SOME APPLICATIONS OF RANDOMNESS IN COMPUTATIONAL COMPLEXITY

BY LUKE FRIEDMAN

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Eric Allender

and approved by

New Brunswick, New Jersey

May, 2013

ABSTRACT OF THE DISSERTATION

Some Applications of Randomness in Computational Complexity

by Luke Friedman

Dissertation Director: Eric Allender

In this dissertation we consider two different notions of randomness and their applications to problems in complexity theory.

In part one of the dissertation we consider Kolmogorov complexity, a way to formalize a measure of the randomness of a single finite string, something that cannot be done using the usual distributional definitions. We let R be the set of random strings under this measure and study what resource-bounded machines can compute using R as an oracle. We show the surprising result that under proper definitions we can in fact define well-formed complexity classes using this approach, and that perhaps it is possible to exactly characterize standard classes such as BPP and NEXP in this way.

In part two of the dissertation we switch gears and consider the use of randomness as a tool in propositional proof complexity, a sub-area of complexity theory that addresses the NP vs. coNP problem. Here we consider the ability of various proof systems to efficiently refute randomly generated unsatisfiable 3-CNF and 3-XOR formulas. In particular, we show that certain restricted proof systems based on Ordered Binary Decision Diagrams requires exponential-size refutations of these formulas. We also outline a new general approach for proving proof complexity lower bounds using random 3-CNF formulas and demonstrate its use on treelike resolution, a weak proof system.

Acknowledgements

First of all I would like to thank my advisor Eric Allender for guiding me through my time as a Ph.D student. He has taught me a whole lot about computer science and how to do research, and he has always been willing to meet with me and help me in any way that he could. I consider it a privilege to have been able to work with him.

I would also like to thank the rest of the math and computer science faculty at Rutgers whom I have interacted with over the years. Particularly I would like to thank Mike Saks for running our weekly reading seminar and for meeting with me on numerous occasions, and Mario Szegedy, Bill Steiger, Martin Farach-Colton, Michael Fredman, Jeff Kahn, and Troy Lee for teaching courses that I enjoyed.

Thanks to Eric Allender, Mike Saks, Swastik Kopparty and Toni Pitassi for taking the time to be on my thesis committee.

I certainly need to acknowledge the co-authors of work that appears in this dissertation. They are Eric Allender, Harry Buhrman, George Davie, Bill Gasarch, Samuel Hopkins, Bruno Loff, Iddo Tzameret, and Yixin Xu.

Carol DiFrancesco is the graduate secretary in the computer science department and could not have been more helpful or kind – countless times I would come to her office with some bureaucratic mess I had gotten myself into, and she would always patiently help me extract myself. Thanks also to Regina Ribaudo, Maryann Holtsclaw, and Aneta Unrat for helping me with payroll, travel, insurance, and other issues.

Lastly, I would like to thank all my friends in the area within and without the department for the times we have spent together. There are inevitable ups and downs over the course of a Ph.D and I could not have gotten through it without you guys.

Research conducted in this thesis was supported in part by NSF grants CCF-0832787, CCF-1064785, CCF-0830133, and also in part by [European Community's]

Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 238381.

Dedication

To my parents, for their unwavering support these last 30 years.

Table of Contents

Chapter 1

Introduction

The use of randomness is an important theme that appears throughout theoretical computer science. Whether there exist languages that are decidable in polynomial-time by a randomized algorithm but not by a deterministic algorithm is formalized as the BPP vs. P problem and is one of the central open problems in computational complexity. One of the notable successes of complexity theory over the last few decades has been a long line of research (including [Yao82, NW94, IW01]) giving strong evidence that these two complexity classes in fact coincide – and therefore that at this level of generality the use of randomness does *not* offer any advantage. However, at finer levels of gradation randomness *is* provably beneficial; for instance, it has been shown that any deterministic comparison-based algorithm for finding the median of n elements will make at least $2n$ comparisons in the worst case [BJ85], whereas there is a randomized algorithm that always succeeds and makes at most $1.5n$ comparisons both in expectation and with high probability [FR75]. And in other contexts randomness is completely integral; important applications such as cryptography, interactive proofs, and property testing, to name just a few, could not even exist without the use of randomness.

Randomness is not only useful as an applied tool, but also has emerged as an important tool in the *analysis* of algorithms and protocols, and in general as a proof technique in theoretical computer science. There are many examples where randomness can be used to prove theorems for which more constructive techniques do not seem to suffice, or at least provides a way of significantly simplifying other known proofs. Often in these cases the computational objects themselves being considered in the proof on the surface seem to be wholly deterministic. The most basic example of this phenomenon is the probabilistic method, championed by Paul Erdős and used throughout mathematics, by which an object with a given property is shown to exist by randomly choosing the object from some distribution and then proving that with positive probability the object will have the given property. One classic example of a sophisticated variant of this approach in computational complexity is the use of random restrictions to prove circuit lower bounds [FSS84, Has86]; these circuit classes have no explicit randomness in their definitions, but randomness can be used in the analysis to understand their strength. In this dissertation we give two applications of this approach of injecting randomness in non-obvious ways into the analysis of problems in order to prove theorems in computational complexity. The dissertation is split into two largely independent parts, using two very different notions of randomness and ways in which it is employed.

In Chapters 2 and 3 we consider the set R of Kolmogorov random strings: those strings for which there is no short description according to some fixed universal reference machine. (In fact, we consider two related notions of R: R_C, the random strings according to the "plain" Kolmogorov complexity, and R_K, the random strings according to the "prefix" complexity). Specifically, we study what extra power is afforded to a resource-bounded machine if it is given the set R as an oracle. Previous work had shown that in this setting R *can* provide additional power: For instance, it had been shown that any language computable in nondeterministic exponential time (NEXP) can be computed by a nondeterministic polynomial-time (NP) machine with oracle access to R [ABK06a], and any language computable in randomized polynomial time (BPP) can be computed by a polynomial-time machine with truth-table (i.e. non-adaptive) oracle access to R [BFKL10].

Although these results were intriguing, it was not clear exactly how to interpret them – how could one hope to understand the *limits* to the power that the oracle R provides to resource-bounded machines when the set R itself is an uncomputable set? The main contribution of this first part of the dissertation is to show that in fact, if the problem is formulated correctly, one *can* provide meaningful upper bounds to the power of R as an oracle. In Chapter 2 we show the first result in this direction, from [AGF13]. Roughly speaking, we show that for any decidable language L, if for every universal reference machine U there exists a polynomial-time machine that can compute L when given nonadaptive oracle access to R_K defined with respect to the machine U, then L is contained in PSPACE. (We are also able to lift this result to get an analogous exponential-space upper bound (EXPSPACE) that holds for nondeterministic polynomial-time machines).

In Chapter 3 we tackle the problem of whether we can improve this PSPACE upper bound to more closely match the known BPP lower bound. Specifically, we explore the possibility of improving the PSPACE upper bound to a PSPACE $\cap$ P/poly upper bound using an approach described in [ADF$^+$12]. This attempt yields both positive and negative results. On the one hand, we show that the approach cannot work for the set R as originally intended, and that new ideas will be needed to get a P/poly upper bound in this case. However, we are also able to show that if we consider a time-bounded variant of Kolmogorov complexity in the definition of the random strings, then in fact the intuition behind the approach *does* yield the desired P/poly upper bound, and that the PSPACE upper bound and BPP lower bound still hold in this setting as well.

Although there is still a significant gap between the known lower bounds and upper bounds, this work suggests the tantalizing possibility of exactly characterizing standard complexity classes such as BPP or NEXP in this unusual way, as the set of languages computable by a resource-bounded machine with oracle access to the Kolmogorov random strings. (Note that while randomness is a main feature of the class BPP, there is no explicit reference to randomness in a class such as NEXP). Such an exact characterization might allow us to harness tools from computability theory that up to this point did not seem applicable to problems in complexity theory. Although such ideas are extremely speculative at this point, it is at least conceivable that this could lead to new class separations or a new way to attack the BPP vs. P problem.

In the second part of the dissertation, we shift our focus to an entirely different type of randomness, and an entirely different sub-area of computational complexity called propositional proof complexity. In propositional proof complexity, the main task is to show that a given family of tautological propositional formulas T does not have small proofs in a given proof system P, for various T and P. Equivalently, as we do in this dissertation, we can consider families of unsatisfiable formulas and argue about the size of the smallest refutations of these families in a given proof system. One of the main motivations for studying propositional proof complexity is that it can be viewed as a way of making gradual progress towards separating nondeterministic polynomial-time (NP) from co-nondeterministic polynomial time (coNP), which would immediately imply a separation of P from NP. Another motivation is that results about specific proof systems also have consequences for related classes of SAT solving algorithms. There are also many connections between propositional proof complexity and circuit complexity and there is a rich interplay between the two fields.

The main families of unsatisfiable formulas we focus on in this dissertation are randomly generated 3-CNF formulas, in which each clause of the formula is chosen uniformly at random from all possible clauses over n variables. For large enough clause density, with high probability such formulas will be unsatisfiable. The intuition is that random formulas lack structure and will therefore be hard to refute in any proof system. Again, thematically, the proof systems themselves are entirely deterministic, but our hope is to use randomness to reason about their power.

In Chapter 4 we consider a specific proof system based on Ordered Binary Decision Diagrams that was introduced by Atserias et al. in [AKV04]. Krajíček proved lower bounds for a strong variant of this system using a method called feasible interpolation [Kra07], and Tveretina et al. proved lower bounds on the size of refutations of a natural combinatorial family of formulas called the pigeonhole formulas in a restricted version of this proof system [TSZ10]. We prove the first lower bounds for restricted versions of this proof system with respect to random formulas. Specifically, we show that with high probability a randomly chosen 3-CNF formula with clause density above the satisfiability threshold requires exponential-size refutations in a restricted version of this proof system. (We also show that with high probability a randomly chosen 3-XOR formula with clause density above the satisfiability threshold requires exponential-size refutations in another restricted version of this proof system).

In Chapter 5 we propose a general framework for proving lower bounds for proof systems using random 3-CNF formulas. Unlike most of the known techniques, this framework is specifically tailored to work with random formulas and was not adapted from techniques previously shown to work with constructive families of formulas. After introducing the framework we demonstrate its use by proving lower bounds for a very weak proof system called treelike resolution. This is essentially a toy example, as lower bounds for random formulas in this system (and for strictly stronger systems) have already been proved using different techniques. However, the hope is that the framework could eventually be cultivated to prove lower bounds on random formulas for stronger proof systems for which such lower bounds are unknown.

It is assumed that the reader is familiar with basic concepts and notation from computational complexity – if not, a good primer is [AB09].

Chapter 2 is largely based on material from [AGF13], Chapter 3 is based on material from [ADF$^+$12] and [ABFL12a], Chapter 4 is based on material from [FX13], and Chapter 5 is based on material from [Fri13]. It should be pointed out that two important theorems from Chapter 3 (Theorems 3.3.2 and 3.3.4), although published in [ABFL12a] and inspired at least partially by [ADF$^+$12], were originally proved by Harry Buhrman and Bruno Loff without any contribution from the author of this dissertation.

Chapter 2
Limits to the Power of Random Strings

2.1 Background on Kolmogorov Complexity

Kolmogorov complexity addresses the following paradox.. Consider an experiment where a fair coin is flipped a million times, and the result is recorded as a binary string of 1s and 0s, where a 1 represents heads and a 0 represents tails. Suppose one is told that the experiment has been conducted twice behind closed doors, resulting in the following strings:

A: 10111001101010100010000001...

B: 00000000000000000000000000...

In the first case, one would not have much reason to question the integrity of the experiment (in fact it is the beginning of a string generated by a real computer-simulated run of this experiment for the purpose of this example). However, even the most trusting observer would probably refuse to believe that the second sequence was a legitimate result of such an experiment.

The paradox arises from the fact that a priori each of the two strings has exactly the same probability of appearing (which we can easily calculate to be $2^{-1,000,000}$). Why then does it seem so much more improbable that string B would appear compared with string A? A good answer is that we doubt string B because it has a short description, and intuitively a random string should not have this property. An equivalent explanation would be that intuitively a random string should not be "compressible". Indeed, string B can be described in English words as "one million zeros", a much more compact representation than literally writing out the one million zeros, whereas it is not clear how to describe string A more succinctly than to literally write out the entire string.

The goal of Kolmogorov complexity is to formalize this intuition about what makes an individual finite string random in a way that our normal distributional notions of randomness cannot, by defining exactly what it means for something to be a valid "description" of a string. To define Kolmogorov complexity we invoke the normal Turing machine model in which a machine M takes as input a binary string and, if it halts, outputs another binary string.

Definition 2.1.1 (Kolmogorov Complexity). *The Kolmogorov complexity of a binary string x with respect to a Turing machine M, denoted by $C_M(x)$, is the length of the shortest string y such that the machine M outputs x on input y. Formally, $C_M(x) = \min |y| : M(y) = x$.*

We will define *universal* Turing machines in terms of Kolmogorov complexity as follows:

Definition 2.1.2 (Universal Turing Machine). *A Turing machine U is a universal Turing machine if for any Turing machine M there exists a constant c_M, depending only on M, such that for all x, $C_U(x) \leq C_M(x) + c_M$.*

Note that the standard two-part universal Turing machine U that takes as input $\langle M, x\rangle$, a description of a Turing machine M and a string x, simulates M on input x, and outputs $M(x)$, is one example of a universal Turing machine under this more general definition. To see this, note that if $M(y) = x$, then $U(\langle M, y\rangle) = x$, so we need only to define c_M to be large enough to account for an encoding of the machine M.

The advantage of using universal machines as the reference machines in the definition of Kolmogorov complexity is that the measure then becomes invariant up to an additive constant. Indeed, for any two universal Turing machines U and U', we have that for all x, $|C_U(x) - C_{U'}(x)| \leq c_{U,U'}$ for some constant $c_{U,U'}$ depending only on the machines U and U'. This is an important property, as we would like the "complexity" of a string to be an absolute property of the string itself and not relative to the underlying reference machine in the definition of Kolmogorov complexity.

In many applications, due to this invariance property, the choice of which universal machine to use as the reference machine is unimportant (although in the application we will explore later in this chapter, the choice of reference machine will often play a crucial role). When this is the case, we will sometimes drop the subscript and denote the Kolmogorov complexity of a string x as $C(x)$, which is shorthand for $C_{U_0}(x)$, where U_0 is some canonical universal machine.

We are now ready to define the set of Kolmogorov random strings – those strings who do not have a description shorter than their own length.

Definition 2.1.3. *The set of Kolmogorov random strings with respect to a universal machine U, denoted by R_{C_U}, is the set of all strings x such that the Kolmogorov complexity of x with respect to U is at least the length of x. Formally, $R_{C_U} = \{x : C_U(x) \geq |x|\}$.*

Again, in situations where the particular universal machine used as the reference machine is immaterial, we will drop the subscript and refer simply to R_C.

We now mention a couple crucial properties of the set of Kolmogorov random strings. For an exhaustive treatment of Kolmogorov complexity, including its applications in various area of mathematics and computer science, we refer the reader to the standard textbook [LV08].

- R_C is uncomputable. That is, there does not exist a Turing machine M that halts on every input x and accepts x if and only if $x \in R_C$. Note however that R_C *is* co-computably enumerable. That is, there *does* exist a Turing machine T that given any input x, halts and accepts x if $x \notin R_C$, and either does not halt or rejects x if $x \in R_C$. T works by simulating in parallel, via a dovetailing procedure, $U_0(y)$ for every string y such that $y < |x|$, and accepting x if U_0 outputs x on any of these y's (Remember, U_0 is the implicit universal reference machine in the definition of R_C). If there does exists some y such that $U_0(y) = x$, so that $x \notin R_C$, then eventually T will discover this and accept x.

- At least a constant fraction of all strings of length n are Kolmogorov random. Formally, there exists a constant c such that for all n, $|R_C| \geq c2^n$.

2.1.1 Prefix Complexity

For the results in this chapter, we will actually need to focus on a popular variant of Kolmogorov complexity called the *prefix complexity.* We will henceforth refer to the version of Kolmogorov complexity described in the previous section as the "plain" complexity. To define the prefix complexity, we first must define what it means for a Turing machine to be a prefix machine.

Definition 2.1.4 (Prefix Machine)**.** *Let λ denote the empty string. A prefix machine is a Turing machine M such that, for all x, if $M(x)$ halts then, for all $y \neq \lambda$, $M(xy)$ does not halt. That is, the domain of M is a prefix code.*

We can mimic the definitions of the plain complexity in order to define the prefix complexity, in each case adding the extra condition that all reference machines must be prefix machines. Thus:

Definition 2.1.5 (Prefix Complexity)**.** *We define the prefix complexity of a string x with respect to a prefix machine M to be $K_M(x) = \min |y| : M(y) = x$.*

Definition 2.1.6 (Universal Prefix Machine)**.** *A prefix machine U is a universal prefix machine if for any prefix machine M there exists a constant c_M, depending only on M, such that for all x, $K_U(x) \leq K_M(x) + c_M$.*

Definition 2.1.7. *We define the set of Kolmogorov prefix random strings with respect to a universal prefix machine U, as $R_{K_U} = \{x : K_U(x) \geq |x|\}$.*

Again, we will use the shorthands $K(x)$ and R_K if the choice of universal reference machine plays a negligible role. Also, if it does not matter which version of the random strings we are using, we will sometimes refer to the set of random strings simply as R.

It is known that, for some constant c, $C(x) - c \leq K(x) \leq C(x) + c + 2\log|x|$, and hence the two versions of Kolmogorov complexity are not very far apart from each other. Although arguably the definition of the prefix complexity is slightly less natural than the plain complexity, the prefix complexity retains most of the important properties of the plain complexity, along with some additional properties that often make it more convenient to work with. For instance, the plain complexity is not sub-additive (i.e. $C(x, y) \leq C(x) + C(y) + c$ does not hold in general for any constant c), and the series $\sum_x 2^{-C(x)}$ diverges, which means it cannot easily be converted into a probability measure. The prefix complexity fixes both of these problems, and is crucial to many of the applications that originally motivated the discovery of Kolmogorov complexity, such as studying the complexity of infinite sequences and defining a universal prior probability measure that could be used as the basis for inductive reasoning. For an in-depth discussion of the trade-offs between the plain and prefix complexity, see [LV08, Chapter 3].

2.1.2 The Random Strings as an Oracle

Our main focus in this part of the dissertation is to study what happens when the set of Kolmogorov random strings is used as an oracle. In order to do this, we need to recall the definitions for a number of different types of reductions that will be used throughout this chapter and the next.

- *Turing reductions.* We say that a language A *$\mathcal{R}$-Turing reduces* to a language B ($A \leq_T^{\mathcal{R}} B$) if there is an oracle Turing machine in class $\mathcal{R}$ that accepts A when given B as an oracle. (An oracle Turing machine with access to oracle B is a Turing machine equipped with an extra oracle tape. During its computation, the machine is permitted to write a string x to the oracle tape and enter a special oracle state, after which it receives the answer to the query "Is x in the language B" at the cost of a single time-step). If we write $\mathcal{C} \leq_T^{\mathcal{R}} B$ for a complexity class $\mathcal{C}$, it means that for all languages $A \in \mathcal{C}$, $A \leq_T^{\mathcal{R}} B$.

- *Truth-table (nonadaptive) reductions.* A truth-table reduction is a Turing reduction with the additional constraint that for a given input the machine computing the reduction must compute all the queries that it will ask before receiving any answers to the query. An equivalent definition that we will make use of in this chapter is the following. For a complexity class $\mathcal{R}$ and languages A and B, we say that A *$\mathcal{R}$-truth-table-reduces* to B ($A \leq_{tt}^{\mathcal{R}} B$) if there is a function q computable in $\mathcal{R}$, such that, on an input $x \in \{0,1\}^*$, q produces an encoding of a circuit λ and a list of queries $q_1, q_2, \ldots q_m$ so that for $a_1, a_2, \ldots, a_m \in \{0,1\}$ where $a_i = 1$ if and only if $q_i \in B$, it holds that $x \in A$ if and only if $\lambda(a_1 a_2 \cdots a_m) = 1$. Specifically, if the function q is polynomial-time computable, we say that A *polynomial-time-truth-table-reduces* to B ($A \leq^p_{tt} B$).

- *Monotone, anti-monotone, and disjunctive truth-table reductions.* In the scenario above, if the circuit λ computes a monotone function (i.e. changing any input bit of the function from 0 to 1 cannot change the output of the function from 1 to 0), then we say that *A $\mathcal{R}$-monotone-truth-table-reduces* to B ($A \leq^{\mathcal{R}}_{mtt} B$). If λ computes an anti-monotone function (i.e. $\neg\lambda$ is monotone), then we say that *A $\mathcal{R}$-anti-monotone truth-table-reduces* to B ($A \leq^{\mathcal{R}}_{amtt} B$). If λ computes an OR function (i.e. $\lambda(a_1 a_2 \cdots a_m) = 1$ if and only if there exists some $a_i \in B$), then we say that *A R-disjunctive-truth-table-reduces* to B ($A \leq^{\mathcal{R}}_{dtt} B$).

In any of these reductions, if no complexity class $\mathcal{R}$ is specified then it is implied that there are no complexity constraints on the machine performing the reduction.

The first main result in this line of research was by Martin in 1966, who showed that any computably enumerable set is Turing reducible to both R_C and R_K [Mar66]. Kummer later improved this result by showing that any recursively enumerable set is in fact disjunctive truth-table reducible to R_C [Kum96]. Muchnik and Positselsky then proved the negative result that for some universal prefix machine U, there is no truth-table reduction from the halting problem to the overgraph of K_U, a function related to K_U that we will define later in the chapter [MP02]. This demonstrated a significant difference between R_K and R_C, as Kummer's result implied that the halting problem *is* truth-table reducible to the overgraph of C_U, regardless of which universal machine U is used.

Relatively recently, researchers began to focus on what happens when the machine with oracle access to the Kolmogorov random strings is resource-bounded. The surprising conclusion was that despite the fact that R is uncomputable, resource-bounded machines *could* extract extra power from these sets. Three curious results in this direction are the following.

Theorem 2.1.8 ([ABK$^+$06b]). PSPACE $\subseteq \mathrm{P}^R$

Theorem 2.1.9 ([ABK06a]). NEXP $\subseteq \mathrm{NP}^R$

Theorem 2.1.10 ([BFKL10]). BPP $\subseteq \{A : A \leq^{\mathrm{p}}_{tt} R\}$

All of these results were derived by means of derandomization techniques. We call these inclusions "curious" because the upper bounds that they provide for the complexity of problems in BPP, PSPACE and NEXP are not even computable; thus at first glance these inclusions may seem either trivial or nonsensical.

