (1987), no. 4, 598—607.

[Raz93] Alexander A. Razborov, *Bounded arithmetic and lower bounds in boolean complexity*, Feasible Mathematics II, Birkhauser, 1993, pp. 344–386.

[San10] Rahul Santhanam, *Fighting perebor: New and improved algorithms for formula and qbf satisfiability*, Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '10, IEEE Computer Society, 2010, pp. 183–192.

[SBI02] N. Segerlind, S. Buss, and R. Impagliazzo, *Switching lemma for small restrictions and lower bounds for k-dnf resolution*, Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on, 2002, pp. 604 – 613.

[Sch99] U. Schöning, *A probabilistic algorithm for k-*SAT *and constraint satisfaction problems*, FOCS, 1999, pp. 410–414.

[Sch05] R. Schuler, *An algorithm for the satisfiability problem of formulas in conjunctive normal form*, Journal of Algorithms **54** (2005), no. 1, 40–44.

[SFM78] J.I. Seiferas, M.J. Fischer, and A.R. Meyer, *Separating nondeterministic time complexity classes*, Journal of the Association for Computing Machinery **25** (1978), no. 1, 146–167.

[Smo87] R. Smolensky, *Algebraic methods in the theory of lower bounds for boolean circuit complexity*, Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (New York City, NY) (Alfred Aho, ed.), ACM Press, May 1987, pp. 77–82.

[TBW04] Christian Thiffault, Fahiem Bacchus, and Toby Walsh, *Solving non-clausal formulas with dpll search*, 2004, pp. 663–678.

[Tou01] Iannis Tourlakis, *Time-space tradeoffs for sat on nonuniform machines*, Journal of Computer and System Sciences **63** (2001), no. 2, 268 – 287.

[Tov84] Craig A. Tovey, *A simplified NP-complete satisfiability problem*, Discrete Applied Mathematics **8** (1984), no. 1, 85 – 89.

[Val79a] L. Valiant, *The complexity of computing the permanent*, Theoretical Computer Science **8** (1979), 189–201.

[Val79b] ———, *The complexity of enumeration and reliability problems*, SIAM J. on Computing **8** (1979), no. 3, 410–421.

[Ven87] H. Venkateswaran, *Properties that characterize logcfl*, Proceedings of the nineteenth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '87, ACM, 1987, pp. 141–150.

[VV86] L. G. Valiant and V. V. Vazirani, *NP is as easy as detecting unique solutions*, Theor. Comput. Sci. **47** (1986), no. 1, 85–93.

[Wil10] Ryan Williams, *Improving exhaustive search implies superpolynomial lower bounds*, Proceedings of the 42nd ACM symposium on Theory of computing (New York, NY, USA), STOC '10, ACM, 2010, pp. 231–240.

[Wil11] ———, *Non-Uniform ACC Circuit Lower Bounds*, Proceedings of the Twenty-Sixth Annual IEEE Conference on Computational Complexity, 2011.

[WLLH07] Chi-An Wu, Ting-Hao Lin, Chih-Chun Lee, and Chung-Yang (Ric) Huang, *Qutesat: a robust circuit-based sat solver for complex circuit structure*, Proceedings of the conference on Design, automation and test in Europe (San Jose, CA, USA), DATE '07, EDA Consortium, 2007, pp. 1313–1318.

[Yao85] Andrew C-C. Yao, *Separating the polynomial-time hierarchy by oracles*, Proc. 26th annual symposium on Foundations of computer science (Piscataway, NJ, USA), IEEE Press, 1985, pp. 1–10.

[Yat37] F. Yates, *The design and analysis of factorial experiments*, Imperial Bureau of Soil Science Technical Communication (1937).

[Zák83] S. Zák, *A Turing machine time hierarchy*, Theoretical Computer Science **26** (1983), 327–333.