A key step toward understanding these inclusions in terms of standard complexity classes is to invoke one of the guiding principles in the study of Kolmogorov complexity: The choice of universal machine should be irrelevant. Theorems 2.1.8 through 2.1.10 actually show that problems in certain complexity classes are *always* reducible to R, no matter *which* universal machine is used as the reference machine. The inclusions also hold regardless of whether we consider R_C or R_K. Combining these insights with the fact that $\mathrm{BPP}, \mathrm{PSPACE}$, and NEXP are all contained in Δ^0_1 (the class of decidable languages), we have

- BPP $\subseteq \Delta^0_1 \cap \bigcap_U \{A : A \leq^{\mathrm{p}}_{tt} R_{K_U}\}$.
- PSPACE $\subseteq \Delta^0_1 \cap \bigcap_U \mathrm{P}^{R_{K_U}}$.
- NEXP $\subseteq \Delta^0_1 \cap \bigcap_U \mathrm{NP}^{R_{K_U}}$.

The question arises as to how powerful the set $\Delta^0_1 \cap \bigcap_U \{A : A \leq_r R_{K_U}\}$ is, for various notions of reducibility $\leq_r$. Before the results of this chapter, no computable upper bound was known for the complexity of any of these classes. (Earlier work [ABK06a] did give an upper bound for a related class defined in terms of a very restrictive notion of reducibility: $\leq^{\mathrm{p}}_{dtt}$ reductions – but this only provided a characterization of P in terms of a class of polynomial-time reductions, which is much less compelling than giving a characterization where the set R is actually providing some useful computational power.)

The next two theorems are the main results from this chapter and show that the class of problems reducible to R_K in this way *does* have bounded complexity; hence it is at least plausible to conjecture that some complexity classes can be characterized in this way:

Theorem 2.1.11 ([AGF13]). $\Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^{\mathrm{p}} R_{K_U}\} \subseteq \mathrm{PSPACE}$

Theorem 2.1.12 ([AGF13]). $\Delta_1^0 \cap \bigcap_U \mathrm{NP}^{R_{K_U}} \subseteq \mathrm{EXPSPACE}$

A stronger inclusion is possible for "monotone" truth-table reductions ($\leq_{mtt}^{\mathrm{p}}$). We show that

Theorem 2.1.13 ([AGF13]). $\Delta_1^0 \cap \bigcap_U \{A : A \leq_{mtt}^{\mathrm{p}} R_{K_U}\} \subseteq \mathrm{coNP} \cap \mathrm{P/poly}$

Combining these results with Theorems 2.1.8 through 2.1.10 we now have:

- $\mathrm{BPP} \subseteq \Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^{\mathrm{p}} R_{K_U}\} \subseteq \mathrm{PSPACE} \subseteq \Delta_1^0 \cap \bigcap_U \mathrm{P}^{R_{K_U}}$.
- $\mathrm{NEXP} \subseteq \Delta_1^0 \cap \bigcap_U \mathrm{NP}^{R_{K_U}} \subseteq \mathrm{EXPSPACE}$.

In particular, note that PSPACE is sandwiched in between the classes of computable problems that are reducible to R_K via polynomial-time truth-table and Turing reductions.

Note that the use of the prefix complexity here *is* crucial. Although we believe that analogous theorems to all those stated here should hold if R_C is substituted for R_K, we do not know how to prove so. The explicit restriction to Δ_1^0 is also important. We believe these theorems hold even if "$\Delta_1^0 \cap$" is erased from the statement of the theorems. For instance, if A is in $\bigcap_U \mathrm{NP}^{R_{K_U}}$, we conjecture that A is computable. There are some preliminary results by Cai, Downey, Epstein, Lempp, and Miller in this direction, but currently this work is still unpublished [Mil13].

In the next section we prove the main theorems stated above, after which we try to lend some further perspective to their meaning.

2.2 Proofs of Main Results

2.2.1 Preliminaries

Before proving our main theorems from this chapter, we introduce a couple of extra definitions and prove a few propositions that will be needed.

Definition 2.2.1 (Overgraph). *If f is a function mapping some domain to the naturals $\mathbb{N}$, then $ov(f)$, the overgraph of f, is $\{(x, y) : f(x) \leq y\}$.*

For instance, $ov(K_U) = \{(x, y) :$ there exists an s, $|s| \leq y$, such that $U(s) = x\}$

The following definition was used implicitly by Muchnik and Positselsky [MP02]:

Definition 2.2.2 (Prefix Free Entropy Function). *A Prefix Free Entropy Function f is a function from $\{0,1\}^*$ to $\mathbb{N}$ such that*

- $\sum_{x\in\{0,1\}^*} 2^{-f(x)} \leq 1$ *and*
- $ov(f)$ *is computably enumerable (c.e.)*

Note that if f is a prefix free entropy function, then 2^{-f} is a special case of what Li and Vitányi call a *Lower Semicomputable Discrete Semimeasure* [LV08, Definition 4.2.2]. The *Coding Theorem* (see [LV08, Theorem 4.3.3]) says that, for any lower semicomputable discrete semimeasure 2^{-f} there is a universal prefix machine M such that $f(x) \leq K_M(x) - 3$. For the case of prefix free entropy functions, one can obtain a tighter bound (and replace the inequality with equality):

Proposition 2.2.3. *Let f be a prefix free entropy function. Given a machine accepting $ov(f)$, one can construct a prefix machine M such that $f(x) = K_M(x) - 2$.*

Proof. Our proof is patterned on the proof of [LV08, Theorem 4.3.3].

Since $ov(f)$ is c.e., there is a bijective enumeration function $D : \mathbb{N} \to \{(x, a) : f(x) \leq a\}$. Let $D(0) = (x_0, a_0), D(1) = (x_1, a_1,), \dots$ be this enumeration. We have that for each x, $\sum_{i\geq f(x)} 2^{-i} \leq 2 \cdot 2^{-f(x)}$ and therefore

$$\sum_{i\geq 0} \frac{1}{2} 2^{-a_i} = \sum_{x} \sum_{i\geq f(x)} \frac{1}{2} 2^{-i} \leq \sum_{x} 2^{-f(x)} \leq 1$$

We identify the set of infinite sequences $S = \{0,1\}^\infty$ with the half-open real interval $[0,1)$; that is, each real number r between 0 and 1 will be associated with the sequence(s) corresponding to the infinite binary expansion of r. We will associate each pair (x_i, a_i) from the enumeration D with a subinterval $I_i \subseteq S$ as follows:

$I_0 = [0, \frac{1}{2}2^{-a_0})$, and for $i \geq 1$, $I_i = [\sum_{k<i} \frac{1}{2}2^{-a_k}, \sum_{k\leq i} \frac{1}{2}2^{-a_k})$. That is, I_i is the half-open interval of length $\frac{1}{2}2^{-a_i}$ that occurs immediately after the interval corresponding to the pair (x_{i-1}, a_{i-1}) that appeared just prior to (x_i, a_i) in the enumeration D.

Since $\sum_{i\geq 0} \frac{1}{2}2^{-a_i} \leq 1$, each $I_i \subseteq S$.

Any *finite* string z also corresponds to a subinterval $\Gamma_z \subseteq S$ consisting of all infinite sequences that begin with z; Γ_z has length $2^{-|z|}$. Given any pair (x_i, a_i), one can determine the interval I_i and find the lexicographically first string z of length $a_i + 2$ such that $\Gamma_z \subseteq I_i$. Since I_i has length $2^{(-a_i+1)}$, it is not too difficult to see that such a string z must exist. (Alternatively, look at the proof of [LV08, Lemma 4.3.3].) Call this string z_i. Observe that, since the intervals I_i are disjoint, no string z_i is a prefix of any other.

We are now ready to present our prefix machine M. Let M be a machine that, given a string z, uses D to start an enumeration of the intervals I_i until it finds (x_i, a_i) such that $\Gamma_z \subseteq I_i$. If it ever finds such a pair, at this point M determines if $z = z_i$, and if so, outputs x_i. Otherwise the machine enters an infinite loop. Since no string z_i is a prefix of any other, M is a prefix machine.

Now consider the shortest string on which M will output x. Let $T = \{i : x_i = x\}$. For every $j \in T$ there exists a string z_j such that $M(z_j) = x$, and the length of z_j will be $a_j + 2$. We have that $\min_{j\in T} a_j = f(x)$, so $K_M(x) = f(x) + 2$. □

Proposition 2.2.4. *Let M and M' be prefix Turing machines. Then there is a prefix machine M'' such that $K_{M''}(x) = \min(K_M(x), K_{M'}(x)) + 1$.*

Proof. The domain of M'' is $\{1x : x \text{ is in the domain of } M\} \cup \{0x : x \text{ is in the domain of } M'\}$. □

Proposition 2.2.5. *Given any prefix machine M and constant c, there is a prefix machine M' such that $K_M(x) + c = K_{M'}(x)$*

Proof. The domain of M' is $\{0^c x : x \text{ is in the domain of } M\}$. $\square$

2.2.2 Overview of Proof of Main Theorem

We are ready to prove the main theorem of this chapter, which we now restate.

Theorem 2.2.6 (Restatement of Theorem 2.1.11)**.** $\Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^p R_{K_U}\} \subseteq \text{PSPACE}$.

Proof. The main idea of the proof can be seen as a blending of the approach of [ABK06a] with the techniques that Muchnik and Positselsky used to prove Theorem 2.7 of [MP02]. (See also [AFG10] for an alternative exposition of this theorem of Muchnik and Positselsky.)

We will actually prove the statement

$$\Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^p ov(K_U)\} \subseteq \text{PSPACE}. \tag{2.1}$$

The theorem follows, since any query "$x \in R_{K_U}$?" can always be modified to the equivalent query "$(x, |x| - 1) \notin ov(K_U)$?", so

$$\Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^p R_{K_U}\} \subseteq \Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^p ov(K_U)\}.$$

To prove the statement (1) it suffices to show that

$$L \in \Delta_1^0 - \text{PSPACE} \Rightarrow \exists \text{ a universal prefix machine } U \text{ s.t. } L \not\leq_{tt}^p ov(K_U). \tag{2.2}$$

Let $L \in \Delta_1^0 - \text{PSPACE}$ be given. It suffices to show how to incorporate a machine deciding membership in L into the construction of a universal prefix machine U such that $L \not\leq_{tt}^p ov(K_U)$. (As part of this construction, we use a diagonalization argument designed to foil every $\leq_{tt}^{\mathrm{p}}$ reduction.) To do this we will use the standard prefix complexity function K, together with a function $F : \{0,1\}^* \to \mathbb{N}$ that we will construct, to form a function $H : \{0,1\}^* \to \mathbb{N}$ with the following properties.

1. F is a total function and $ov(F)$ is c.e.

2. $H(x) = \min(K(x) + 5, F(x) + 3)$.

3. $\sum_{x \in \{0,1\}^*} 2^{-H(x)} \leq \frac{1}{8}$.

4. $L \not\leq_{tt}^{p} ov(H)$.

Claim 2.2.7. *Given the above properties, $H = K_{U'}$ for some universal prefix machine U' (which by Property 4 ensures that (2.2) holds).*

Proof. By Properties 2 and 3 we have that $\sum_{x \in \{0,1\}^*} 2^{-F(x)+3} \leq \frac{1}{8}$. Therefore $\sum_{x \in \{0,1\}^*} 2^{-F(x)}$ ≤ 1, which along with Property 1 means that F is a prefix free entropy function. By Proposition 2.2.3 we then have that $F+2$ is K_M for some prefix machine M. By Proposition 2.2.5 we have that $K(x)+4$ is $K_{U''}$ for some universal prefix machine U''. Therefore, by Proposition 2.2.4, $H(x) = \min(K(x)+5, F(x)+3) = \min(K(x)+4, F(x)+2)+1$ is $K_{U'}$ for some universal prefix machine U'. □

It remains to show that for a given computable set $L \notin$ PSPACE we can always construct functions H and F with the desired properties. Let us first informally discuss the ideas before providing the formal construction.

Our control over H comes from our freedom in constructing the function F. The construction will occur in stages – at any given time in the construction there will be a "current" version of F which we will denote by F^*. Similarly, there will be a "current" version of K denoted by K^*, which represents our knowledge of K at a given stage. At all times, H^*, our "current" version of H, will be defined as $\min(K^*(x)+5, F^*(x)+3)$.

Originally we set $F^*(x) = 2|x| + 3$ and K^* as the empty function. At each stage of the construction we will assume that a new element (x, y) is enumerated into $ov(K)$ according to some fixed enumeration of $ov(K)$. (This is possible since $ov(K)$ is c.e.) When this occurs K^* is updated by setting $K^*(x) = \min(K^*(x), y)$. (Since K^* is a partial function, it is possible that $K^*(x)$ was previously undefined. In this case we set $K^*(x) = y$.) Similarly, during the construction at times we will modify F by enumerating elements into $ov(F)$. Whenever we enumerate an element (x, y) into $ov(F)$, F^* is updated by setting $F^*(x) = \min(F^*(x), y)$.

Let $\gamma_1, \gamma_2, \ldots$ be a list of all possible polynomial time truth table reductions from L to $ov(H)$. This is formed in the usual way: we take a list of all Turing machines and put a clock of $n^i + i$ on the ith one and we will interpret the output as an encoding of a Boolean circuit on atoms of the form "$(z, r) \in ov(H)$".

We need to ensure that $L \not\leq_{tt}^{p} ov(H)$. We break this requirement up into an infinite number of requirements:

$$R_e : \gamma_e \text{ is not a polynomial-time tt-reduction of } L \text{ to } ov(H).$$

At stage e of the construction we will begin to attempt to satisfy the requirement R_e. For a particular input x, let $\gamma_e(x)$ be an encoding of a circuit $\lambda_{e,x}$. The output of the circuit $\lambda_{e,x}$ is determined by the truth values of the atoms "$(z, r) \in ov(H)$" that label the inputs to the circuit. Define $\lambda_{e,x}[H']$ to be the truth value obtained by taking the circuit $\lambda_{e,x}$ and for each atom "$(z, r) \in ov(H)$" using the truth value of "$(z, r) \in ov(H')$" in its place. In order to satisfy the requirement R_e, we would like to find some x such that $\lambda_{e,x}[H] \neq L(x)$, where $L(x)$ is the characteristic function of L. The problem is that at a given stage s we can "guess" at the value of $\lambda_{e,x}[H]$ by computing $\lambda_{e,x}[H^*]$, but in general we cannot know the value of $\lambda_{e,x}[H]$ for sure, because as H^* evolves the value of $\lambda_{e,x}[H^*]$ may change. The main difficulty is that the function K is out of our control and determining whether $(z, r) \in ov(H)$ is in general an uncomputable task.

We do have *some* influence over the situation though, due to our control of F. Indeed, for any atom "$(z, r) \in ov(H)$", we can ensure that the truth value of the atom is 1 by enumerating $(z, r - 3)$ into $ov(F)$. (Note that for all x, the value of $H^*(x)$ can only decrease over time). We have to be careful about making this type of change though; if we are too liberal in modifying F we may violate the condition $\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq 1/8$ in the process. Thus the construction becomes a balancing act – we will try to use F to satisfy R_e while at the same time maintaining the invariant that $\sum_{x\in\{0,1\}^*} 2^{-H^*(x)} \leq 1/8$. (In particular, if F_s is the function F^* at the beginning of stage s, for all x we will not want $\lim_{s\to\infty} F_s(x)$ to be very much smaller than $K(x)$).

As part of our solution, for each R_e we will find a suitable witness x and set up a game $\mathcal{G}_{e,x}$ played between us (making moves by enumerating elements into $ov(F)$), and K, who makes moves by enumerating elements into $ov(K)$. (Even though elements are obliviously enumerated into $ov(K)$ according to some fixed enumeration we will treat K as if it is a willful adversary). The witness x will be chosen so that we have a winning strategy; as long as K continues to make legal moves we can respond with changes to F (our own legal moves) that both assure that R_e is satisfied and that $\sum_{x\in\{0,1\}^*} 2^{-H^*(x)} \leq 1/8$. Our ability to find such a witness x follows from our assumption that the computable language L is not in PSPACE; if no such witness exists, then membership in L reduces to finding which player has a winning strategy in one of these games, which can be done in PSPACE.

It is possible that K will cheat by enumerating elements into $ov(K)$ in such a way that it plays an illegal move. In this case we will simply destroy the game $\mathcal{G}_{e,x}$ and start all over again with a new game $\mathcal{G}_{e,x'}$, using a different witness x'. However we will be able to show that if K cheats infinitely often on games associated with a particular requirement R_e, then $\sum_{x\in\{0,1\}^*} 2^{-K(x)}$ diverges. This contradicts K being a prefix complexity function. Hence K can only cheat finitely often.[1]

The requirements $R_1, R_2, R_3, \ldots$ are listed in priority ordering. If during stage s a move is played on a game $\mathcal{G}_{e,x}$, we say that R_e is "acting". In this case for all $e < e' \leq s$, if $\mathcal{G}_{e',y}$ is the game associated with $R_{e'}$ currently being played, we destroy this game and start a new game $\mathcal{G}_{e',y'}$ with some new witness y'. When this happens we say that each of the $R_{e'}$ has been "injured" by R_e. The reason this works in the end is that at some point $R_1, R_2, \ldots, R_{e-1}$ have stopped acting, so R_e will no longer ever be injured by some higher priority requirement.

2.2.3 Description of the Games

Now let us describe one of the games $\mathcal{G}_{e,x}$ in more depth and provide some analysis of the game. Let the inputs to the Boolean circuit $\lambda_{e,x}$ (encoded by $\gamma_e(x)$) be labeled by the atoms $\{(z_1, r_1), \ldots, (z_k, r_k)\}$. Let $X_e = \{z_1, \ldots, z_k\}$. Note that the queries

[1]This reliance on the convergence of $\sum_{x\in\{0,1\}^*} 2^{-K(x)}$ is the key reason why our proof does not go through in the case where we consider the plain complexity $C(x)$ as opposed to the prefix complexity $K(x)$.

in this reduction are of the form: "Is $H(z_i) \leq r_i$?". If $H^*(z_i) \leq r_i$ then we already know $H(z_i) \leq r_i$, so we can replace that input to the circuit with the value *TRUE* and simplify the circuit accordingly. Renumber the z's, rename k to again be the number of questions, and rename X_e to be the set of all z's being asked about. When we are done we have atoms $\{(z_1, r_1), \ldots, (z_k, r_k)\}$ and we know that $(\forall z_i \in X_e)[H^*(z_i) > r_i]$.

We make one more change to X_e. If there exists an element z_i such that $z_i \in X_e$ and $z_i \in X_{e'}$ for some $e' < e$, then changing H^* on the value z_i during the game $\mathcal{G}_{e,x}$ could affect the game associated with the requirement $R_{e'}$, which would upset our priority ordering. Hence we will take

$$X_e = X_e - \bigcup_{e'<e} X_{e'}.$$

This will ensure that R_e cannot injure any $R_{e'}$ with $e' < e$.

While working on requirement R_e we will need to evaluate the circuit $\lambda_{e,x}$. This will involve answering queries of the form $H(z) \leq r$. There will be two types of queries:

- If $z \in \bigcup_{e'<e} X_{e'}$ then we answer FALSE, which is the correct value unless the appropriate $R_{e'}$ acts. However, if this occurs, then all of the work done on R_e will be wiped out anyway.

- If $x \in X_e$ then we answer with the status of $H^*(z) \leq r$. The key is that we have some control over this if the answer is FALSE and may purposely change it.

Let $H^*_{e,x}$ be the function H^* when the game $\mathcal{G}_{e,x}$ is first constructed. Let $\epsilon = 2^{-e-i_e-6}$. (How i_e is determined will be explained later). The game $\mathcal{G}_{e,x}$ is played on a labeled DAG. The label of each node of the DAG has the following two parts:

1. A function h that maps X_e to $\mathbb{N}$. The function h provides conjectured values for H restricted to X_e. The function h will be consistent with $H^*_{e,x}$ in that $(\forall i)[h(z_i) \leq H^*_{e,x}(z_i)]$.

2. A truth value *VAL*, which is the value of $\lambda_{e,x}$ assuming that $(\forall z \in X_e)[H(z) = h(z)]$. Note that this will be either YES or NO indicating that either, under assumption $(\forall z \in X_e)[H(z) = h(z)]$, $\lambda_{e,x}$ thinks $x \in L$ or thinks $x \notin L$.

There is a separate node in the DAG for every possible such function h.

Let us place an upper bound on the size of this DAG. The set X_e contains at most $|x|^e$ queries. For any query z_i, $H(z_i)$ can take at most $2|z_i|+6$ values (since it is always bounded by $F^*(z_i)+3$). Note also that $|z_i| \leq |x|^e$. Thus there are at most $(2|x|^e+6)^{|x|^e}$ possible choices for h. For all large x this is bounded by $2^{|x|^{2e}}$, so note that we can represent a *particular* node in the DAG with $|x|^{2e}+1$ bits.

We now describe the start node and how to determine the edges of the DAG.

1. There is a node (h, VAL) where $h = H^*_{e,x}$ restricted to X_e. This is the start node and has indegree 0.

2. There is an edge from (h, VAL) to (h', VAL') if for all $z_i \in X_e$, $h(z_i) \geq h'(z_i)$ (so it is possible that H^* could at some point evolve from $H^*_{e,x}$ to h, and then at a later point evolve from h to h'.)

The game $\mathcal{G}_{e,x}$ is played between two players, the YES player and the NO player. Each player has a score, which originally is zero, and represents how much the player has been penalized so far in the game. (In other words a high score is bad). The game starts with a token placed on the start node. The YES player goes first (although this choice is arbitrary), after which the players alternate moves.

On a given turn a player can either leave the token where it is or move the token to a new node in the DAG. Suppose a player moves the token from a node t to a node t', where h is the function labeling t and h' is the function labeling t'. In this case we add $\sum_{z_i \in X_e}(2^{-h'(z_i)} - 2^{-h(z_i)})$ to the player's score.

A player can legally move the token from node t to t' if

1. There is an edge from t to t' in the game DAG.

2. The score of the player after making the move does not exceed ϵ.

The YES player wins if the token ends up on a node such that $VAL =$ YES, and the NO player wins if the token ends up on a node such that $VAL =$ NO. Note that because the game is entirely deterministic, for a given game $\mathcal{G}_{e,x}$, either the YES player has a winning strategy or the NO player has a winning strategy. Let $val(\mathcal{G}_{e,x}) = 1$ if the YES player has a winning strategy on the game $\mathcal{G}_{e,x}$ and $val(\mathcal{G}_{e,x}) = 0$ otherwise.

During the actual construction the games will be played between us (the construction) trying to make the computation go one way, and K (which we do not control) trying to make it go (perhaps) another way. We will always ensure that we play the side of the player who has the winning strategy in the game. We will effect our moves by enumerating elements into $ov(F)$, which changes F^* and hence H^*. (To move the token to a node labeled with the function h, we modify H^* so that h equals H^* restricted to the set X_e) The K moves will occur when a new element is enumerated into $ov(K)$ at the beginning of each stage, which changes K^* and hence H^*. (In this case K is moving the token to the node in the game DAG labeled by the new H^*).

The key is that the players' scores measure how much the sum $\sum_{x\in\{0,1\}^*} 2^{-H^*(x)}$ has gone up, which we bound by not allowing a player's score to exceed ϵ. (Of course K is oblivious to the rules of the game and will at times cheat – we take this into account as part of our analysis.) One final note: it is possible that K will simply stop playing a game in the middle and never make another move. This will not matter to us in the construction; what is important is that we have a winning strategy and if K does move we always have a winning response.

2.2.4 The Formal Construction

We now present the formal details of the stage construction.

Stage 0:

- Let F^* initially be defined as $F^*(x) = 2|x| + 3$.
- Let K be the standard prefix complexity function, and K^* initially be the empty function.
- At all times throughout the construction, we have that $H^*(x) = \min(K^*(x) + 5, F^*(x) + 3)$. (In the case where $K^*(x)$ is undefined, let $H^*(x) = F^*(x) + 3$). We will define H_s to be the function H^* as it is at the beginning of stage s.

- For all e, set $i_e = 0$. In the future i_e will be the number of times R_e has been injured by the requirements $R_{e'}$, $1 \leq e' \leq e-1$.
- Let $HEAP$ be an object that enumerates strings in the normal lexicographical order. So the first time that $HEAP$ is called it returns the string '0', the second time it returns '1', then '00', '01', etc.

Stage s (for $s \geq 1$):

Let (x', y') be the sth element in the fixed enumeration of $ov(K)$. Update K^* by setting $K^*(x') = \min(K^*(x'), y')$. (This automatically updates H^* as well)

(**) For $1 \leq e \leq s$ we consider requirement R_e.

There are two possibilities:

1. There is no game associated with R_e in progress. This can occur because either $e = s$ or because the game associated with R_e was destroyed during the last round.

   In this case we continue to get strings from $HEAP$ until a string x is found that has the following property:

   - If we define a new game $\mathcal{G}_{e,x}$ using the current H^*, then $val(\mathcal{G}_{e,x}) \neq L(x)$, where $L(x)$ is the characteristic function of L.

   We will later show in Claim 4 that in a finite number of steps we will always find such an x.

   Once we have found the string x, construct the game $\mathcal{G}_{e,x}$ in the way described in the previous section and begin the game. For this game, we will play as the YES player if $val(\mathcal{G}_{e,x}) = 1$, and as the NO player if $val(\mathcal{G}_{e,x}) = 0$. (That is, we will always play as the player who has a winning strategy for the game).

2. The game associated with R_e is already in progress (again call this game $\mathcal{G}_{e,x}$).

   There are a few sub-cases to consider.

   (a) K has not changed on X_e at all since the last stage. We do nothing.

(b) K on X_e has changed so much that the move K plays causes his score to exceed ϵ (i.e. K "cheats"). In this case we destroy the game $\mathcal{G}_{e,x}$.

(c) It is our turn in $\mathcal{G}_{e,x}$, either because the token is on the start node of the DAG and we are the YES player, or because K on X_e has changed in a way that his score does not exceed ϵ, so he has played a legal move.

In this case we play the move dictated by our winning strategy (which we can assume we have stored or which we can recompute each time). This may be to do nothing or it may involve moving the token to a new node, in which case we change H^* accordingly by enumerating elements into $ov(F)$.

If either case (b) or (c) occurs, we say that "R_e is acting", in which case for all e' such that $s \geq e' \geq e+1$: Set $i_{e'}$ to $i_{e'}+1$ and destroy the game associated with $R_{e'}$. Note: i_e does *not* change in case (b), even though $\mathcal{G}_{e,x}$ is destroyed.

If R_e acts then proceed to the next stage. Otherwise return to (**) and process the next e.

END OF CONSTRUCTION

2.2.5 Wrapping up the Proof of the Main Theorem

Claim 2.2.8. *For all e, each R_e acts at most finitely often and is satisfied.*

Proof. We prove this by induction on e. Assume that the claim is true for all $e' < e$. We show that the claim holds for e. By the inductive hypothesis there exists a stage s' such that, for all $s \geq s'$, for all $e' < e$, $R_{e'}$ does not act at stage s.

Let $\mathcal{G}_{e,x}$ be the game associated with R_e at stage s. If $\mathcal{G}_{e,x}$ is never destroyed in a later stage, then (by construction) R_e will be satisfied (since for $H = \lim_{s\to\infty} H_s$, our winning strategy ensures that $\gamma_{e,x}[H]$ evaluates to YES if and only if x is not in L).

Suppose that $\mathcal{G}_{e,x}$ is destroyed at some point. Then, since by the inductive hypothesis R_e cannot be injured by higher priority requirements, by the rules of the construction it must be that the "player" K cheats on the game $\mathcal{G}_{e,x}$. In doing this, K is adding at least $\epsilon = 2^{-e-i_e-6}$ to $\sum_{x\in X_e} 2^{-K^*(x)}$ and hence to $\sum_{x\in\{0,1\}^*} 2^{-K^*(x)}$.

Once K cheats and destroys the game $\mathcal{G}_{e,x}$, a new witness x' is found and a new game $\mathcal{G}_{e,x'}$ is started during the next stage. Once again if this game is never destroyed then R_e will be satisfied. If this game is also later destroyed, this means that another $\epsilon = 2^{-e-i_e-6}$ is added to $\sum_{x\in\{0,1\}^*} 2^{-K^*(x)}$. The crucial observation is that since i_e did not change, this is the same ϵ as before.

This process keeps repeating. If the games associated with R_e continue to be destroyed indefinitely, then $\sum_{x\in\{0,1\}^*} 2^{-K(x)} \geq \epsilon + \epsilon + \cdots$ so it diverges. This contradicts K being a prefix free entropy function.

Hence eventually there is some game $\mathcal{G}_{e,x''}$ that is played throughout all the rest of the stages. Since the game DAG for $\mathcal{G}_{e,x''}$ is finite, this means that eventually R_e stops acting and is satisfied.

□

Claim 2.2.9. $\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq \frac{1}{8}$.

Proof. We have that $H = \lim_{s\to\infty} H_s$, and thus

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} = \sum_{x\in\{0,1\}^*} 2^{-H_0(x)} + \sum_{s\geq 1}\sum_{x\in\{0,1\}^*} (2^{-H_{s+1}(x)} - 2^{-H_s(x)}).$$

That is, we can bound the sum by bounding H_0 and by bounding the changes that occur to H over the lifetime of the construction (some of which are made by K, and some by F).

Originally $H^*(x) = F^*(x) + 3 = 2|x| + 6$ for all x, so $\sum_{x\in\{0,1\}^*} 2^{-H_0(x)} = \frac{1}{32}$.

The total contribution that K can make to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ is bounded by the expression $\sum_{x\in\{0,1\}^*} 2^{-K(x)+5}$. Since K is a prefix free entropy function, this contribution is at most $1/32$.

Let us now consider the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ due to movements by F on games on which K eventually cheats. On each of these games F contributes less to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ than K, so from the above we can say that the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ while playing these games is at most $1/32$.

Finally, let us consider the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ due to movements by F on games that are never destroyed, or are destroyed by higher priority requirements. Consider such games associated with a particular requirement R_e. During the first such game associated with R_e, $i_e = 0$, so F can change at most $\epsilon = 2^{-e-i_e-6} = 2^{-e-6}$. On the second such game associated with R_e, $i_e = 1$, so F can change at most $\epsilon = 2^{-e-7}$. Generalizing, we see that the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ on such games associated with R_e is

$$\sum_{i=6}^{\infty} 2^{-e-i} = 2^{-e}\sum_{i=6}^{\infty} 2^{-i} = 2^{-e-5}$$

Hence, the total contribution that F makes to $\sum_{x\in\{0,1\}^*} 2^{-H(x)}$ on games that are never destroyed, or are destroyed by higher priority requirements is at most $\sum_{e=1}^{\infty} 2^{-e-5} = \frac{1}{32}$.

Putting all this information together we have that

$$\sum_{x\in\{0,1\}^*} 2^{-H(x)} \le \frac{1}{32} + \frac{1}{32} + \frac{1}{32} + \frac{1}{32} = \frac{1}{8}$$

□

All that remains is to show that whenever a new game associated with R_e is constructed, a witness x with the appropriate property can be found. Recall that we are searching for an x such that

- If we define a new game $\mathcal{G}_{e,x}$ using the current H^*, then $val(\mathcal{G}_{e,x}) \neq L(x)$, where $L(x)$ is the characteristic function of L.

Claim 2.2.10. In the above situation, a witness with the desired property can always be found in a finite number of steps

Proof. Suppose for contradiction that during some stage s for some e we are not able to find such an x. Let y be the last string that was taken from $HEAP$ before this endless search for an x began. This means that for all strings $x > y$ (under the normal lexicographical ordering), when we construct the game $\mathcal{G}_{e,x}$, $val(\mathcal{G}_{e,x}) = L(x)$. But this gives a PSPACE algorithm to decide L, which we now describe.

Hardwire in the value of $L(x)$ for every $x \leq y$. Also hardwire in the function H^* at this moment in the construction and $X_{e'}$ for all $e' \leq e$. (It is possible to hardwire in H^* because at any given moment in the construction only finitely many elements have been enumerated into $ov(F)$ and $ov(K)$.)

On an input $x \leq y$, refer to the lookup table to decide $L(x)$. On an input $x > y$, use the stored values of H^* and the $X_{e'}$'s to construct $\mathcal{G}_{e,x}$ and output $val(\mathcal{G}_{e,x})$. As noted previously, for all large x we can represent a *particular* node in the DAG of $\mathcal{G}_{e,x}$ with $|x|^{2e}+1$ bits. Despite the fact that there are exponentially many nodes in the graph, an alternating polynomial-time Turing machine can search for winning strategies on the DAG, as follows:

Note that the number of moves in the game is bounded by a polynomial in $|x|$, since each move in the game involves lowering the value of $H^*(z)$ for one of the polynomially-many queries z. Thus, to determine if a player has a winning strategy from some game position C (represented by the circuit $\lambda_{e,x}$, along with the values of H^* restricted to the set X_e that is used in the game $\mathcal{G}_{e,x}$), it suffices to check if there *exists* a move for this player causing the circuit $\lambda_{e,x}$ to take on the desired value, and such that *for every* move of the opposing player, there *exists* a move of this player that again causes $\lambda_{e,x}$ to take on the desired value, such that ... until there are no more legal moves possible for the opposing player. We can represent any state of the game (i.e., the node where the token currently lies, plus the scores of the players) by a number of bits bounded by a polynomial in $|x|$. Given the functions h and h' for any two nodes in the DAG, along with the scores of each player, it is easy to determine in polynomial time if it is legal to move from h to h', and to compute the scores of each player after the move. (It suffices to verify that for all z, $h(z) \leq h'(z)$, and to add up a polynomial number of rationals of the form $a/2^b$ where $b = n^{O(1)}$.) As mentioned above, the length of any path in the DAG is bounded by a polynomial in n (since the values of h always decrease). Thus, determining winning strategies is possible in alternating polynomial time.

Since alternating polynomial time is equal to PSPACE [CKS81], this contradicts the fact that $L \notin$ PSPACE.

$\square$

This concludes the proof of Theorem 2.2.6. □

2.2.6 Proofs of Other Main Results

We now restate and prove the other main results from this chapter.

Theorem 2.2.11 (Restatement of Theorem 2.1.13)**.** $\Delta_1^0 \cap \bigcap_U \{A : A \leq^{\mathrm{p}}_{mtt} R_{K_U}\} \subseteq \mathrm{coNP} \cap \mathrm{P/poly}$.

Proof. The containment in P/poly comes from [ABK06a].

Note that a reduction showing $L \leq^{\mathrm{p}}_{mtt} R_{K_U}$ corresponds to an *anti-monotone* reduction to $ov(K_U)$ (where the only queries are of the form "Is $K_U(z) < |z|$?") Thus this same reduction is an anti-monotone reduction from the complement of L to the complement of R_{K_U}. If we replace each Boolean function in this anti-monotone reduction with its complement, we obtain a *monotone* reduction of L to $ov(K_U)$.

Thus it suffices to show that any set that is $\leq^{\mathrm{p}}_{mtt}$-reducible to the overgraph $ov(K_U)$ for every U is in NP.

The proof of this containment is almost identical to the proof of Theorem 2.2.6. The only difference is now we consider an arbitrary language $L \notin \mathrm{NP}$, and must show that when a game $\mathcal{G}_{e,x}$ is constructed corresponding to a polynomial time *monotone* truth table reduction γ_e, determining whether $val(\mathcal{G}_{e,x}) = 1$ can be computed in NP. Note that in the monotone case, the NO player of the game has no incentive to ever make a move, as doing so could only change the value of the circuit $\lambda_{e,x}$ from NO to YES. Therefore whether the YES player has a winning strategy in the game depends solely on whether the YES player can legally move the token from the start node to a node u in the game DAG labeled by YES. This is an NP question – the certificate is the node u, which as we have seen can be represented by a polynomial number of bits in $|x|$.

□

Theorem 2.2.12 (Restatement of Theorem 2.1.12)**.** $\Delta_1^0 \cap \bigcap_U \mathrm{NP}^{R_{K_U}} \subseteq \mathrm{EXPSPACE}$.

Proof. An NP-Turing reduction can be simulated by a truth-table reduction computable in exponential time, where all queries have length bounded by a polynomial in the input length. Carrying out the same analysis as in the proof of Theorem 2.2.6, but changing the time bound on the truth-table reductions from polynomial to exponential, immediately yields the EXPSPACE upper bound.

□

2.3 Encoding in the Overgraph

We conjecture that our main results can be improved in several ways. In this section, we consider one type of improvement, and we present some reasons why a different proof strategy will be required in order to obtain such an improvement.

Theorem 2.2.6 shows that, for every decidable set A outside PSPACE, there is *some* universal prefix machine U such that $A \not\leq_{tt}^{p} R_{K_U}$. It is natural to ask if there is a decidable set A such that, for *every* universal machine U, $A \not\leq_{tt}^{p} R_{K_U}$. It is plausible that this holds for every decidable set A that is not in P/poly, a hypothesis that we explore more closely in the next chapter. (Some weaker results in this direction have been proved. It is known that if a decidable set A has high-enough non-uniform complexity, then for *any* universal machine U, any $\leq_{tt}^{\mathrm{p}}$ reduction from A to R_{C_U} must make at least $n/4 \log n$ queries [ABK06a]. Related questions were also explored by Hitchcock [Hit10].)

However, the following theorem shows that no such improvement can carry over to the *overgraph* $ov(R_{K_U})$, which suggests that quite different techniques may be required than were employed in this chapter. (Related observations occur in [MP02, Theorem 2.6].)

Theorem 2.3.1. *Let A be a computable set. Then there exists a universal (prefix) machine U such that $A \leq_{tt}^{\mathrm{p}} ov(C_U)$ ($A \leq_{tt}^{\mathrm{p}} ov(K_U)$, respectively).*

Proof. We present the proof for $ov(K_U)$. It is clear that the proof carries over also for the case of $ov(C_U)$.

Let M be the universal prefix machine that defines $K(x)$.

Consider the machine U that does not halt on any input in $00\{0,1\}^*$, and behaves as follows on inputs of the form $1d$ or $01d$ for any string d:

1. Simulate $M(d)$, and if it halts, let the output be x (so that $K(x) \leq |d|$).

2. Determine if $x \in A$ or not.

3. (a) If $x \in A$ and $|d|$ is even, then $U(1d) = x$ and $U(01d) = \uparrow$.

   (b) If $x \in A$ and $|d|$ is odd, then $U(1d) = \uparrow$ and $U(01d) = x$.

   (c) If $x \notin A$ and $|d|$ is even, then $U(1d) = \uparrow$ and $U(01d) = x$.

   (d) If $x \notin A$ and $|d|$ is odd, then $U(1d) = x$ and $U(01d) = \uparrow$.

Note that U is a prefix machine, and that for all x $K_U(x) \leq K(x) + 2$.

Clearly, $x \in A$ if and only if $K_U(x)$ is odd. This can be determined by making a linear number of truth-table queries to $ov(K_U)$.

□

2.4 Perspective and Open Problems

How should one interpret the theorems presented here?

Prior to this work, the inclusion NEXP $\subseteq$ NP^{R_K} was just a curiosity, since it was not clear that it was even meaningful to speak about efficient reductions to an undecidable set. Here, we show that if we view R_K not as merely a single undecidable set, but as a *class* of closely-related undecidable sets (differing only by the "insignificant" choice of the universal Turing machine U), then the computable sets that are always in NP^{R_K} *is* a complexity class sandwiched between NEXP and EXPSPACE. The obvious question is whether this class is actually *equal* to NEXP (or to EXPSPACE). Any characterization of a complexity class in terms of efficient reductions to a class of undecidable sets would raise the possibility of applying techniques from computability theory to questions in complexity theory, where they had seemed inapplicable previously.

One possible objection to the theorems presented here is that they make use of universal Turing machines U that are far from "natural". However, we see little to be gained in trying to formulate a definition of a "natural" universal Turing machine. Even basic questions such as whether there is a truth-table reduction from the Halting Problem to R_K depend on the choice of the universal Turing machine U [MP02, ABK06a], and the only machines for which the answer is known (positive and negative) are all decidedly “unnatural”. A detailed study of analogous questions that arise when one considers other variants of Kolmogorov complexity (such as various types of “monotone” Kolmogorov complexity) has been carried out by Day [Day09].

All of the positive results, showing that problems *are* efficiently reducible to R_K hold using a quite general notion of “universal Turing machine”, and we believe that the approach used here and in [ABK06a] to “factor out” the idiosyncrasies of individual universal machines is a more productive route to follow.

Alternatively, one can view our results as placing limits on what sets can be *proved* to be reducible to R_K, using only an “axiomatic” approach to Kolmogorov complexity. Let us expand on this view. The theory of Kolmogorov complexity (for instance, as developed in [LV08]) relies on the existence of universal machines U, in order to develop the measures K and C, but no properties are required of such machines U, other than that, for any other (prefix) machine M, $\exists c_M \forall x C_U(x) \leq C_M(x) + c_M$ (or $K_U(x) \leq K_M(x) + c_M$, respectively). The rest of the theory can proceed, using just this axiom about the machine U that defines C or K.

Our results show that, for any decidable set A outside of EXPSPACE, A cannot be proven to lie in NP^{R_K} without introducing additional axioms describing additional properties of the universal machine that defines K. One could pursue the same types of questions that we do in this chapter using a more stringent definition of what constitutes a universal machine, but at the cost of adding additional axioms to the study of Kolmogorov complexity that in some sense are not strictly necessary.

As mentioned in the introduction to this chapter, we conjecture that our main theorems hold even if “$\Delta_1^0 \cap$” is erased from the statement of the theorems, and if the plain complexity is substituted for the prefix complexity. However, the proof technique used in this chapter makes explicit use of the computability of the language L during the diagonalization construction, and also relies on the fact that $K(x)$ is a prefix-entropy function, so currently these problems remain open.

The theorems presented here all relativize. For instance, for any computable oracle B, if $A \notin \mathrm{PSPACE}^B$, then there is a universal prefix Turing machine U such that

$A \not\leq_{tt}^{\mathrm{P}^B} R_{K_U}$. (Note that, for computable oracles B, there is no need to "relativize" R_{K_U}. A similar restatement is possible for noncomputable oracles B, too.) However, it seems quite possible to us that, say, if it were possible to *characterize* NEXP in terms of NP^{R_K}, that this might proceed via nonrelativizable techniques. Whether or not these hopes come to fruition, the types of characterizations of complexity classes investigated in this chapter are quite different than those that have been studied in the past, and we hope that new insights will result as new connections are explored.

Chapter 3

The Time-Bounded Kolmogorov Complexity Case

3.1 Introduction

Let us for notational convenience define a shorthand for one of the new complexity classes that was introduced in the previous chapter:

Definition 3.1.1 (Deterministic Truth-Table Reductions to the Random Strings: DTTR)**.** $\mathsf{DTTR} = \Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^{\mathrm{p}} R_{K_U}\}$

Recall that combining the main theorem of the last chapter with previous results, we were able to derive the following chain of inclusions:

$$\mathrm{BPP} \subseteq \mathsf{DTTR} \subseteq \mathrm{PSPACE} \subseteq \mathrm{P}^{R_K}$$

A natural question is whether DTTR sits closer to BPP or PSPACE. In [ADF$^+$12] it was conjectured that DTTR actually characterizes BPP exactly, the idea being that a polynomial-time truth-table reduction from R_K to a decidable language can only exploit R_K as a source of pseudorandomness. As mentioned previously, proving such a conjecture would allow us to use tools from Kolmogorov complexity to study questions about the class BPP, and as we will discuss in section 3.6, could possibly even be a route towards proving BPP = P that avoids some known barriers.

In [ADF$^+$12] it was suggested that a good first step towards proving $\mathsf{DTTR} = \mathrm{BPP}$ would be to improve the upper bound from $\mathsf{DTTR} \subseteq \mathrm{PSPACE}$ to $\mathsf{DTTR} \subseteq \mathrm{PSPACE} \cap \mathrm{P/poly}$. Because there are not really natural complexity classes that lie between BPP and $\mathrm{PSPACE} \cap \mathrm{P/poly}$, this would be evidence that $\mathrm{BPP} = \mathsf{DTTR}$. [ADF$^+$12] actually lays out an entire strategy for getting this improved upper bound. The idea is to show that any machine M computing a polynomial-time truth-table reduction from R to a decidable language L cannot make use of queries of length greater than $O(\log n)$. This would immediately imply a P/poly upper-bound, as we can encode the answers to all queries of length at most $O(\log n)$ as an advice string of length poly(n), and then give this advice string to the machine M in place of the oracle R. (The machine M would answer NO to any of its queries about whether a string of length greater than $O(\log n)$ was in R). [ADF+12] also linked these problems to questions in logic by demonstrating that if certain true statements were provable in various formal systems of arithmetic, then the $\mathsf{DTTR} \subseteq$ P/poly inclusion would follow via the above strategy.

In this chapter we show that the above approach must fail, or at least that it requires significant changes. Interestingly, we can also prove that this intuition — that the large queries can be answered with NO — *can* be used to show the P/poly inclusion if we use a *time-bounded* variant of Kolmogorov complexity, K^t. While demonstrating this discrepancy we show several other ways in which reductions to R_K and R_{K^t} are actually very different; in particular, we construct a counter-intuitive example of a polynomial-time non-adaptive reduction that distinguishes R_K from R_{K^t}, for any sufficiently large time-bound t.

To investigate the time-bounded Kolmogorov complexity setting we define a class TTRT as a time-bounded analog of DTTR; informally, TTRT is the class of problems that are polynomial-time truth-table reducible to R_{K^t} for every sufficiently fast-growing time-bound t, and every "time-efficient" universal Turing machine used to define K^t. We prove that, for all monotone nondecreasing computable functions $\alpha(n) = \omega(1)$,

$$\text{BPP} \subseteq \mathsf{TTRT} \subseteq \text{PSPACE}/\alpha(n) \cap \text{P/poly}.$$

Here, PSPACE/$\alpha(n)$ is a "slightly non-uniform" version of PSPACE. We believe that this indicates that TTRT is "closer" to BPP than it is to PSPACE.

It would be more appealing to avoid the advice function, and we are able to do so, although this depends on a fine point in the definition of time-efficient prefix-free Kolmogorov complexity. This point involves a subtle technical distinction, and will be left for the appropriate section. To summarize:

- In Section 3.3 we prove that $\mathsf{TTRT} \subseteq$ P/poly, by using the same basic idea of

[ADF+12]. We further show, however, that this approach will not work to prove $\mathsf{DTTR} \subseteq \mathrm{P/poly}$, and by reversing the logic connection of [ADF+12], this will give us an independence result in certain extensions of Peano arithmetic.

- Then in section 3.4 we prove that $\mathsf{TTRT} \subseteq \mathrm{PSPACE}/\alpha(n)$, which is a non-trivial adaptation of the techniques from [AGF13]. In section 3.5 we show how to get an analogous result without the super-constant advice term.

In section 3.6 and section 3.7 we discuss prospects for future work.

3.2 Preliminaries

For a set A of strings, let $A^{\leq n}$ denote the set of all strings of length at most n in A. In this chapter, a function $t : \mathbb{N} \to \mathbb{N}$ is called a "time-bound" if it is non-decreasing and time-constructible. We use the following time-bounded version of Kolmogorov complexity:

Definition 3.2.1 (Time-bounded Kolmogorov Complexity)**.** *For a prefix machine M and a time-bound t, $K^t_M(x)$ is the length of the smallest string y such that on input y M halts in fewer than $t(|x|)$ time steps and outputs x.*

Let us define what it means for a machine to be "universal" in the time-bounded setting:

Definition 3.2.2. *A prefix machine U is a time-efficient universal prefix machine if there exist constants c and c_M for each prefix machine M, such that*

1. *$\forall x$, $K_U(x) \leq K_M(x) + c_M$, and*
2. *$\forall x$, $K^t_U(x) \leq K^{t'}_M(x) + c_M$ for all $t > t'^c$.*

Then $R_{K^t_U}$ is the set of K^t_U-*random* strings: $\{x | K^t_U(x) \geq |x|\}$.

Now we can formally define the time-bounded analogue of DTTR:

Definition 3.2.3. TTRT *is the class of languages L such that there exists a time bound t_0 (depending on L) such that for all time-efficient universal prefix machines U and for all time-constructible $t \geq t_0$, $L \leq^p_{tt} R_{K^t_U}$.*

The proof of Corollary 12 in [BFKL10] shows that, if $t \geq t_0 = 2^{2^{2n}}$, then BPP $\leq_{tt}^{p}$ $R_{K_U^t}$, for any time-efficient universal U. This implies:

Theorem 3.2.4 ([BFKL10])**.** BPP $\subseteq$ TTRT.

Now we prove some basic facts about time-bounded prefix-free Kolmogorov complexity.

Proposition 3.2.5. *For any machine M and $t'(|x|) > 2^{|x|}t(|x|)$, the answer to the query "$x \in R_{K_M^t}$?" can be computed in time t'.*

Proof. Simulate the machine M on every string of length less than $|x|$ for $t(|x|)$ steps. Because there are fewer than $2^{|x|}$ such strings, the bound follows. □

Proposition 3.2.6. *Let $L \leq_{tt}^{p} R_{K_U^t}$ for some time-bound t. Then there exists a constant k such that the language L can be computed in $t_L(n) = 2^{n^k}t(n^k)$ time.*

Proof. Let M be a machine that computes L by running the polynomial-time truth-table reduction from L to $R_{K_U^t}$, and computing by brute-force the answer to any queries from the reduction. Using Proposition 3.2.5, we have that for large enough k, M runs in at most $t_L(n) = 2^{n^k}t(n^k)$ time, so L is decidable within this time-bound. □

It is the ability to compute R_{K^t} for short strings that makes the time-bounded case different from the unbounded case. This will be seen in proofs throughout the chapter.

3.3 How and Why to Distinguish R_K from R_{K^t}

At first glance, it seems reasonable to guess that a polynomial-time reduction would have difficulty telling the difference between an oracle for R_K and an oracle for R_{K^t}, for large enough t. Indeed $R_K \subseteq R_{K^t}$ and in the limit for $t \to \infty$ they coincide.

One might even suspect that a polynomial-time reduction must behave the same way with R_{K^t} and R_K as oracle, already for modest time bounds t. However, this intuition is wrong. Here is a counter-example for adaptive polynomial-time reductions.

Observation 3.3.1. *There is a polynomial-time algorithm which, given oracle access to R_K and input 1^n, outputs a K-random string of length n. However, for any time-bound t such that $t(n+1) \gg 2^n t(n)$, there is no polynomial-time algorithm which, given oracle access to R_{K^t} and input 1^n, outputs a K^t-random string of length n.*

For the algorithm, see [BFNV05]; roughly, we start with a small random string and then use [BFNV05, Theorem 15] (described later) to get a successively larger random string. But in the time-bounded case in [BM97] it is shown that on input 1^n, no polynomial-time machine M can query (or output) any K^t-random string of length n: in fact, $M(1^n)$ is the same for both oracles R_{K^t} and $R' = R_{K^t}^{\leq n-1}$. This is proven as follows: since R' can be computed in time $t(n)$ (by Proposition 3.2.5), then any query of length $\geq n$ made by $M^{R'}(1^n)$ is described by a pointer of length $O(\log n)$ in time $t(n)$, and hence is not in R_{K^t}.

3.3.1 Small Circuits for Sets Reducible to R_{K^t}

We now prove that TTRT is a subset of P/poly. Actually, we will prove that this holds even for *Turing* reductions to R_{K_U}, (for *every* universal Turing machine U):

Theorem 3.3.2 ([ABFL12a]). *Suppose $A \in \mathsf{DTIME}(t_1)$ and $M : A \leq_T^p R_{K^t}$, for some time-bounds t, t_1 with $t(n+1) \geq 2^n t(n) + 2^{2^n} t_1(2^n)$.*[1] *Then $A \in \mathrm{P/poly}$; in fact, if M runs in time n^c, and $R' = R_{K^t}^{\leq \lceil (c+1)\log n \rceil}$, then $\forall x \in \{0,1\}^n$ $M^{R'}(x) = A(x)$.*

Proof. Let $\ell(n) = \lceil (c+1)\log n \rceil$, $R'(n) = R_{K^t}^{\leq \ell(n)}$, and suppose that $M^{R'(n)}(x) \neq A(x)$ for some x of length n. Then we may find the first such x in time $2^{\ell(n)} t(\ell(n)) + 2^{n+1}(t_1(n) + O(n^c))$ (cf. Proposition 3.2.5), and each query made by $M^{R'(n)}(x)$ can be output by a program of length $c \log n + O(1)$, running in the same time bound. But since $A(x) \neq M^{R'(n)}(x)$, it must be that, with $R'(n)$ as oracle, M makes some query q of size $m \geq \ell(n) + 1$ which is random for t-bounded Kolmogorov complexity (because both small and nonrandom queries are answered correctly when using R' instead of R_{K^t}). Hence we have both that q is supposed to be random, and that q can

[1] *For example, if $A \in \mathrm{EXP}$, then t can be doubly-exponential. If A is elementary-time computable, then t can be an exponential tower.*

be output by a program of length $< \ell(n)$ in time $2^{\ell(n)}t(\ell(n)) + 2^{n+1}(t_1(n) + O(n^c)) \ll 2^{\ell(n)}t(\ell(n)) + 2^{2^{\ell(n)}}t_1(2^{\ell(n)}) \leq t(\ell(n)+1) \leq t(m)$, which is a contradiction. □

Corollary 3.3.3 ([ABFL12a]). $\mathsf{TTRT} \subseteq \mathrm{P/poly}$.

Proof. Let $L \in \mathsf{TTRT}$. By the definition of TTRT, $L \leq^p_{tt} R_{K^{t_0}}$ for some t_0. Using Proposition 3.2.6, we then have that L is decidable in time $t_L(n) = 2^{n^k}t_0(n^k)$ for some constant k. Choose a time-bound t such that $t(n+1) \geq 2^n t(n) + 2^{2^n}t_L(2^n)$. By the definition of TTRT, since $t > t_0$, we have that $L \leq^p_{tt} R_{K^t_{U_0}}$, from which by Theorem 3.3.2 it follows that $L \in \mathrm{P/poly}$. □

PSPACE $\leq^p_T R_K$ [ABK+06b], but Theorem 3.3.2 implies that PSPACE $\not\leq^p_T R_{K^t}$ for sufficiently-large t, unless PSPACE $\subseteq$ P/poly. This highlights the difference between the time-bounded and ordinary Kolmogorov complexity, and how this comes to the surface when working with reductions to the corresponding sets of random strings. We wish to emphasize at this point that the proof of the inclusion PSPACE $\leq^p_T R_K$ relies on the ability of a P^{R_K} computation to construct a large element of R_K, whereas the P/poly upper bound in the time-bounded case relies on the *inability* to use the oracle to find such a string, in the time-bounded setting.

3.3.2 A Reduction Distinguishing R_K from R_{K^t} and an Incorrect Conjecture

Theorem 3.3.2 shows that a polynomial-time truth-table reduction to R_{K^t} for sufficiently-large t will work just as well if only the logarithmically-short queries are answered correctly, and all of the other queries are simply answered NO.

The authors of [ADF+12] conjectured that a similar situation would hold if the oracle were R_K instead of R_{K^t}. More precisely, they proposed a proof-theoretic approach towards proving that DTTR is in P/poly: Let PA_0 denote Peano Arithmetic, and for $k > 0$ let PA_k denote PA_{k-1} augmented with the axiom "PA_{k-1} is consistent". In [ADF+12] it is shown that, for any polynomial-time truth-table reduction M reducing a decidable set A to R_K, one can construct a true statement of the form $\forall n \forall j \forall k \Psi(n, j, k)$

(which is provable in a theory such as Zermelo-Frankel), with the property that if, for each fixed (**n**,**j**,**k**) there is some k' such that PA$_{k'}$ proves ψ(**n**,**j**,**k**), then $\mathsf{DTTR} \subseteq \mathrm{P/poly}$. Furthermore, if these statements were provable in the given extensions of PA, it would follow that, for each input length n, there is a finite subset $R' \subseteq R_K$ consisting of strings having length at most $O(\log n)$, such that $M^{R'}(x) = A(x)$ for all strings x of length n.

Thus the authors of [ADF$^+$12] implicitly conjectured that, for any polynomial-time truth-table reduction of a decidable set to R_K, and for any n, there would be some setting of the short queries so that the reduction would still work on inputs of length n, when all of the long queries are answered NO. While we have just seen that this is precisely the case for the time-bounded situation, the next theorem shows that this does not hold for R_K, even if "short" is interpreted as meaning "of length $< n$". (It follows that infinitely many of the statements ψ(**n**,**j**,**k**) of [ADF$^+$12] are independent of every PA$_{k'}$.)

Theorem 3.3.4 ([ABFL12a]). *There is a truth-table reduction $M : \{0,1\}^* \leq^p_{tt} R_K$, such that, for all large enough n:*

$$\forall R' \subseteq \{0,1\}^{\leq n-1} \exists x \in \{0,1\}^n \; M^{R'}(x) \neq 1.$$

Proof. Theorem 15 of [BFNV05] presents a polynomial-time procedure which, given a string z of even length $n-2$, will output a list of constantly-many strings $z_1, \ldots, z_c$ of length n, such that at least one of them will be K-random if z is. We use this to define our reduction M as follows: on input $x = 00\ldots0z$ of length n having even $|z|$, we query each of z, $z_1, \ldots, z_c$, and every string of length at most $\log n$. If there are no strings of length at most $\log n$ in the oracle, we reject. Else, if z is in the oracle but none of the z_i are, we reject. On all other cases we accept.

By [BFNV05, Theorem 15], and since R_K has strings at every length, it is clear that M accepts every string with oracle R_K, and rejects every string if $R' = \varnothing$. However, for any non-empty set $R' \subseteq \{0,1\}^{\leq n-1}$, let $\ell \leq n-1$ be the highest even length for which $R'^{=\ell} \neq \varnothing$, and pick $z \in R'^{=\ell}$. Then we will have $z \in R'^{=\ell}$ but every $z_i \notin R^{=\ell+2}$, hence $M^{R'}(00\ldots0z)$ rejects. $\square$

In fact, if we let $R' = R_{K^t}^{\leq n-1}$, for even n, then for the first $x = 00z$ such that $M^{R'}(x) = 0$, we will have $z \in R' \subseteq R_{K^t}$, but each z_i can be given by a small pointer in time $O(2^{n-1}t(n-1))$ (again we use Proposition 3.2.5), and hence $z_i \notin R_{K^t}$ for suitably fast-growing t. Thus $M^{R_{K^t}}(x) = 0 \neq M^{R_K}(x)$, and we conclude:

Observation 3.3.5. *If $t(n+1) \gg 2^n t(n)$, then the non-adaptive reduction M above behaves differently on the oracles R_K and R_{K^t}.*

3.4 Polynomial Space with Advice

Our single goal for this section is proving the following:

Theorem 3.4.1 ([ABFL12a])**.** *For any computable unbounded function $\alpha(n) = \omega(1)$,*

$$\mathsf{TTRT} \subseteq \text{PSPACE}/\alpha(n).$$

The proof of this theorem is patterned closely on that of Theorem 2.2.6, although a number of complications arise in the time-bounded case. There is enough new material that for completeness we include the entire proof here, despite the fact that there is some overlap. One can refer to the paper [ABFL12b] for a presentation that is not self-contained but emphasizes the differences between the proof in the time-bounded case and the unbounded case. Before proving the theorem we present several supporting propositions.

Proposition 3.4.2. *For any time bound t and time-efficient universal prefix machine U,*

$$\sum_{x\in\{0,1\}} 2^{-K_U^t(x)} \leq 1.$$

Proof. From the Kraft Inequality (see e.g. [LV08], Theorem 1.11.1), $\sum_{x\in\{0,1\}} 2^{-K_U(x)} \leq 1$ for any prefix machine U. For any time bound t and string x, $K_U^t(x) \geq K_U(x)$, so adding a time bound can only decrease the sum on the left side of this inequality. □

Proposition 3.4.3. *Let f be a function such that*

1. $\sum_{x\in\{0,1\}^*} 2^{-f(x)} \leq 1$, *and*

2. there is a machine M computing $f(x)$ in time $t(|x|)$.

Let $t'(|x|) > 2^{2|x|}t(|x|)$. Then for some M', $K^{t'}_{M'}(x) = f(x) + 2$.

Proof. The proof is similar to the proof of Proposition 2.2.3. Let

$$E = \langle x_0, f(x_0)\rangle, \langle x_1, f(x_1)\rangle, \ldots$$

be an enumeration of the function f ordered lexicographically by the strings x_i.

We identify the set of infinite sequences $S = \{0,1\}^\infty$ with the half-open real interval $[0,1)$; that is, each real number r between 0 and 1 will be associated with the sequence(s) corresponding to the infinite binary expansion of r. We will associate each element $\langle x_i, f(x_i)\rangle$ from the enumeration E with a subinterval $I_i \subseteq S$ as follows:

$I_0 = [0, 2^{-f(x_0)})$, and for $i \geq 1$, $I_i = [\sum_{k<i} 2^{-f(x_k)}, \sum_{k\leq i} 2^{-f(x_k)})$. That is, I_i is the half-open interval of length $2^{-f(x_i)}$ that occurs immediately after the interval corresponding to the element $\langle x_{i-1}, f(x_{i-1})\rangle$ that appeared just prior to $\langle x_i, f(x_i)\rangle$ in the enumeration E.

Since $\sum_{i\geq 0} 2^{-f(x_i)} \leq 1$, each $I_i \subseteq S$.

Any *finite* string z also corresponds to a subinterval $\Gamma_z \subseteq S$ consisting of all infinite sequences that begin with z; Γ_z has length $2^{-|z|}$. Given any element $\langle x_i, f(x_i)\rangle$, there must exist a lexicographically first string z_i of length $f(x_i) + 2$ such that $\Gamma_{z_i} \subseteq I_i$. Observe that, since the intervals I_i are disjoint, no string z_i is a prefix of any other.

Let M' be the following machine. On input z, M' runs M to compute the enumeration E until it finds an element $\langle x_i, f(x_i)\rangle$ that certifies that $z = z_i$. If it finds such an element then M' outputs x_i.

Suppose that M' outputs x_i on input z, and let $\langle x_i, f(x_i)\rangle$ be the element of E corresponding to x_i. Before outputting x_i, M' must compute $|\langle x_j, f(x_j)\rangle|$ for every string x_j such that $x_j < x_i$ (under the lexicographical ordering). There are at most $2^{|x_i|+1}$ strings x_j such that $x_j < x_i$, so overall this will take less than $2^{2|x_i|}t(|x_i|)$ time.

M' will be a prefix machine, and we have that $K^{t'}_{M'}(x) = f(x) + 2$. □

Proposition 3.4.4 (Analogue to Proposition 2.2.4). *Let U be a time-efficient universal prefix Turing machine and let M be any prefix Turing machine. Suppose that t, t', and*

t'' are time bounds and f, g are two time-constructible increasing functions, such that f is upper bounded by a polynomial, and $t''(|x|) \geq \max\{f(t(|x|)), g(t'(|x|))\}$.

Then there is a time-efficient universal prefix machine U' such that

$$K_{U'}^{t''}(x) = \min(K_U^t(x), K_M^{t'}(x)) + 1.$$

Proof. On input $0y$, U' runs U on input y. If U would output string x on y after s steps, then U' outputs string x after $f(s)$ steps. Similarly, on input $1y$, U' runs M on input y. If M would output string x on y after s steps, then U' outputs string x after $g(s)$ steps.

Note that because U is an efficient universal prefix machine, U' will be an efficient universal prefix machine as well. □

Proposition 3.4.5 (Analogue of Proposition 2.2.5)**.** *Given any time-efficient universal prefix machine U, time bound t, and constant $c \geq 0$, there is a time-efficient universal prefix machine U' such that $K_{U'}^t(x) = K_U^t(x) + c$.*

Proof. On input $0^c x$, M' runs M on input x, and doesn't halt on other inputs. □

Proof of Theorem 3.4.1. Fix α, and suppose for contradiction that $L \in \mathsf{TTRT}-\text{PSPACE}/\alpha(n)$. Let t_0 be the time bound given in the definition of TTRT, and assume without loss of generality that $t_0(n)$ is greater than the time required to compute the length of the advice $\alpha(n)$, and let U_0 be some arbitrary time-efficient universal prefix machine. By the definition of TTRT, $L \leq_{tt}^p R_{K_{U_0}^{t_0}}$. Therefore, by Proposition 3.2.6, L is decidable in time $t_L(n) = 2^{n^k} t_0(n^k)$ for some constant k.

Let $t^*(n)$ be an extremely fast-growing time-constructible function, so that for any constant d, we have $t^*(\log(\alpha(n))) > 2^{n^d} t_L(n)$ for all large n. To get our contradiction, we will show that there exists a time-efficient universal prefix machine U such that $L \not\leq_{tt}^p R_{K_U^{t^{*3}}}$. Note that because $t^* > t_0$, this is a contradiction to the fact that $L \in \mathsf{TTRT}$.

For any function $f : \{0,1\}^* \to \mathbb{N}$, define $R_f = \{x : f(x) \geq |x|\}$. We will construct a function $F : \{0,1\}^* \to \mathbb{N}$ and use it to form a function $H : \{0,1\}^* \to \mathbb{N}$ such that:

1. F is a total function and $F(x)$ is computable in time $t^{*2}(|x|)$ by a machine M;

2. $H(x) = \min(K^{t^*}_{U_0}(x) + 5, F(x) + 3)$;

3. $\sum_{x\in\{0,1\}^*} 2^{-H(x)} \leq 1/8$;

4. $L \not\leq^p_{tt} R_H$.

$\square$

Claim 3.4.6 (Analogue of Claim 2.2.7)**.** *Given the above properties $H = K^{t^{*3}}_U$ for some efficient universal prefix machine U.*

By Property 4 this ensures that the theorem holds.

Proof. By Property 3 we have that $\sum_{x\in\{0,1\}^*} 2^{-(F(x)+3)} \leq 1/8$. Hence $\sum_{x\in\{0,1\}^*} 2^{-F(x)} \leq 1$. Using this along with Property 1, we then have by Proposition 3.4.3 that $K^{t^{*3}}_{M'} = F+2$ for some prefix machine M'. By Proposition 3.4.5 we have that $K^{t^*}_{U'} = K^{t^*}_{U_0} + 4$ for some efficient universal prefix machine U'. Therefore, by Proposition 3.4.4, with $f(n) = n^3, g(n) = n$, we find that $H(x) = \min(K^{t^*}_{U_0}(x) + 5, F(x) + 3) = \min(K^{t^{*3}}_{M'}, K^{t^*}_{U'}(x)) + 1$ is $K^{t^{*3}}_U$ for some efficient universal prefix machine U. $\square$

All we now need to show is that, for our given language L, we can always construct functions H and F with the four desired properties.

Let $\gamma_1, \gamma_2, \ldots$ be a list of all possible polynomial-time truth-table reductions from L to R_H. This is formed in the usual way: we take a list of all Turing machines and put a clock of $n^i + i$ on the ith one and we will interpret the output on a string x as an encoding of a Boolean circuit on atoms of the form "$z \in R_H$". (i.e. these atoms form the input gates of the circuit, and their truth values determine the output of the circuit.) We will refer to the string z as a *query*.

As in the proof of Theorem 2.2.6, to ensure that $L \not\leq^p_{tt} R_H$ (Property 4), we need to satisfy an infinite list of requirements of the form

R_e : γ_e is not a polynomial-time truth-table reduction of L to R_H.

As part of our construction we will set up and play a number of games, which will enable us to satisfy each of these requirements R_e in turn. Our moves in the game will define the function F (and thus indirectly H). Originally we have that $F(z) = 2|z| + 3$ for all strings z. Potentially during one of these games, we will play a move forcing a string z to be in the complement of R_H. To do this we will set $F(z) = |z| - 4$. Therefore, a machine M can compute $F(z)$ by running our construction, looking for the first time during the construction that $F(z)$ is set to $|z| - 4$, and outputting $|z| - 4$. If a certain amount of time elapses (to be determined later) during the construction without $F(z)$ ever being set to $|z| - 4$, then the machine M outputs the default value $2|z| + 3$.

3.4.1 Description of the Games

Let us first describe abstractly the games that will be played during the construction; afterwards we will explain how it is that we use these games to satisfy each requirement R_e. (Note that these games are defined differently than those in the proof of Theorem 2.2.6).

For a given requirement R_e, a game $\mathcal{G}_{e,x}$ will be played as followed for some string x:

First we calculate the circuit $\gamma_{e,x}$, which is the output of the reduction γ_e on input x. Let F^* be the function F as it is at this point of the construction when the game $\mathcal{G}_{e,x}$ is about to be played. For any atom "$z_i \in R_H$" that is an input of this circuit such that $|z_i| \leq \log(\alpha(|x|)) - 1$, we calculate $r_i = \min(K^{t^*}_{U_0}(z_i) + 5, F^*(z_i) + 3)$. If $r_i < |z_i|$ we substitute FALSE in for the atom, and simplify the circuit accordingly, otherwise we substitute TRUE in for the query, and simplify the circuit accordingly. (We will refer to this as the "pregame preprocessing phase".)

The remaining queries z_i are then ordered by increasing length. There are two players, the F player (whose moves will be played by us during the construction), and the K player (whose moves will be determined by $K^{t^*}_{U_0}$). As in the proof of Theorem 2.2.6, in each game the F player will either be playing on the YES side (trying to make the final value of the circuit equal TRUE), or the NO side (trying to make the final value of the circuit equal FALSE).

Let S_1 be the set of queries from $\gamma_{e,x}$ of smallest length, let S_2 be the set of queries that have the second smallest length, etc. So we can think of the queries being partitioned into an ordered set $\mathcal{S} = (S_1, S_2, \ldots, S_r)$ for some r.

The scoring for the game is similar to that in the proof of Theorem 2.2.6; originally each player has a score of 0 and a player loses if his score exceeds some threshold ϵ. When playing a game $\mathcal{G}_{e,x}$, we set $\epsilon = 2^{-e-3}$.

Originally we have that the truth value of all the atoms in the game are TRUE. In round one of the game, the K player makes some (potentially empty) subset Z_1 of the queries from S_1 nonrandom; i.e. for each $z \in Z_1$ he sets the atom "$z \in R_H$" to the value FALSE. For any $Z_1 \subseteq S_1$ that he chooses to make nonrandom, $\sum_{z \in Z_1}(2^{-(|z|-6)} - 2^{-(2|z|+3)})$ is added to his score. As in the proof of Theorem 2.2.6, a player can only legally make a move if doing so will not cause his score to exceed ϵ.

After the K player makes his move in round 1, the F player responds, by making some subset Y_1 of the queries from $S_1 - Z_1$ nonrandom. After the F player moves, $\sum_{z \in Y_1} 2^{-(|z|-4)} - 2^{-(2|z|+3)}$ is added to his score.

This is the end of round one. Then we continue on to round two, played in the same way. The K player goes first and makes some subset of the queries from S_2 nonrandom (which makes his score go up accordingly), and then the F player responds by making some subset of the remaining queries from S_2 nonrandom. Note that if a query from S_i is not made nonrandom by either the K player or the F player in round i, it cannot be made nonrandom by either player for the remainder of the game.

After r rounds are finished the game is done and we see who wins, by evaluating the circuit $\gamma_{e,x}$ using the answers to the queries that have been established by the play of the game. If the circuit evaluates to TRUE (FALSE) and the F player is playing as the YES (NO) player, then the F player wins, otherwise the K player wins.

Note that the game is asymmetric between the F player and the K player; the F player has an advantage due to the fact that he plays second in each round and can make an identical move for fewer points than the K player. Because the game is asymmetric, it is possible that F can have a winning strategy playing on *both* the YES and NO sides. Thus we define a set $val(\mathcal{G}_{e,x'}) \subseteq \{0,1\}$ as follows: $0 \in val(\mathcal{G}_{e,x'})$ if the F player has a winning strategy playing on the NO side in $\mathcal{G}_{e,x'}$, and $1 \in val(\mathcal{G}_{e,x'})$ if the F player has a winning strategy playing on the YES side in $\mathcal{G}_{e,x'}$.

3.4.2 Description of the Construction

Now we describe the construction. In contrast to the situation in the proof of Theorem 2.2.6, we do not need to worry about playing different games simultaneously or dealing with requirements in an unpredictable order; we will first satisfy R_1, then R_2, etc. To satisfy R_e we will set up a game $\mathcal{G}_{e,x}$ for an appropriate string x of our choice, and then play out the game in its entirety as the F player. We will choose x so that we can win the game $\mathcal{G}_{e,x}$, and will arrange that by winning the game we ensure that R_e is satisfied. It is possible that the K player will "cheat" on game $\mathcal{G}_{e,x}$, if our interpretation of the function $K_{U_0}^{t^*}$, which will determine the moves of the K player, does not translate into legal moves in the game. In this case we quit the game $\mathcal{G}_{e,x}$ and we play $\mathcal{G}_{e,x'}$ for some new x'. However, we will show that the K player cannot cheat infinitely often on games for a particular e, so eventually R_e will be satisfied.

Originally we define the function F so that $F(z) = 2|z|+3$ for all strings z. Suppose s time steps have elapsed during the construction up to this point, and we are getting ready to construct a new game in order to satisfy requirement R_e. (Either because we just finished satisfying requirement R_{e-1}, or because K cheated on some game $\mathcal{G}_{e,x}$, so we have to start a new game $\mathcal{G}_{e,x'}$). Starting with the string $0^{t^{*4}(s)}$ (i.e. the string of $t^{*4}(s)$ zeros), we search strings in lexicographical order until we find an x' such that $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. (Here, L denotes the characteristic function of the set L.)

Once we find such a string x' (which we will prove we always can), then we play out the game $\mathcal{G}_{e,x'}$ with the F player (us) playing on the YES side if $L(x') = 0$ and the NO side if $L(x') = 1$. To determine the K player's move in the ith round, we let $Z_i \subseteq S_i$ be those queries $z \in S_i$ for which $K_{U_0}^{t^*}(z) \leq |z| - 6$. Our moves are determined by our winning strategy; whenever we play a move that makes a query z nonrandom, we update the function F so that $F(z) = |z| - 4$. Note that whenever either of the player plays a move involving a query z in one of the games (which we have called "making z nonrandom"), he *does* make the query z nonrandom in the sense that $R_H(z)$ is fixed to the value 0 for good.

To finish showing that Property 4 will be satisfied, it suffices to prove the following three claims.

Claim 3.4.7. *If during the construction we win a game $\mathcal{G}_{e,x}$, then R_e will be satisfied and will stay satisfied for the remainder of the construction.*

Proof. Suppose that we win a game $\mathcal{G}_{e,x}$. Let $H^* = \min(Kt^*_{U_0} + 5, F^* + 3)$, where F^* is the function F immediately after the game $\mathcal{G}_{e,x}$ is completed. Our having won the game implies that when evaluating the circuit $\gamma_{e,x}$, while substituting the truth value of "$z \in R_{H^*}$" for any query of the form "$z \in R_H$", we have that $\gamma_{e,x} \neq L(x)$, which means that the reduction γ_e does not output the correct value on input x and thus R_e is satisfied. For any game $\mathcal{G}_{e',x'}$ that is played later in the construction, by design x' is always chosen large enough so that any query that is not fixed during the pregame preprocessing has not appeared in any game that was played previously, so $\mathcal{G}_{e',x'}$ will not conflict with $\mathcal{G}_{e,x}$ and R_e will remain satisfied for the remainder of the construction. □

Claim 3.4.8. *For any given requirement R_e, the K player will only cheat on games $R_{e,x}$ for a finite number of strings x.*

Proof. If the K player cheats on a game $R_{e,x}$, it means that he makes moves that causes his score to exceed $\epsilon = 2^{-e-3}$. By the definition of how K's moves are determined, this implies that $\sum_{z\in Z_{e,x}} 2^{-(Kt^*_{U_0}(z)-6)} \geq \epsilon$, so $2^{-Kt^*_{U_0}(z)} \geq \epsilon/64$, where $Z_{e,x}$ is defined to be the set of all the queries that appear in the game $\mathcal{G}_{e,x}$ that are not fixed during the preprocessing stage. However, for any two games $G_{e,x}$ and $G_{e,x'}$ the sets $Z_{e,x}$ and $Z_{e,x'}$ are disjoint, so if K cheated on an infinite number of games associated with the requirement R_e, then this would imply that $\sum_{z\in\{0,1\}^*} 2^{Kt^*_{U_0}(z)} \geq \epsilon/64 + \epsilon/64 + \cdots$. But this divergence would violate Lemma 3.4.2. □

Claim 3.4.9. *During the construction, for any requirement R_e, we can always find a witness x with the needed properties to construct $\mathcal{G}_{e,x}$.*

Proof. Suppose for some requirement R_e, our lexicographical search goes on forever without finding an x such that $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. Then $L \in \text{PSPACE}/\alpha(n)$, which is a contradiction.

Here is the PSPACE algorithm to decide L. Hardcode all the answers for the initial sequence of strings up to the point where we got stuck in the construction. Let F^* be the function F up to that point in the construction. On a general input x, construct $\gamma_{e,x}$. The advice function $\alpha(n)$ will give the truth-table of $\min(K_{U_0}^{t^*}(z)+5, F^*(z)+3)$ for all queries z such that $|z| \leq \log(\alpha(|x|))-1$. For any query z of $\gamma_{e,x}$ such that $|z| \leq \log(\alpha(|x|))-1$, fix the answer to the query according to the advice.

If the F player had a winning strategy for both the YES and NO player on game $\mathcal{G}_{e,x}$, then we wouldn't have gotten stuck on R_e. Also the F player must have a winning strategy for either the YES or the NO player, since he always has an advantage over the K player when playing the game. Therefore, because we got stuck, it must be that the F player has a winning strategy for the YES player if and only if $L(x)=1$. Once the small queries have been fixed, finding the side (YES or NO) for which the F player has a winning strategy on $\mathcal{G}_{e,x}$, and hence whether $L(x)=1$ or $L(x)=0$, can be done in PSPACE.

To prove this, we will show that the predicate "The F player has a winning strategy as the YES player on $\mathcal{G}_{e,x}$" can be computed in alternating polynomial time, which by [CKS81] is equal to PSPACE. To compute this predicate, we must determine if *for every* move of the K player in round 1, there *exists* a move for the F player in round 1, such that *for every* move of the K player in round 2, there *exists* a move for the F player in round 2... such that when the game is finished the circuit $\gamma_{e,x}$ evaluates to TRUE. We can represent any state of the game (i.e. which of the polynomial number of queries have been fixed to be nonrandom so far, the score of the players, the current round, and whose turn it is) by a number of bits bounded by a polynomial in $|x|$. Also, given a move by one of the players, it is easy to determine in polynomial time whether the move is legal and to compute the new score of the player after the move. (It suffices to add up a polynomial number of rationals of the form $a/2^b$ where $b=n^{O(1)}$). Also, because there are only a polynomial number of queries in the circuit $\gamma_{e,x}$, the total number of moves in the game is bounded by a polynomial. Finally, evaluating the circuit at the end of the game can be done in polynomial time. Thus the predicate in question can be computed in alternating polynomial time, which completes the proof. $\square$

The following claim shows that Property 1 is satisfied.

Claim 3.4.10. *$F(z)$ is computable in time ${t^*}^2(|z|)$.*

Proof. The function F is determined by the moves we play in games during the construction. In order to prove the claim, we must show that if during the construction we as the F player make a move that involves setting a string z to be nonrandom, then fewer than ${t^*}^2(|z|)$ time steps have elapsed during the construction up to that point. The machine M that computes F will on input z run the construction for ${t^*}^2(|z|)$ steps. If, at some point before this during the construction, we as the F player make z nonrandom, then M outputs $|z| - 4$. Otherwise M outputs $2|z| + 3$.

Suppose during the construction that we as the F player make a move that sets a query z to be nonrandom during a game $\mathcal{G}_{e,x}$. Note that $|z| \geq \log(\alpha(|x|))$, otherwise z would have been fixed during the preprocessing stage of the game.

There are at most $2^{|x|+1}$ strings x' that we could have considered during our lexicographic search to find a game for which we had a winning strategy before finally finding x. Let s be the number of time steps that have elapsed during the construction before this search began.

Let us first bound the amount of time it takes to reject each of these strings x'. To compute the circuit $\gamma_{e,x'}$ takes at most $|x'|^k$ time for some constant k. For each query y such that $|y| \leq \log(\alpha(|x'|)) - 1$ we compute $\min(K^{t^*}_{U_0}(y) + 5, F^*(y) + 3)$. To calculate $F^*(y)$ it suffices to rerun the construction up to this point and check whether a move had been previously made on the string y. To do this takes s time steps, and by construction we have that $t^*(|z|) \geq t^*(\log \alpha(|x|)) > |x| \geq |x'| \geq {t^*}^4(s)$, so $s < |z|$. By Proposition 3.2.5, to compute $K^{t^*}_{U_0}(y)$ takes at most $2^{|y|}t^*(|y|) \leq 2^{|z|}t^*(|z|)$ time steps. Therefore, since there can be at most $|x'|^k$ such queries, altogether computing $\min(K^{t^*}_{U_0}(y) + 5, F^*(y) + 3)$ for all these y will take fewer than $|x'|^k 2^{|z|} t^*(|z|)$ time steps.

Then we must compute $L(x')$, and check whether $(1 - L(x')) \in val(\mathcal{G}_{e,x'})$. Computing $L(x')$ takes $t_L(|x'|)$ time. By Claim 3.4.9, once the small queries have been fixed appropriately, computing $val(\mathcal{G}_{e,x'})$ can be done in PSPACE, so it takes at most $2^{|x'|^d}$ time for some constant d.

Compiling all this information, and using the fact that for each of these x' we have that $|x'| \leq |x|$, we get that the total number of timesteps needed to reject all of these x' is less than $2^{|x|^{d'}} 2^{|z|} t_L(|x|) t^*(|z|)$ for some constant d'.

During the actual game $\mathcal{G}_{e,x}$, before z is made nonrandom the construction might have to compute $K^{t^*}_{U_0}(y) + 5$ for all queries of $\gamma_{e,x}$ for which $|y| \leq |z|$. By Proposition 3.2.5 this takes at most $|x|^k 2^{|z|} t^*(|z|)$ time.

Therefore, overall, for some constant d'' the total amount of time steps elapsed before z is made nonrandom in the construction is at most

$$T = 2^{|x|^{d''}} 2^{|z|} t_L(|x|) t^*(|z|) + s < t^{*2}(|z|).$$

Here the inequality follows from the fact that $t^*(\log(\alpha(|x|))) > 2^{|x|^d} t_L(|x|)$ for any constant d, and that $|z| \geq \log(\alpha(|x|))$.

$\square$

Finally, to finish the proof of the theorem we need to show that Property 3 is satisfied.

Claim 3.4.11. $\sum_{x \in \{0,1\}^*} 2^{-H(x)} \leq \frac{1}{8}$.

Proof. To begin, notice that

$$\sum_{x \in \{0,1\}^*} 2^{-H(x)} = \sum_{x \in \{0,1\}^*} 2^{-\min(K^{t^*}_{U_0}(x)+5, F(x)+3)} \leq \sum_{x \in \{0,1\}^*} 2^{-(K^{t^*}_{U_0}(x)+5)} + \sum_{x \in \{0,1\}^*} 2^{-(F(x)+3)}.$$

By Proposition 3.4.2, $\sum_{x \in \{0,1\}^*} 2^{-K^{t^*}_{U_0}(x)} \leq 1$, so $\sum_{x \in \{0,1\}^*} 2^{-(K^{t^*}_{U_0}(x)+5)} \leq 1/32$. We also have that $\sum_{x \in \{0,1\}^*} 2^{-(F(x)+3)} = (1/8) \sum_{x \in \{0,1\}^*} 2^{-F(x)}$. Therefore, it is enough that $\sum_{x \in \{0,1\}^*} 2^{-F(x)} \leq 1/2$, as this would imply that

$$\sum_{x \in \{0,1\}^*} 2^{-H(x)} \leq \frac{1}{32} + \frac{1}{8} \times \frac{1}{2} \leq \frac{1}{8}.$$

Let Z_F be the set of all those queries that we (the F player) make nonrandom during the construction by playing a move in one of the games. We have that

$$\begin{aligned}\sum_{x\in\{0,1\}^*} 2^{-F(x)} &= \sum_{x\in Z_F} 2^{-(|x|-4)} + \sum_{x\notin Z_F} 2^{-(2|x|+3)} \\ &= \sum_{x\in\{0,1\}^*} 2^{-(2|x|+3)} + \sum_{x\in Z_F} (2^{-(|x|-4)} - 2^{-(2|x|+3)}) \\ &\leq \frac{1}{8} + \sum_{x\in Z_F} (2^{-(|x|-4)} - 2^{(2|x|+3)}).\end{aligned}$$

Thus it now suffices to show that $tot_F = \sum_{x\in Z_F}(2^{-(|x|-4)} - 2^{(2|x|+3)}) \leq 1/4$. Notice that tot_F is exactly the total number of points that the F player accrues in all games throughout the lifetime of the construction. First let us consider those games on which the K player cheats. We know that in all these games, the F player accrues fewer points than the K player, and in particular accrues fewer points during these games than tot_K, the total number of points the K player accrues in all games throughout the lifetime of the construction. Let Z_K be the set of all those queries that the K player makes nonrandom during the construction by playing a move in one of the games. We have that

$$tot_K = \sum_{z\in Z_K} 2^{-(|z|-6)} - 2^{-(2|z|+3)} \leq \sum_{z\in Z_K} 2^{-(K^{t^*}_{U_0}(z)+5)} \leq \sum_{z\in\{0,1\}^*} 2^{-(K^{t^*}_{U_0}(z)+5)} \leq \frac{1}{32},$$

where the first inequality uses that for all $z \in Z_K$, $K^{t^*}_{U_0}(z) \leq |z| - 6$, and the last inequality again comes from Proposition 3.4.2.

Now consider games on which K does not cheat – for each R_e there will be exactly one of these. On each of these games the F player can accrue at most $\epsilon = 2^{-e-3}$ points. Thus the total number of points the F player accrues on all games that K does not cheat on is at most $\sum_{e=1}^{\infty} 2^{-e-3} = 1/8$.

Therefore $tot_F \leq 1/32 + 1/8 \leq 1/4$.

□

3.5 Removing the Advice

With the plain Kolmogorov complexity function C, it is fairly clear what is meant by a "time-efficient" universal Turing machine. Namely, U is a time-efficient universal

Turing machine if, for every Turing machine M, there is a constant c so that, for every x, if there is a description d for which $M(d) = x$ in t steps, then there is a description d' of length $\leq |d| + c$ for which $U(d') = x$ in at most $ct \log t$ steps. However, with prefix-free Kolmogorov complexity, the situation is more complicated. The easiest way to define universal Turing machines for the prefix-free Kolmogorov complexity function K is in terms of *self-delimiting Turing machines.* These are machines that have one-way access to their input tape; x is a valid input for such a machine if the machine halts while scanning the last symbol of x. For such machines, the notion of time-efficiency carries over essentially unchanged. However, there are several other ways of characterizing K (such as in terms of partial-recursive functions whose domains form a prefix code, or in terms of prefix-free entropy functions). The running times of the machines that give short descriptions of x using some of these other conventions can be substantially less than the running times of the corresponding self-delimiting Turing machines. This issue has been explored in detail by Juedes and Lutz [JL00], in connection with the P versus NP problem. Given that there is some uncertainty about how best to define the notion of time-efficient universal Turing machine for K^t-complexity, one possible response is simply to allow much more leeway in the time-efficiency requirement.

If we do this, we are able to get rid of the small amount of non-uniformity in our PSPACE upper bound.

Definition 3.5.1. *A prefix machine U is an f-efficient universal prefix machine if there exist constants c_M for each prefix machine M, such that*

1. *$\forall x$, $K_U(x) \leq K_M(x) + c_M$; and*
2. *$\forall x$, $K_U^t(x) \leq K_M^{t'}(x) + c_M$ for all $t(n) > f(t'(n))$.*

In Definition 3.2.2 we defined a time-efficient universal prefix machine to be any poly(n)-efficient universal prefix machine.

Definition 3.5.2. *Define TTRT' to be the class of languages L such that for all computable f there exists t_0 such that for all f-efficient universal prefix machines U and $t \geq t_0$, $L \leq_{tt}^{p} R_{K_U^t}$.*

Theorem 3.5.3 ([ABFL12a]). BPP $\subseteq \mathsf{TTRT}' \subseteq$ PSPACE $\cap$ P/poly.

Note that $\mathsf{TTRT}' \subseteq \mathsf{TTRT}$, so from Theorem 3.3.2 we get $\mathsf{TTRT}' \subseteq$ P/poly. Also, the proofs in [BFKL10] can be adapted to show that BPP $\subseteq \mathsf{TTRT}'$. So all we need to show is the PSPACE inclusion.

Proof of Theorem 3.5.3. The proof is similar to the proof of Theorem 3.4.1, with some minor technical modifications. Let L be an arbitrary language from $\mathsf{TTRT}' -$ PSPACE. Because $\mathsf{TTRT}' \subseteq \mathsf{TTRT}$, as in the proof of Theorem 3.4.1 we have that L is decidable in time $t_L < 2^{n^k} t'(n^k)$ for some fixed time bound t' and constant k.

Define f to be a fast enough growing function that $f(n) > 2^{(t_L(n^d))^d}$ for any constant d, for all large n. By the definition of TTRT', for this f there exists a t_0 such that for all $t \geq t_0$, $L \leq_{tt}^{p} R_{K_U^t}$. Let $t^*(n)$ be a time bound such that for all n, $t^*(n) > f(n)$ and $t^*(n) > t_0(n)$. To get our contradiction, we will show that there exists an f-efficient universal prefix machine U and constant $c > 1$ such that $L \not\leq_{tt}^{p} R_{K_U^v}$, where $v(|x|) = 2^{(t_L(t^*(|x|)))^c} > t_0(|x|)$.

We will make use of the following revised proposition:

Proposition 3.5.4 (Revised Proposition 3.4.4). *Let U and M be an n^c-efficient universal prefix Turing machine and a prefix Turing machine respectively. Let t, t' be time bounds and f, g be two time-constructible increasing functions, such that $g(n^c) < f(n)$. Let $t''(|x|) = g(t(|x|)) = h(t'(|x|))$. Then there is an f-efficient universal prefix machine U' such that*

$$K_{U'}^{t''}(x) = \min(K_U^t(x), K_M^{t'}(x)) + 1.$$

Proof. Almost identical to before: On input $0y$, U' runs U on input y. If U would output string x on y after s steps, then U' outputs string x after $g(s)$ steps. Similarly, on input $1y$, U' runs M on input y. If M would output string x on y after s steps, then U' outputs string x after $h(s)$ steps.

Note that because U is an n^c-efficient universal prefix machine, U' will be an f-efficient universal prefix machine. $\square$

We will construct functions F and H such that

1. F is a total function such that for all x, $F(x) \leq 2|x| + 3$, and $F(x)$ is computable in time $2^{(t_L(t^*(|x|)))^d}$ by a machine M for some constant d.

2. $H(x) = \min(K^{t^*}_{U_0} + 5, F(x) + 3)$.

3. $\sum_{x \in \{0,1\}^*} 2^{-H(x)} \leq 1/8$

4. $L \not\leq^p_{tt} R_H$

Claim 3.5.5 (Revised Claim 3.4.6)**.** *Given the above properties $H = K^v_U$ for some f-efficient universal prefix machine U (which by Property 4 ensures that the theorem holds)*

Proof. By Property 3 we have that $\sum_{x \in \{0,1\}^*} 2^{-F(x)+3} \leq 1/8$. Therefore it holds that

$$\sum_{x \in \{0,1\}^*} 2^{F(x)} \leq 1.$$

Using this along with Property 1, we then have by Proposition 3.4.3 that $K^u_{M'} = F + 2$ for some prefix machine M' and constant d', where $u(x) = 2^{(t_L(t^*(|x|)))^{d'}}$. By Proposition 3.4.5 we have that $K^{t^*}_{U'} = K^{t^*}_{U_0} + 4$ for some $n^{c'}$-efficient universal prefix machine U'. Therefore, by Proposition 3.5.4, $H(x) = \min(K^{t^*}_{U_0}(x) + 5, F(x) + 3) = \min(K^{t^*}_{U'}(x), K^u_{M'}(x)) + 1$ is K^v_U for some f-efficient universal prefix machine U and constant $c > 1$, where $v(|x|) = 2^{(t_L(t^*(|x|)))^c}$. (In this last step we are using the fact that $f(n) > 2^{(t_L(n^k))^k}$ for any constant k to ensure that U is an f-efficient universal prefix machine by Proposition 3.5.4). □

The construction is virtually the same as in the proof of Theorem 3.4.1.

There is one change from the proof of Theorem 3.4.1 in how the games are played. During the preprocessing step of a game $\mathcal{G}_{e,x}$, all queries z such that $t^*(|z|) \leq |x|$ are fixed according to $\min(K^{t^*}_{U_0}(z) + 5, F^*(z) + 3)$.

If we get stuck during our lexicographical search to find a suitable x' to play the game $\mathcal{G}_{e,x'}$, then this implies that the language L is in PSPACE, since by Proposition 3.2.5, for some constant k fixing all queries z such that $t^*(|z|) \leq |x|$ according to $\min(K^{t^*}_{U_0}(z) + 5, F^*(z) + 3)$ can be done in $|x|^k 2^{|z|} t^*(|z|) \leq |x|^k t^*(|z|)^2 \leq |x|^{k+2}$ time (and then it is a PSPACE computation to determine which side the F player has a winning strategy for).

It remains to prove the following claim.

Claim 3.5.6. *$F(z)$ is computable in time $2^{(t_L(t^*(|z|)))^d}$ for some constant d.*

Proof. Suppose during the construction we as the F player make a move that sets a query z to be nonrandom during a game $\mathcal{G}_{e,x}$. Note that $t^*(|z|) > |x|$, otherwise z would have been fixed during the preprocessing stage of the game.

As in the proof of Claim 3.4.10, we can bound the total amount of time steps elapsed before z is made nonrandom in the construction to be at most

$$T = 2^{|x|^d} 2^{|z|} t_L(|x|) t^*(|z|) + s < 2^{(t_L(t^*(|z|)))^d}$$

□

This concluses the proof of Theorem 3.5.3. □

3.6 An Approach Towards BPP = P?

It is popular these days to conjecture that BPP = P, and much of this popularity is owing to results such as those of Impagliazzo and Wigderson [IW97], who showed that BPP = P if there is a problem in E that requires circuits of exponential size. But note that a proof that BPP = P that proceeds by first proving circuit size lower bounds yields *much* more than "merely" a proof that BPP = P. It also provides a recipe that one can follow, to start with an arbitrary probabilistic algorithm and replace it with an equivalent deterministic one of comparable complexity.

Indeed, Goldreich has recently argued that *any* proof of BPP = P must proceed along these lines, in that any proof that these classes are equal yields pseudorandom generators that are suitable for derandomizing BPP [Gol11a, Gol11b].

But there is a catch! Goldreich's proof requires that the BPP = P question be phrased in terms of *promise* problems, rather than using the more traditional definition in terms of language classes, that we have used here.

We do not dispute Goldreich's assertion that the formulation in terms of promise problems is in many ways more natural and useful than the traditional definition. And we certainly agree that it would be much more useful to have a recipe for obtaining derandomizations, rather than merely a proof that a derandomization must exist. But we find it intriguing that a proof that $\mathsf{DTTR} = \mathrm{P}$ would prove that $\mathrm{BPP} = \mathrm{P}$ merely by showing that there would be a contradiction otherwise, and owing to the highly non-computable objects in the definition, it is not clear that such a proof would lend itself to an effective construction of a general-purpose derandomization algorithm. (In particular, it is not clear that it would yield the equality of the promise classes.) Similarly, proving $\mathrm{BPP} = \mathrm{P}$ by showing that $\mathsf{TTRT} = \mathrm{P}$ or $\mathsf{TTRT}' = \mathrm{P}$ would not seem to immediately yield derandomization algorithms because of the extremely high time-bounds used in the definitions of those classes. That is, since any such proofs would deliver less than a proof that yields a derandomization, it is at least conceivable that they would be easier to obtain.

We do not wish to suggest that we have any idea of how to obtain such a proof. After all, we are currently unable even to prove $\mathsf{DTTR} \subseteq \mathrm{P/poly}$, or that either TTRT or TTRT' is contained in BPP.

Also, it is clear that such a proof must use non-relativizing techniques. For instance, the work of [BFKL10] shows that, for any decidable oracle B, BPP^B is P^B-truth-table reducible to R_{K_U} for every U. (There is no need to add an oracle to the definition of R_{K_U}.) Thus it is not true that, for every decidable B, $\Delta_1^0 \cap \bigcap_U \{A : A \leq_{tt}^{\mathrm{p}^B} R_{K_U}\} = \mathrm{P}^B$, because Heller [Hel86] has presented such a B relative to which $\mathrm{BPP}^B = \mathrm{NEXP}^B$.

3.7 Conclusion

This chapter gave mixed results in the quest to exactly characterize BPP in terms of resource-bounded reductions to the Kolmogorov random strings. On the one hand progress was made in the time-bounded Kolmogorov complexity setting, but also it was shown that similar techniques do not work in the unbounded Kolmogorov complexity case, and that there are significant differences between R_K and R_{K_t} in this setting.

Perhaps a first step towards further progress is to figure out a way to combine the ideas of this chapter with the techniques of Chapter 2, which take advantage of the fact that DTTR is defined in terms of the intersection over all universal prefix machines. The P/poly upper bound for TTRT' from this chapter (and the related upper bound for TTRT) holds *regardless* of which universal machine U is used to define R_K, so really the upper bound holds for a much larger class than is actually being considered. Although this stronger inclusion worked for a non-uniform upper bound such as P/poly, certainly this cannot be the case for a uniform upper bound such as BPP, since there are languages of arbitrarily large uniform time-complexity in the class $\Delta^0_1 \cap \{A : A \leq^{\mathrm{p}}_{tt} R_{K_U}\}$ for a specific universal prefix machine U. It appears that new ideas will be needed in order to prove or disprove BPP $= \mathsf{DTTR}$.

Chapter 4

Lower Bounds for an OBDD-based Proof System Using Random Formulas

4.1 Background on Propositional Proof Complexity

Propositional proof complexity is a sub-area of computational complexity concerned with the following type of question: given a proof system P and an unsatisfiable propositional formula σ, what is the shortest refutation of σ in P? Here an unsatisfiable propositional formula is a formula made up of a finite set of Boolean variables and logical connectives such that regardless of how the variables are set to TRUE or FALSE, the formula always evaluates to FALSE. For our purposes we can restrict attention to formulas presented in conjunctive normal form (CNF).

A propositional proof system can be formally defined as a polynomial-time function P such that

$$\forall \sigma \in \mathcal{U} \ \exists y [P(\sigma, y) = 1]$$
$$\forall \tau \notin \mathcal{U} \ \forall y [P(\tau, y) = 0]$$

where $\mathcal{U}$ is the set of all unsatisfiable propositional formulas encoded as strings. For $\sigma \in \mathcal{U}$, we think of y as a proof that σ is unsatisfiable (i.e. a refutation). This definition captures the intuitive features of a propositional proof system:

- **Completeness**: It should be possible to refute any unsatisfiable formula.
- **Soundness**: It should be impossible to refute any satisfiable formula.

- **Efficiency**: Verifying whether a refutation is correct should be an efficient process.

Note that there is no restriction on the size of a refutation y, only that checking whether y is correct can be done in polynomial time. If a proof system P has the additional feature that there exists a polynomial p such that for any $\sigma \in \mathcal{U}$ there exists a correct refutation y of σ such that $|y| \leq p(|\sigma|)$, we say that the proof system P is *polynomially-bounded*. Cook and Reckhow introduced this abstract notion of a proof system[1] and they also established the following fundamental connection between propositional proof complexity and the standard computational complexity.

Theorem 4.1.1 ([CR79]). NP = coNP *if and only if there exists a polynomially-bounded proof system.*

It is generally conjectured that NP $\neq$ coNP, in which case for every proof system P there exists certain families of unsatisfiable formulas for which the smallest refutations in P must be of super-polynomial size. This suggests a natural program: attempt to show that stronger and stronger proof systems P are not polynomially-bounded as progress towards showing that NP $\neq$ coNP.

4.1.1 Examples of Propositional Proof Systems

A whole landscape of proof systems of varying strengths has been mapped out and studied – here we give a brief overview of some important examples (for a more in-depth look, see for instance the surveys [BP98, Seg07, Kra95].) All of these proof systems are *line-based*; that is, a refutation in the proof system is a sequence of lines, where each line represents a propositional formula that is satisfiable under the assumption that the formula being refuted is satisfiable. Although this property is not required according to the abstract Cook-Reckhow definition of a proof system, almost all concrete proof systems studied in the literature have this property. Each line is either a representation

[1]Originally Cook and Reckhow defined proof systems in terms of proofs of tautologies – propositional formulas that evaluate to TRUE regardless of how the variables are set. But proof systems can be equivalently defined in terms of refutations of unsatisfiable formulas, and we will use this alternative formulation throughout the next two chapters, as it will simplify the exposition in our case.

of one of the clauses of the formula being refuted (called an *axiom*) or is derived from previous lines using the sound inference rules of the proof system. The final line of the refutation represents a contradiction (an obviously unsatisfiable proposition); because the inference rules of the proof system are sound, this demonstrates that the formula being refuted in fact must have been unsatisfiable.

There are two fundamental ways in which the proof systems below differ:

1. How are lines of a refutation represented? Proof systems that have more expressive representations tend to have smaller refutations (i.e. be more powerful) than those systems with less expressive representations. For instance, a proof system where every line must be a clause is weaker than a proof system where each line is allowed to be a formula or a circuit.
2. What are the inference rules of the proof system? In other words, what are the rules by which a new non-axiom line is allowed to be derived from earlier lines?

- **Resolution**: Resolution is one of the simplest non-trivial proof systems that has been studied. In a resolution refutation, every line of the refutation is a clause. There is a single inference rule in the system: A line $C \vee D$ may be derived from two previous lines $C \vee x$ and $D \vee \bar{x}$, where x is a variable and C and D are arbitrary disjunctions of literals. The final line of a resolution refutation is the empty clause, $\emptyset$, which is trivially unsatisfiable.

  Resolution is a weak system and is one of the few proof systems that is relatively well-understood. There are many families of unsatisfiable formulas based on combinatorial principles which provably require exponential-size resolution refutations, including formulas based on the pigeonhole principle [Hak85] and Tseitsin formulas [Tse68]. Randomly generated formulas, which will be defined later and are our focus in this dissertation, also are exponentially-hard for resolution.

- **Frege systems**: In a Frege system the lines of the refutation are propositional formulas. There are a number of derivation rules similar to the ones that may be found in a standard textbook on propositional logic. For instance, a Frege system is generally equipped with a *modus ponens* rule: The line B may be derived from the lines $\neg A \vee B$ and A (where A and B are two arbitrary sub-formulas).

Frege systems are very strong; no super-quadratic lower bounds on the size of Frege refutations are currently known for any family of unsatisfiable formulas. One can define weaker versions of these systems by restricting the type of formulas that may constitute a line. For instance, AC^0-Frege (also known as *constant-depth* Frege), is a system where each line of a refutation must be a constant-depth formula. Exponential lower bounds for certain families of formulas such as the pigeonhole formulas are known for this restricted system [Ajt94, PBI93]. One can actually define stronger versions of Frege systems as well by expanding the representations; for instance *Extended-Frege* systems, in which lines are allowed to be circuits instead of just formulas, is believed by many to be strictly stronger than regular Frege systems.[2] (Although, of course, since we basically have no lower bounds even for Frege systems, we are very far from being able to prove such a conjecture).

- **Algebraic systems**: In algebraic proof systems, lines are generally represented as equations or inequalities. For instance, in the *polynomial calculus* system, lines are represented as polynomial equations of the form $Q(\mathbf{x}) = 0$. To refute an unsatisfiable formula ϕ, each clause of ϕ is first converted to an axiom that is satisfied if and only if the clause is satisfied. For instance, if $C = x_i \vee \bar{x_j} \vee x_k$ is a clause of ϕ, then this corresponds to an axiom $(1 - x_i)x_j(1 - x_k) = 0$. (For each variable x_i there is also an axiom of the form $x_i(1 - x_i) = 0$, which enforces that x_i is a 0-1 variable). The two inference rules of the system are:
  - **Linear combination**: From the lines $Q(\mathbf{x}) = 0$ and $R(\mathbf{x}) = 0$, one can derive the line $aQ(\mathbf{x}) + bR(\mathbf{x}) = 0$, where a, b are elements of some field associated with the proof system.

[2]The standard way to compare propositional proof systems is by means of *polynomial simulations*. A proof system P polynomially-simulates a proof system Q if there exists a polynomial-time function f such that if π is a refutation of an unsatisfiable formula ϕ in Q, then $\pi' = f(\pi)$ is a refutation of ϕ in P. We can say P is strictly stronger than Q if P polynomially-simulates Q but Q does not polynomially-simulate P.

  - **Multiplication by a variable**: From the line $Q(\mathbf{x}) = 0$ one can derive the line $x_i Q(\mathbf{x}) = 0$ where x_i is any variable.

  The final line of a polynomial calculus refutation is the equation $1 = 0$, a contradiction demonstrating that the formula ϕ was unsatisfiable. Exponential lower bounds for the polynomial calculus are known for many standard families of unsatisfiable formulas, such as the pigeonhole principle and random formulas [Raz98, BSI99].

  In the *cutting planes* system, lines are represented as linear inequalities of the form $\sum_{i=1}^{n} a_i x_i \geq X$, where each a_i is an integer. In the example above, the clause $C = x_i \vee \bar{x_j} \vee x_k$ would be converted to an axiom $x_i + (1 - x_j) + x_k \geq 1$. The three inference rules of the system are:

  - **Addition**: From the lines $\sum_{i=1}^{n} a_i x_i \geq X$ and $\sum_{i=1}^{n} b_i x_i \geq Y$, one can derive the line $\sum_{i=1}^{n} (a_i + b_i) x_i \geq X + Y$.
  - **Multiplication by a positive integer**: From the line $\sum_{i=1}^{n} a_i x_i \geq X$ one can derive the line $\sum_{i=1}^{n} c a_i x_i \geq cX$ for any positive integer c.
  - **Division by a positive integer**: From the line $\sum_{i=1}^{n} c a_i x_i \geq X$, one can derive the line $\sum_{i=1}^{n} a_i x_i \geq \lceil X/c \rceil$

  The final line of a cutting planes refutation is the contradiction $0 \geq 1$. The cutting planes system is a strong system; the only known super-polynomial lower bounds for the system were proved by a reduction to Razborov's monotone circuit lower bounds for the clique problem [Raz85] using the feasible interpolation method [Pud97].

- **OBDD-based systems**: There are many other proof systems based on different representations than the ones discussed above. A specific proof system that we will focus on in this chapter is a proof system where each line is represented by an ordered binary decision diagram (OBDD), which is a special type of branching program. The definition of this system and what we know about its limitations are discussed in detail in section 4.3.

4.1.2 Proof Complexity and SAT Solving

Building up techniques towards proving NP $\neq$ coNP is not the only reason for studying proof complexity; studying specific proof systems is independently interesting, partially due to the importance for real-world applications of developing good algorithms for solving the satisfiability problem. Because many important optimization problems can be efficiently encoded as satisfiability instances, an enormous amount of research has been invested into developing SAT solvers that outperform the trivial exponential-time brute force algorithm. Proof complexity lower bounds can be used to lower bound the runtime of certain classes of these solvers.

For instance, many of the best SAT solvers used in practice are variants of the DPLL algorithm [DLL62, DP60], a simple branching algorithm based on setting a variable to either TRUE or FALSE and then recursively considering the simplified formulas that result. It has been shown that from the execution of a DPLL-based SAT solver on an unsatisfiable formula σ, one can derive a resolution refutation of σ that has size proportional to the runtime of the solver. This holds even for DPLL-based solvers that use sophisticated heuristics to choose the order to branch on variables and "clause learning" to do more efficient backtracking [BKS04]. Thus, any resolution lower bounds immediately imply lower bounds for the runtime of these algorithms as well. As noted previously, resolution is one of the simplest and well-studied proof systems; exponential lower bounds on the size of resolution refutations of many families of unsatisfiable formulas are known, which proves that any DPLL-based SAT solver will have exponential runtime on these same families of formulas. This is a general principle; anytime we derive lower bounds for a propositional proof system on a family of unsatisfiable formulas, as a byproduct we get lower bounds on the runtime of some class of co-nondeterministic SAT solving algorithms on those same instances.

4.2 Random Formulas

In this dissertation we focus on the ability of proof systems to refute randomly generated 3-CNF formulas, where each clause is independently and uniformly chosen at random from all possible clauses. Random CNF formulas have been studied extensively, both as a benchmark for measuring in some sense the average case performance of SAT solvers, and also as a tool for proving proof complexity lower bounds. It is well-known that if a random 3-CNF formula on n variables is generated with Δn clauses for large enough constant Δ, then with high probability the formula will be unsatisfiable [DBM03]. The lack of structure in these formulas makes them hard to refute; indeed, it is conceivable that they require exponential-size refutations in *any* proof system, and since even generating candidate hard formulas for strong proof systems such as Frege systems can be difficult (see for instance [Raz03]), they are a natural choice for proving lower bounds.

Another appealing feature of random formulas is that they do not have an analogue in circuit complexity. Progress on propositional proof complexity has run somewhat parallel to circuit complexity, usually with breakthroughs in circuit complexity lower bounds coming first and informing new lower bounds for proof complexity. For instance, the combinatorial lower bounds for constant-depth Frege systems rely on random restrictions and a switching lemma adapted from the AC^0 circuit lower bounds of [FSS84, Has86]. Sometimes the connection to circuit complexity is more direct; as mentioned before, in the case of the cutting planes system, the only known super-polynomial lower bounds for any class of formulas rely on a reduction to monotone circuit lower bounds. Presently, progress in propositional proof complexity lags slightly behind circuit complexity in the sense that we have some lower bounds for constant depth circuits with mod gates [Raz87, Smo87, Wil11], but no lower bounds for restrictions of Frege systems that work with such circuits. Proving circuit lower bounds is notoriously difficult, and a number of barriers such as relativization, natural proofs, and algebrization have been identified in that field [BGS75, RR97, AW08]. If a major breakthrough in proof complexity is to occur anytime soon, such as super-polynomial lower bounds for Frege systems or the even stronger Extended Frege systems, it is likely that there will have to be a departure from the paradigm of adapting circuit complexity techniques.

Random formulas are intriguing in this respect because they highlight a major difference between proof complexity and circuit complexity. In circuit complexity we are not interested in lower bounds on the circuit size of arbitrary functions, as a trivial counting argument shows that a randomly chosen function will have high circuit complexity. Instead, we try to prove lower bounds for *explicit* functions (usually defined as functions in some complexity class such as NP). However, it is not clear how to formulate the notion of a random function in a class like NP in a useful way. On the other hand, no such trivial counting argument shows that a random formula must have large proof complexity, and proving that all proof systems require super-polynomial size refutations of random formulas would be sufficient to show that NP $\neq$ coNP. This opens up the possibility of employing probabilistic methods and other nonconstructive techniques to prove proof complexity lower bounds for random formulas that perhaps cannot be used in the circuit complexity setting.

However, despite their promise, the lack of structure that makes random formulas potentially useful for proving lower bounds can be an impediment as well. Historically, lower bounds on random formulas have come only after lower bounds on explicit formulas have already been proven. For instance, the first resolution lower bounds were for classes of formulas capturing the idea of "counting" such as Tseitin formulas and propositional encodings of the pigeonhole principle [Tse68, Hak85]. It was only later that Szémeredi and Chvátal were able to adapt some of these techniques to work for random formulas as well [CS88]. In the case of constant-depth Frege systems, we have exponential lower bounds for certain explicit families of formulas like the pigeonhole principle [Ajt94, PBI93], but no super-polynomial bounds are known for random formulas. Therefore, developing new techniques for working with random formulas is an important task.

Along with random 3-CNF formulas, in this chapter we will also consider random 3-XOR formulas. Like a 3-CNF formula a 3-XOR formula is a conjunction of clauses, but each clause is satisfied if and only if exactly one or three of its literals are satisfied. Unlike in the 3-CNF case, determining satisfiability of a 3-XOR formula is known to be computable in polynomial time, since such formulas can be equivalently represented as a system of linear equations over $\mathbf{F}_2$, and then an algorithm such as Gaussian elimination can be used to test the solvability of the system. However, random 3-XOR formulas retain a lot of the important properties of random 3-CNF formulas, and because they are easier to reason about they have been useful in proving lower bounds for weak proof systems (e.g. [Ale05]).

4.3 OBDD-based Proof Systems

Ordered Binary Decision Diagrams (OBDDs) are data structures for representing Boolean functions that were originally introduced in [Bry86] and have found a wide variety of applications in areas of computer science such as VLSI design and model checking. They have also emerged as a basis for SAT solving algorithms that have been demonstrated to be competitive on certain classes of formulas with the state-of-the-art DPLL-based solvers that are generally used in practice [PV04, HD04]. Informally, they are read-once branching programs where variables must be queried according to a fixed order. Part of what makes OBDDs so useful is that their relatively rigid structure makes it possible to manipulate them efficiently, as illustrated by the following theorem:

Theorem 4.3.1 ([Bry86])**.**

- *For any given Boolean function f on n variables and variable order π there is a unique (up to isomorphism) minimal OBDD computing f.*
- *Computing the conjunction of two OBDDs can be done in polynomial time.*
- *Determining whether an OBDD representing a function f_1 majorizes an OBDD representing a function f_2 (i.e for all x, $f_1(x) \geq f_2(x)$) is computable in polynomial time .*

A proof system based on OBDDs was introduced in [AKV04]. The basic idea of such a system is simple: Given an unsatisfiable 3-CNF (or 3-XOR) formula $\mathcal{F}$, an OBDD refutation of $\mathcal{F}$ with respect to a variable order π is a sequence $\text{OBDD}_1, \text{OBDD}_2, \ldots \text{OBDD}_t \equiv 0$, where each OBDD_i uses the variable order π and is either the OBDD representation of a clause from $\mathcal{F}$ (an axiom), or is the conjunction of two OBDDs derived earlier (i.e. $\text{OBDD}_i = \text{OBDD}_j \wedge \text{OBDD}_k$ for some $j, k < i$). One can also include a weakening rule, so that OBDD_i may also be an OBDD such that OBDD_i majorizes OBDD_j for some $j < i$. Such a refutation system is sound and complete, and because computing the conjunction of two OBDDs can be done in polynomial time (as well as determining whether one OBDD majorizes another in the case of a weakening), verifying whether a refutation is correct is also polynomial-time computable. Thus these OBDD-based systems qualify as propositional proof systems in the formal sense introduced by Cook and Reckhow. The OBDDs representing axioms in this type of refutation are small, as well as the final OBDD OBDD_t. Therefore, if a refutation in one of these OBDD-based systems has a polynomial number of steps, whether it is polynomial-size or not depends only on whether one of the intermediate OBDDs computed along the way has super-polynomial size. The only nondeterministic choices the prover must make are which variable order π to use, and in what order to combine OBDDs. (If a weakening rule exists, the prover must also choose when and how to use it). These choices can be crucial however in determining the size of the refutation; for instance, it is a simple exercise to show that for certain functions the OBDD representation has size $O(n)$ according to one variable order yet size $\Omega(2^n)$ according to another order.

By restricting the options the prover has in making these choices, one can define different variants of this OBDD-based system that have varying strengths. One reason for doing so is that no current OBDD-based SAT solver takes full advantage of the power offered by the underlying OBDD-based proof system in its unrestricted form. This is a common phenomenon in SAT solving – basing solvers on more powerful proof systems does not necessarily make the solvers better. The reason is that as the proof systems become more powerful, trying to deterministically make the nondeterministic choices of the proof system becomes an increasingly difficult task. This is highlighted by the fact that the best general purpose SAT solvers in use today are variants of the DPLL algorithm, which is based on the resolution system, one of the weakest proof systems that has been studied. In the case of OBDD-based systems, it is not clear how to best make use of the full weakening rule, and even determining the best variable order to use in an OBDD representation of a single function is an NP-complete problem [BW96]. Particularly when considering random formulas, because of their symmetry and lack of structure, it seems unlikely that one variable order would be exponentially better than another, or even if such a good order did exist that it could be found efficiently.

However, the sheer number of different possible variable orders make proving such a fact difficult from a technical standpoint.

From a theoretical point of view, restricting these OBDD systems creates interesting intermediate systems. It was proved in [AKV04] that allowing unrestricted use of the weakening rule makes the OBDD proof system as strong as CP*, a variant of the cutting planes system where coefficients are represented in unary, that is strictly stronger than resolution and for which the only known lower bounds are based on feasible interpolation. However, if we do not allow weakening the story changes significantly – in this case there exist certain families of unsatisfiable formulas for which the smallest OBDD refutations are exponentially larger than the smallest resolution refutations [TSZ10]. Despite this apparent weakness, it has not been proved that the Frege system, a powerful system that could even conceivably be optimal, can polynomially-simulate this restricted OBDD system. The reason is that the lines of Frege systems are formulas, which cannot directly simulate the dag-like structure of OBDDs. Thus studying different variations of restricted OBDD-based systems is one possible route towards bridging the gap between systems we know to be weak and those for which we do not have lower bounds on natural families of formulas.

Krajíček gave exponential lower bounds for the OBDD-based system of [AKV04] in its full generality using a form of the feasible interpolation method [Kra07], and these are currently the only lower bounds known for this strongest variant. Tveretina et al. showed that if the weakening rule is disallowed, then an OBDD-based refutation of the pigeonhole principle must have exponential size [TSZ10], building upon a similar result from Groote and Zantema [GZ03], who had also restricted the system to only consider specific variable orders.

4.4 Statement of Main Results

In this chapter we take a first step towards understanding the limitations of OBDD-based systems to refute *random* formulas by proving exponential lower bounds for certain restricted variants.

In particular we consider two restricted OBDD-based systems, which we can denote by OBDD* and OBDD+. In both systems the weakening rule is excluded. In the OBDD* system the variable order that will be used for the refutation is fixed before the random formula is chosen. Because random formulas are generated symmetrically with respect to the variables, without loss of generality we can fix the identity order $\mathcal{I}$ that orders a set of variables $x_1, x_2, \dots x_n$ as $x_1 < x_2 < \cdots < x_n$. In the OBDD* system the prover has the freedom to combine OBDDs during the refutation in an arbitrary way. In the OBDD+ system, the prover has the freedom to choose any variable order π *after* seeing the random formula ϕ that is to be refuted. However, during the refutation, the clauses of ϕ (represented as OBDDs) must be combined in a predetermined fashion corresponding to some canonical ordering of the clauses in ϕ.

The following two theorems are our main results:

Theorem 4.4.1 ([FX13]). *Let Δ be a sufficiently large constant. There exists an $\epsilon > 0$, such that with high probability when ϕ is a random 3-CNF formula on n variables with clause density Δn, ϕ is unsatisfiable and any OBDD* refutation of ϕ must have size at least $2^{\epsilon n}$.*[3]

Theorem 4.4.2 ([FX13]). *Let Δ be a sufficiently large constant. There exists an $\epsilon > 0$ such that with high probability when ϕ is a random 3-XOR formula on n variables with clause density Δn, ϕ is unsatisfiable and any OBDD+ refutation of ϕ must have size at least $2^{\epsilon n}$.*

The progress we make in this chapter is summarized in Figure 4.1.

4.5 Preliminaries and Notation

We will denote a set of n Boolean variables as $\{x_1, \dots, x_n\}$. A literal x_i^j, $j \in \{0, 1\}$, is either a variable or its negation. An assignment α to a set of n variables is a function

[3]This theorem can be proved almost identically in the case where we consider a random 3-XOR formula as well. Also, a close inspection of the proof shows that for either the 3-CNF or 3-XOR case, if instead of fixing the variable order $\mathcal{I}$ we allow the prover to fix any set S of $2^{\delta n}$ variable orders for sufficiently small δ before seeing the random formula ϕ, then to choose one of the variable orders from S after seeing ϕ, the theorem still holds in this scenario as well.

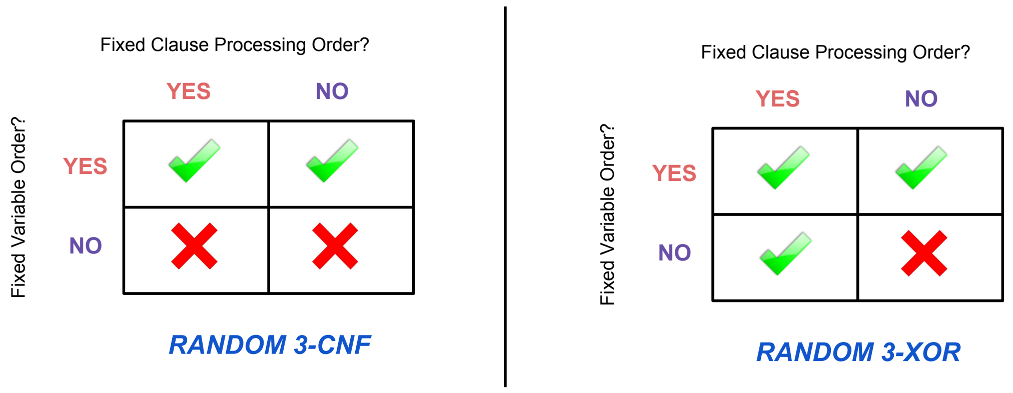

Figure 4.1: *A summary of the results from this chapter. We consider different OBDD-based proof systems, none of which include a weakening rule. The systems differ according to two possible restrictions: (1) Is the variable order that will be used in the refutation fixed before the random formula is chosen? (2) Are the clauses processed in the refutation according to a canonical order in which they appear in the input formula? We consider both random 3-CNF and random 3-XOR formulas. A check mark appears in the box corresponding to a given proof system and type of random formula if we prove exponential lower bounds for this combination in this chapter, and an X appears in the box if proving lower bounds in this case is still open.*

$[n] \rightarrow \{0,1\}$, where $[n]$ denotes the set $\{1,2,\ldots,n\}$. An assignment α satisfies a literal x_i^j if and only if $\alpha(i) = j$.

A clause C is a set of literals. An assignment α satisfies C as a CNF clause if and only if α satisfies some literal in C, and α satisfies C as an XOR clause if and only if α satisfies an odd number of literals in C. A 3-CNF (3-XOR) formula $\mathcal{F}$ over n variables is a list of clauses $(C_1,\ldots,C_m)$, where each of the clauses contains three literals from variables in the set $\{x_1,\ldots x_n\}$. It is satisfied by an assignment α if and only if every clause in $\mathcal{F}$ is satisfied by α as a CNF (XOR) clause. If it is irrelevant whether we are referring to a 3-CNF formula or a 3-XOR formula, we will often refer to the formula simply as a 3-formula.

Definition 4.5.1 (Random 3-formula). *A random 3-formula ϕ on n variables with clause density Δ is a 3-formula $(C_1,\ldots,C_{\Delta n})$, where each clause C_i is chosen uniformly at random from all of the $2^3\binom{n}{3}$ possible clauses.*[4]

Technically a random 3-formula ϕ is a fixed formula chosen according to the above

[4]Defining a random formula in one of the other standard ways, for instance by having the clauses chosen without replacement, would not affect the results in this chapter.

process. However, sometimes we will slightly abuse notation and speak about the probability that ϕ has a certain property; in this case we are referring to the probability that a formula chosen according to the distribution defined by the above process has that property.

Let π be a total order on a set of variables $\{x_1, \ldots, x_n\}$. We will refer to π simply as an *order*. Alternatively, we can view π as a permutation such that $\pi(i) = j$ if and only if the i-th variable in the order of π is x_j. We will also write $\pi^{-1}(j) = i$ to indicate that $\pi(i) = j$. We also define the identity order $\mathcal{I}$ such that for all i, $\mathcal{I}(i) = i$.

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function on n variables and let $\mathbf{z} \in \{0,1\}^t$ for $t \leq n$. We define $f|_{\pi,\mathbf{z}}$ to be the function $f' : \{0,1\}^{n-t} \to \{0,1\}$ that is the function f restricted so that for each $1 \leq i \leq t$, if $\pi(i) = j$, then x_j is fixed to the constant value $\mathbf{z}_i$.

Definition 4.5.2 (OBDD). *Given an order π on $\{x_1, \ldots, x_n\}$, an ordered binary decision diagram with respect to π, denoted by* OBDD_π*, is a branching program with the following structure. An* OBDD_π *is a layered directed acyclic graph with layers 1 through $n+1$. Layer 1 contains a single root node, and layer $(n+1)$ contains two final nodes, one labeled with the value 0 and the other labeled with the value 1. Every node in layers 1 through n has outdegree two: such a node v on level i has one outgoing edge to a node on level $i+1$ labeled with the value 0, and another outgoing edge to a node on level $i+1$ labeled with the value 1.*

An OBDD_π *defines a Boolean function $\{0,1\}^n \to \{0,1\}$ in the following way. For an assignment α on n variables, we start at the root node, and for $i = 1$ to n, advance along the edge labeled with $\alpha(\pi(i))$. When this process is complete, we will have arrived at one of the final nodes. If this final node is labeled with* 0*, then we define* $\mathrm{OBDD}_\pi(\alpha) = 0$*, and otherwise we define* $\mathrm{OBDD}_\pi(\alpha) = 1$*, where now we are associating α with an n bit string in the natural way.*

$|\mathrm{OBDD}_\pi|$ *denotes the size (the number of nodes) of the OBDD.*

An important property of OBDDs is that for a given Boolean function $f : \{0,1\}^n \to$

$\{0,1\}$ and an ordering π, there is a unique minimal OBDD$_\pi$ up to isomorphism computing f [Bry86]. Thus for a given f we can safely refer to OBDD$_\pi(f)$ as *the* OBDD computing f according to π.

The following simple theorem (and corollary) provide general techniques for proving lower bounds on $|\text{OBDD}_\pi(f)|$.

Theorem 4.5.3 ([SW93]). *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function on n variables and π an order. Let $k = |\{f|_{\pi,\mathbf{z}} : \mathbf{z} \in \{0,1\}^t\}|$ (i.e., k counts the number of distinct sub-functions of f that can be produced by fixing the first t variables according to π). Then the t-th level of $OBDD_\pi(f)$ contains k nodes.*

Corollary 4.5.4 ([TSZ09]). *Let f be a Boolean function on n variables and π an order. Suppose the following conditions hold*

1. *$x_1, \cdots, x_t$ are the least t variables according to π for some $t < n$.*
2. *$B \subseteq \{1, \ldots, t\}$.*
3. *$\mathbf{z} \in \{0,1\}^t$.*
4. *For all $\mathbf{x}, \mathbf{x}' \in \{0,1\}^t$, if $\mathbf{x} \neq \mathbf{x}'$ and $\mathbf{x}_i = \mathbf{x}'_i = \mathbf{z}_i$ for all $i \notin B$, then there exists $\mathbf{y} \in \{0,1\}^{n-t}$ such that $f(\mathbf{x}, \mathbf{y}) \neq f(\mathbf{x}', \mathbf{y})$.*

Then $|\text{OBDD}_\pi(f)| \geq 2^{|B|}$.

Definition 4.5.5 (OBDD$^*_\pi$ refutation). *Given an unsatisfiable 3-formula $\mathcal{F}$ and an order π, an $OBDD^*_\pi$ refutation of $\mathcal{F}$ is a sequence $OBDD_\pi(f_1), OBDD_\pi(f_2), \cdots, OBDD_\pi(f_t \equiv 0)$ such that for each f_i one of the following conditions is satisfied:*

1. *f_i is a clause of $\mathcal{F}$. (In this case we say that f_i is an* axiom*).*
2. *$f_i = f_j \wedge f_k$ for some $j, k < i$.*

The size of the $OBDD^*_\pi$ refutation is defined as $\sum_{i=1}^{t} |OBDD_\pi(f_i)|$.

We define $S^*_\pi(\mathcal{F})$ to be the minimum size of any OBDD$^*_\pi$ *refutation of $\mathcal{F}$. In this chapter we focus on $\pi = \mathcal{I}$ and thus will refer to $S^*_{\mathcal{I}}(\mathcal{F})$.*

Definition 4.5.6 (OBDD$_\pi^+$ refutation). *An* OBDD$_\pi^+$ *refutation of an unsatisfiable 3-formula $\mathcal{F} = (C_1, \ldots, C_m)$ is an* OBDD$_\pi^*$ *refutation where the clauses of $\mathcal{F}$ are processed one at a time in order. Precisely, an* OBDD$_\pi^+$ *refutation of $\mathcal{F}$ is a sequence $OBDD_\pi(f_1), OBDD_\pi(f_2), \cdots, OBDD_\pi(f_{2m} = 0)$ where for $1 \leq i \leq m$, $f_i = C_i$, $f_{m+1} = C_1$, and for $m+2 \leq j \leq 2m$, $f_j = f_{j-1} \wedge f_{j-m}$. We define $S_\pi^+(\mathcal{F})$ to be the size of the unique* OBDD$_\pi^+$ *refutation of $\mathcal{F}$, and we define $S^+(\mathcal{F})$ to be the minimum over π of $S_\pi^+(\mathcal{F})$.*

We will make use of the following bounds related to satisfiability thresholds.

Theorem 4.5.7 ([DBM03]). *There exists $\Delta^* \leq 4.51$ such that for large n, with high probability a random 3-CNF formula with n variables and clause density $\Delta > \Delta^*$ will be unsatisfiable.*

Theorem 4.5.8 ([DM02]). *There exists $\Delta^* \leq 0.91$ such that for large n, with high probability a random 3-XOR formula with n variables and clause density $\Delta > \Delta^*$ will be unsatisfiable.*

We will also need the following lemma, which is a restatement of a result that appeared in [CS88]. For S a subset of the clauses of a 3-formula $\mathcal{F}$, let $var(S)$ be the set of all variables that appear in at least one of the clauses of S (ignoring the sign of the literal). We call a 3-formula $\mathcal{F}$ on n variables an (x, y)-expander if for all subsets S of the clauses of $\mathcal{F}$ such that $|S| \leq xn$, $|var(S)| \geq y|S|$.

Lemma 4.5.9 ([CS88]). *For all $y < 2$ and $\Delta > 0$, there exists positive x such that with high probability a random 3-formula on n variables with clause density Δ will be an (x, y)-expander.*

Finally, we need two results on systems of distinct representatives that follow from Hall's marriage theorem. For a clause C, let $var(C)$ be the set of variables appearing in C, and for a set of clauses S, let $var(S) = \cup_{C \in S} var(C)$. We say a subset S of clauses has a system of distinct representatives (SDR) if there is a one-to-one function $\sigma : S \rightarrow var(S)$ such that for all $C \in S$, $\sigma(C) \in var(C)$.

Lemma 4.5.10 ([Hal35]). *Let S be a subset of clauses. S has an SDR if and only if for all $S' \subseteq S$, $|var(S')| \geq |S'|$.*

Lemma 4.5.11 ([CS88]). *Let S be a set of clauses and V a set of variables. S has an SDR σ with at most t elements of V in the range of σ if and only if it has an SDR and for all $S' \subseteq S$, $|S'| - |var(S') \setminus V| \leq t$.*

4.6 Proof of the OBDD* Case

The purpose of this section is to prove Theorem 4.4.1, which we now restate.

Theorem 4.6.1 (restatement of Theorem 4.4.1). *Let $\Delta > 4.51$. There exists a constant $\epsilon > 0$ such that, with high probability when ϕ is a random 3-CNF formula on n variables with clause density Δ, ϕ is unsatisfiable and $S^*_{\mathcal{I}}(\phi) \geq 2^{\epsilon n}$.*

The main work in our proof of Theorem 4.6.1 is proving the following lemma.

Lemma 4.6.2. *Let $\Delta > 4.51$. There exist constants $\delta, \epsilon > 0$ such that, with high probability when ϕ is a random 3-CNF formula on n variables with clause density Δ, ϕ is a $(\delta, 1.9)$ expander and the following holds: Let S be any subset of the clauses of ϕ such that $\delta n/2 \leq |S| \leq \delta n$, and let f_S be the conjunction of these clauses. Then $|OBDD_{\mathcal{I}}(f_S)| \geq 2^{\epsilon n}$.*

Proof of Theorem 4.6.1. Because $\Delta > 4.51$, by Theorem 4.5.7 with high probability ϕ will be unsatisfiable. Let $P = OBDD_{\mathcal{I}}(f_1), OBDD_{\mathcal{I}}(f_2), \cdots, OBDD_{\mathcal{I}}(f_t = 0)$ be an OBDD$^*_{\mathcal{I}}$ refutation of ϕ. Each f_i is a conjunction of some subset of clauses S of ϕ. Let $|f_i|$ denote $|S|$.

By Lemma 4.5.9, there exists a constant δ such that with high probability ϕ is a $(\delta, 1.9)$ expander. By Lemma 4.5.10 this means that any subset S of clauses of ϕ with $|S| \leq \delta n$ has an SDR. Any set of clauses S that has an SDR σ is satisfiable, since an assignment that for each clause $C \in S$ sets $\sigma(C)$ to the value that satisfies C will satisfy S. Therefore, since f_t is the constant 0 function, which is trivially unsatisfiable, $|f_t| \geq \delta n$. For every f_i that is an axiom, we have $|f_i| = 1$. If $f_i = f_j \wedge f_k$ for some $j, k < i$, then $|f_i| \leq |f_j| + |f_k|$. Therefore, for each $i \in t$, $|f_i| \leq 2 \max_{j<i} |f_j|$. This implies that

there exists $i \in [t]$ such that $\delta n/2 \le |f_i| \le \delta n$. By Lemma 4.6.2, $|OBDD_{\mathcal{I}}(f_i)| \ge 2^{\epsilon n}$, so P has size at least $2^{\epsilon n}$. $\square$

The remainder of this section is devoted to proving Lemma 4.6.2. First we prove a few other lemmas that will be useful towards this goal.

Lemma 4.6.3. *Let $\Delta > 0$ and $0 < \delta < \Delta$ be some constant. There exists $\epsilon > 0$, such that with high probability when ϕ is a random 3-formula on n variables with clause density Δ, for any set T of ϵn variables, the number of clauses from ϕ that contain a variable from T is less than δn.*

Proof. Let T be a set of variables such that $|T| = \epsilon n$. For any constant $c > 1$, the probability that a particular clause of ϕ contains a variable from T is less than $3c\epsilon$. Therefore, the expected number of clauses from ϕ that do not contain a variable from T is at least $(1 - 3c\epsilon)\Delta n$. Let $b = (1 - 3c\epsilon)\Delta$. Let X be the random variable that counts the number of clauses from ϕ that do not contain a variable from T. In order for the number of clauses from ϕ that contain a variable from T to be less than δn, we need that $bn - X < \delta n$, so $X > (b - \delta)n = (1 - \frac{\delta}{b})bn$.

By a Chernoff bound, the probability that $X \le (1 - \frac{\delta}{b})bn$ is less than $e^{-bn\left(\frac{\delta}{b}\right)^2/2} = e^{-\left(\frac{\delta^2}{2b}\right)n}$. The total number of sets of variables that contain exactly ϵn variables is $\binom{n}{\epsilon n} \le \left(\frac{en}{\epsilon n}\right)^{\epsilon n} = \left(\frac{e}{\epsilon}\right)^{\epsilon n}$. By a union bound, the probability that there exists some set of variables T with $|T| = \epsilon n$ such that more than δn clauses from ϕ contain a variable from T is less than

$$e^{-\left(\frac{\delta^2}{2b}\right)n} \cdot \left(\frac{e}{\epsilon}\right)^{\epsilon n}.$$

By choosing a small enough value of ϵ, this probability is exponentially small in n. $\square$

Lemma 4.6.4. *Let $\Delta > 0$ and $0 < \delta < \Delta$ be some constant. There exists $\epsilon > 0$, such that with high probability when ϕ is a random 3-formula on n variables with clause density Δ, the following property holds: For all sets S of clauses from ϕ with $|S| \ge \delta n$, there exists a set of clauses $T \subseteq S$ with $|T| = \epsilon n$ such that the clauses in T are disjoint (i.e. no two clauses of T share a common variable).*

Proof. Let ϵ' be the constant that comes out of Lemma 4.6.3 corresponding to δ. Take a maximal set $T \subseteq S$ such that the clauses in T are disjoint. Let $\epsilon = \frac{\epsilon'}{3}$. We have that $|T| \geq \epsilon n$, otherwise $var(T)$ is a set of $\epsilon' n$ variables that appear in at least δn clauses, which would violate Lemma 4.6.3. $\square$

Definition 4.6.5 (splits)**.** *Let t be a positive integer less than n and $\mathcal{F}$ a 3-formula. For a clause $C \in \mathcal{F}$, we say that t left-splits C according to an order π if there is exactly one variable $x_i \in var(C)$ such that $\pi^{-1}(i) \leq t$. In this case we define* $\mathrm{left}_{C,t,\pi} = x_i$*. Similarly, we say that t right-splits C according to an order π if there is exactly one variable $x_i \in var(C)$ such that $\pi^{-1}(i) > t$, and in this case define* $\mathrm{right}_{C,t,\pi} = x_i$*. If t either right-splits or left-splits C, then we will sometimes simply say that t splits C.*

Lemma 4.6.6. *Let $\Delta, \delta > 0$ be any constants, and for some $0 < \epsilon < 1$, let $\Gamma_\epsilon = \{\lceil \epsilon n \rceil, \lceil 2\epsilon n \rceil, \lceil 3\epsilon n \rceil, \cdots, \lceil (1-\epsilon)n \rceil\}$. Then with high probability when ϕ is a random 3-formula on n variables with clause density Δ, for any set of clauses S from ϕ, with $|S| \geq \delta n$, there exists $t \in \Gamma_\epsilon$ such that at least $(\delta - 7\epsilon\Delta)\epsilon n$ of the clauses are left-split by t according to $\mathcal{I}$.*

Proof. Say two variables x_i and x_j have distance d if $|i - j| = d$. Let C be a clause from ϕ. If all variables from C have pairwise distance at least ϵn, then some element of Γ_ϵ must left-split C according to $\mathcal{I}$. The probability that any two given variables from C have distance at least ϵn is at least $1 - 2\epsilon$, so the probability that all variables from C have pairwise distance at least ϵn is at least $1 - 6\epsilon$.

Therefore the expected number of clauses that are left-split according to $\mathcal{I}$ by at least one element of Γ_ϵ is at least $\Delta n(1 - 6\epsilon)$. Therefore, by a Chernoff bound, with high probability, for any constant $c > 1 - (1 - 6\epsilon) = 6\epsilon$, at most $c\Delta n$ clauses from ϕ are not left-split according to $\mathcal{I}$ by any element of Γ_ϵ.

Assume the above property of ϕ holds and consider an arbitrary set of δn clauses from ϕ. By an averaging argument, there exists $t \in \Gamma_\epsilon$ such that $(\delta - c\Delta)\epsilon n$ of these clauses are left-split according to $\mathcal{I}$ by t. Choosing $c = 7\epsilon$ completes the proof of the lemma. $\square$

Lemma 4.6.7. *Let $\Delta > 4.51$, and let δ' be the constant that comes out of Lemma 4.5.9 such that with high probability a random 3-formula with clause density Δ is a $(\delta', 1.9)$ expander. Let $\delta < \delta'$ be some constant. There exists constants $\gamma, \epsilon > 0$, such that with high probability when ϕ is a random 3-formula on n variables with clause density Δ, the following property holds: For all sets T of clauses from ϕ, with $\delta n \leq |T| \leq \delta' n$, there exists $S \subseteq T$ such that*

1. *$|S| = \gamma n$.*
2. *There exists $t \in \Gamma_\epsilon$ such that every clause $C \in S$ is left-split by t according to $\mathcal{I}$.*
3. *The clauses of S are disjoint.*
4. *T has an SDR σ, such that for every clause $C \in S$, exactly one variable in C is in the range of σ.*

Proof. Suppose the conclusions of Lemma 4.5.9, Lemma 4.6.4, and Lemma 4.6.6 hold with respect to ϕ (which occurs with high probability). Let T be a set of clauses from ϕ such that $\delta n \leq |T| \leq \delta' n$. By Lemma 4.6.6, there exists a constant ϵ such that for $\lambda = (\delta - 7\epsilon\Delta)\epsilon > 0$, at least λn clauses from T are left-split by $t \in \Gamma_\epsilon$ according to $\mathcal{I}$. Call this set of λn clauses U.

By Lemma 4.6.4, for some constant λ' we can find a set of $\lambda' n$ disjoint clauses $U' \subseteq U$. Now we invoke Lemma 4.5.11 to show that there exists an SDR σ for T such that at most $1.6\lambda' n$ of the $3\lambda' n$ variables in $var(U')$ are in the range of σ. To do this it suffices to show that for any set of clauses $S' \subseteq T$, $|S'| - |var(S') \setminus var(U')| \leq 1.6\lambda' n$. If $|S'| \leq 1.6\lambda' n$ then trivially the inequality is satisfied. Otherwise, if $|S'| > 1.6\lambda' n$, then because ψ is a $(\delta', 1.9)$ expander, $|var(S')| \geq 1.9|S'|$, so

$$|S'| - |var(S') \backslash var(U')| \leq -0.9|S'| + 3\lambda' n \leq -1.44\lambda' n + 3\lambda' n \leq 1.6\lambda' n$$

Because there are at most $1.6\lambda' n$ variables from $var(U')$ in the range of σ, there must exist a set of clauses $S \subseteq U'$, with $|S| = 0.4\lambda' n$, such that for every clause $C \in S$, exactly one variable in C is in the range of σ. This set S satisfies the requirements of the lemma. $\square$

We are now ready to prove Lemma 4.6.2.

Proof of Lemma 4.6.2. By Lemma 4.5.9, there exists $\delta > 0$ such that with high probability ϕ is a $(\delta, 1.9)$ expander, and also with high probability the conclusion of Lemma 4.6.7 holds.

Let S be a subset of the clauses of ϕ such that $\delta n/2 \leq |S| \leq \delta n$. Let $S' \subseteq S$ be the set guaranteed to exist by Lemma 4.6.7 with the four properties from that lemma, and σ the corresponding SDR for S. Let $\text{left}_{S'} = \{\text{left}_{C,t,\mathcal{I}} : C \in S'\}$.

In order to prove the lemma we will make use of Corollary 4.5.4 to show that $|OBDD_{\mathcal{I}}(f_S)| \geq 2^{|\text{left}_{S'}|} \geq 2^{\epsilon n}$ for some constant $\epsilon > 0$. Our set B from that theorem will be $\text{left}_{S'}$. Define $\mathbf{z} \in \{0,1\}^t$ as follows. For each $1 \leq i \leq t$ such that $x_i = \sigma(C)$ for some clause C, let $\mathbf{z}_i$ be the value that satisfies the clause C. Assign all other values of $\mathbf{z}$ arbitrarily.

To finish the proof of the lemma, we need that for all $\mathbf{x}, \mathbf{x}' \in \{0,1\}^t$, if $\mathbf{x} \neq \mathbf{x}'$ and $\mathbf{x}_i = \mathbf{x}'_i = \mathbf{z}_i$ for all $i \notin B$, then there exists $\mathbf{y} \in \{0,1\}^{n-t}$ such that $\phi(\mathbf{x}, \mathbf{y}) \neq \phi(\mathbf{x}', \mathbf{y})$.

Let $\mathbf{x}, \mathbf{x}' \in \{0,1\}^t$ such that $\mathbf{x} \neq \mathbf{x}'$ and $\mathbf{x}_i = \mathbf{x}'_i = \mathbf{z}_i$ for all $i \notin B$. Let j be an index such that $\mathbf{x}_j \neq \mathbf{x}'_j$. Let C be the clause from S' such that $\mathbf{x}_j = \text{left}_{C,t,\mathcal{I}}$.

Define $\mathbf{y}$ as follows. Let p and q be the two indices other than j such that $\mathbf{x}_p$ and $\mathbf{x}_q$ are in the clause C. Define $\mathbf{y}_{p-t}$ and $\mathbf{y}_{q-t}$ each to be the value that does not satisfy the clause C. For each clause $D \neq C$ such that $D \in S'$, let r and s be the two indices greater than t such that $\mathbf{x}_r$ and $\mathbf{x}_s$ are in the clause D. Define $\mathbf{y}_{r-t}$ and $\mathbf{y}_{s-t}$ each to be the value that satisfies D. For any index i such that $t < i \leq n$ and $\mathbf{x}_i = \sigma(E)$ for some clause E other than C, define $\mathbf{y}_{i-t}$ to be the value that satisfies the clause E. Assign all other values of $\mathbf{y}$ arbitrarily. Note that because the clauses in S' are disjoint and for each clause C in S' exactly one of the variables of C is in the range of σ, it is always possible to form the partial assignment $\mathbf{y}$ according to these rules. (Note in particular that if $\sigma(E) = x$ for some clause $E \notin S'$, then $x \notin var(S')$).

Either $\mathbf{x}_j$ satisfies the clause C, or $\mathbf{x}'_j$ does. Assume without loss of generality that $\mathbf{x}_j$ does. Then $\phi(\mathbf{x}', \mathbf{y}) = 0$, since the assignment $(\mathbf{x}', \mathbf{y})$ does not satisfy the clause C. However, $\phi(\mathbf{x}, \mathbf{y}) = 1$, since the assignment $(\mathbf{x}, \mathbf{y})$ satisfies every clause in ϕ. This completes the proof. □

4.7 Proof of the OBDD+ Case

The purpose of this section is to prove Theorem 4.4.2, which we now restate.

Theorem 4.7.1 (Restatement of Theorem 4.4.2)**.** *Let $\Delta > 0.91$. There exists a constant $\epsilon > 0$ such that, with high probability when ϕ is a random 3-XOR formula on n variables with clause density Δ, ϕ is unsatisfiable and $S^{+}(\phi) \geq 2^{\epsilon n}$.*

The following three lemmas are needed in the proof.

Lemma 4.7.2. *For $0 < \epsilon < 1$, let $\Gamma_\epsilon = \{\lceil \epsilon n \rceil, \lceil 2\epsilon n \rceil, \lceil 3\epsilon n \rceil, \ldots, \lceil (1-\epsilon)n \rceil\}$. Let $\Delta > 0.5$. There exists $\epsilon, \delta > 0$ such that, with high probability when ϕ is a random 3-formula on n variables with clause density Δ, the following property holds: For* any *order π, there exists some $t_\pi \in \Gamma_\epsilon$ such that more than δn of the clauses from ϕ are split by t_π according to π.* [5]

Proof. By Lemma 4.5.9, for all $y < 2$, there exists an $x > 0$ such that with high probability ϕ will be an (x, y) expander. We will pick some $y < 2, \epsilon > 0$, and $\delta > 0$ to be determined later in the proof.

Suppose for contradiction that there exists a π such that no $t \in \Gamma_\epsilon$ splits more than δn of the clauses from ϕ. Then there exists $S \subseteq \phi$, with $|S| \geq \Delta n - \frac{\delta n}{\epsilon}$, such that for each clause $C \in S$, all the variables from C are contained within an interval of Γ_ϵ: i.e. there exists $t \in \{0\} \cup \Gamma_\epsilon$ such that for all $x_i \in C$, $\pi^{-1}[i] > t$ and $\pi^{-1}[i] < t + \epsilon n$. In this case we will say that the clause C is contained in the interval t according to π.

By an averaging argument there exists some $t \in \{0\} \cup \Gamma_\epsilon$ and $T \subseteq S$ with $|T| \geq \epsilon(\Delta n - \frac{\delta n}{\epsilon}) = \epsilon \Delta n - \delta n$ such that every clause from T is contained in the interval t.

Suppose that we choose y, ϵ, and δ in such a way that

1. $y < 2$

[5] *In fact, using a slightly more complicated first moment argument, one can prove the stronger statement that this lemma holds even if we fix $t_\pi = n/2$.*

2. $y(\epsilon\Delta n - \delta n) > \epsilon n$.

3. $xn \geq \epsilon\Delta n - \delta n$

Then this would be a contradiction, because it would imply by the expansion properties of ϕ that not all of the clauses from T can in fact be contained in the interval t.

All that remains is to show that we can satisfy these inequalities simultaneously.

In order to simultaneously satisfy the first two inequalities, we must be able to satisfy the following inequality:

$$2(\epsilon\Delta n - \delta n) \geq \epsilon n$$

Solving for δ, we get that

$$\begin{aligned}
&2\epsilon\Delta n - 2\delta n \geq \epsilon n \\
&2\delta n \leq 2\epsilon\Delta n - \epsilon n \\
&\delta \leq \epsilon\Delta - \frac{\epsilon}{2} \\
&\delta \leq \epsilon(\Delta - \frac{1}{2})
\end{aligned}$$

Note that because $\Delta > \frac{1}{2}$, δ can satisfy this inequality and still be positive.

A stronger condition than the third inequality is that $\epsilon \leq \frac{x}{\Delta}$. Thus by choosing y to be sufficiently close to 2, then choosing ϵ and δ as shown above we get our contradiction and the lemma is proved. □

Lemma 4.7.3. *There exists $\lambda > 0$ such that, with high probability when ϕ is a random 3-XOR formula with clause density $\Delta = 0.6$, ϕ is a $(0.6, 1 + \lambda)$ expander.*

Proof. This type of calculation is by now standard (see for instance [BSG03], Lemma 5.1), although in our case it is slightly messier than usual because of the more specific bounds that we need.

Let $\delta = 0.01$ and $\Delta = 0.6$. Let A_i denote the event that a set of clauses S of size i has expansion

$$\frac{|var(S)|}{|S|} < 1 + \delta,$$

where $i = 1, 2, \cdots, \Delta n$. There are $\binom{\Delta n}{i}$ such sets of clauses and $\binom{n}{(1+\delta)i}$ possible small vertex sets. The probability for a single edge to fall within the small vertex set is $\binom{(1+\delta)i}{3}/\binom{n}{3} \le \left(\frac{(1+\delta)i}{n}\right)^3$. Thus

$$\Pr[A_i] \le \binom{\Delta n}{i}\binom{n}{(1+\delta)i}\left(\frac{(1+\delta)i}{n}\right)^{3i}.$$

We need to bound the probability $\Pr[A_i]$ in order to show that

$$\Pr[\bigvee_{i=1}^{\Delta n} A_i] = o(1).$$

Since $\Pr[\bigvee_{i=1}^{\Delta n} A_i] \le \Pr[\bigvee_{i=1}^{\sqrt[3]{n}} A_i] + \Pr[\bigvee_{i=\sqrt[3]{n}+1}^{cn} A_i] + \Pr[\bigvee_{i=0.1n+1}^{\Delta n} A_i]$, where c is a constant between 0 and Δ to be determined later, it is enough to show that

$$\Pr[\bigvee_{i=1}^{\sqrt[3]{n}} A_i] = n^{-\Omega(1)} = o(1) \tag{4.1}$$

$$\Pr[\bigvee_{i=\sqrt[3]{n}+1}^{cn} A_i] = n^{-\Omega(\sqrt[3]{n})} = o(1) \tag{4.2}$$

$$\Pr[\bigvee_{i=cn+1}^{\Delta n} A_i] = 2^{-\Omega(n)} = o(1) \tag{4.3}$$

Using the estimation $\binom{a}{b} \le (\frac{ea}{b})^b$, we get

$$\begin{aligned}\Pr[A_i] &\le \left(\frac{e\Delta n}{i}\right)^i\left(\frac{en}{(1+\delta)i}\right)^{(1+\delta)i}\left(\frac{(1+\delta)i}{n}\right)^{3i} \\ &= \left[\frac{e^{2+\delta}\cdot\Delta\cdot(1+\delta)^{2-\delta}\cdot i^{1-\delta}}{n^{1-\delta}}\right]^i.\end{aligned}$$

Thus

$$\begin{aligned}\Pr[\bigvee_{i=1}^{\sqrt[3]{n}} A_i] &\le \sqrt[3]{n}\cdot\frac{e^{2+\delta}\cdot\Delta\cdot(1+\delta)^{2-\delta}\cdot\sqrt[3]{n}^{1-\delta}}{n^{1-\delta}} \\ &= e^{2+\delta}\cdot\Delta\cdot(1+\delta)^{2-\delta}\cdot n^{\frac{2\delta-1}{3}}\end{aligned}$$

which tends to 0 as n tends to infinity. This implies (4.1).

In order to show (4.2) and (4.3), we need to use a better approximation for $\Pr[A_i]$:

$$\begin{aligned}\Pr[A_i] &\le \binom{\Delta n}{i}\binom{n}{(1+\delta)i}\left(\frac{(1+\delta)i}{n}\right)^{3i} \\ &= \frac{(\Delta n)!}{(\Delta n-i)!\cdot i!}\cdot\frac{n!}{(n-(1+\delta)i)!\cdot((1+\delta)i)!}\cdot\left(\frac{(1+\delta)i}{n}\right)^{3i}\end{aligned}$$

Using Stirling's approximation,

$$\sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n} \le n! \le e n^{n+\frac{1}{2}} e^{-n}$$

we get

$$\begin{aligned}
\Pr[A_i] &\le \frac{e(\Delta n)^{\Delta n+\frac{1}{2}} e^{-\Delta n}}{\sqrt{2\pi}(\Delta n - i)^{\Delta n - i + \frac{1}{2}} e^{-(\Delta n - i)} \sqrt{2\pi} i^{i+\frac{1}{2}} e^{-i}} \\
&\cdot \frac{e n^{n+\frac{1}{2}} e^{-n}}{\sqrt{2\pi}(n-(1+\delta)i)^{n-(1+\delta)i+\frac{1}{2}} e^{-(n-(1+\delta)i)} \sqrt{2\pi}((1+\delta)i)^{(1+\delta)i+\frac{1}{2}} e^{-((1+\delta)i)}} \cdot \left(\frac{(1+\delta)i}{n}\right)^{3i} \\
&\le \frac{(\Delta n)^{\Delta n+\frac{1}{2}}}{(\Delta n - i)^{\Delta n - i + \frac{1}{2}} \cdot i^{i+\frac{1}{2}}} \cdot \frac{n^{n+\frac{1}{2}}}{(n-(1+\delta)i)^{n-(1+\delta)i+\frac{1}{2}} \cdot ((1+\delta)i)^{(1+\delta)i+\frac{1}{2}}} \cdot \left(\frac{(1+\delta)i}{n}\right)^{3i} = f(i).
\end{aligned}$$

Then

$$\begin{aligned}
\ln f(i) &= (\Delta n + \frac{1}{2}) \ln(\Delta n) - (\Delta n - i + \frac{1}{2}) \ln(\Delta n - i) - (i + \frac{1}{2}) \ln i \\
&+ (n + \frac{1}{2}) \ln n - (n - (1+\delta)i + \frac{1}{2}) \ln(n - (1+\delta)i) - ((1+\delta)i + \frac{1}{2}) \ln((1+\delta)i) \\
&+ 3i(\ln((1+\delta)i) - \ln n) = g(i), \\
\frac{\mathtt{d}g(i)}{\mathtt{d}i} &= \frac{\Delta n - i + 1/2}{\Delta n - i} + \ln(\Delta n - i) - \frac{i + 1/2}{i} - \ln i + \frac{(1+\delta)(n - (1+\delta)i + 1/2)}{n - (1+\delta)i} \\
&+ (1+\delta) \ln(n - (1+\delta)i) - \frac{(1+\delta)i + 1/2}{i} - (1+\delta) \ln((1+\delta)i) + 3(\ln((1+\delta)i) - \ln n) + 3 \\
&= \ln(\Delta n - i) + (1-\delta) \ln i + (1+\delta) \ln(n - (1+\delta)i) - 3 \ln n \\
&+ \frac{1}{2(\Delta n - i)} - \frac{1}{i} + \frac{1+\delta}{2(n - (1+\delta)i)} + (2-\delta) \ln(1+\delta) + 3 \\
&= \ln \frac{(\Delta n - i) i^{1-\delta} (n - (1+\delta)i)^{1+\delta}}{n^3} + \frac{1}{2(\Delta n - i)} - \frac{1}{i} + \frac{1+\delta}{2(n - (1+\delta)i)} + (2-\delta) \ln(1+\delta) + 3.
\end{aligned}$$

When $i \in [\sqrt[3]{n}, cn]$, for large enough n, we have

$$\begin{aligned}
\frac{\mathtt{d}g(i)}{\mathtt{d}i} &\le \ln \frac{(\Delta n - i) i^{1-\delta} (n - (1+\delta)i)^{1+\delta}}{n^3} + 4 \\
&\le \ln \frac{(\Delta n)(cn)^{1-\delta} n^{1+\delta}}{n^3} + 4 \\
&= \ln(\Delta c^{1-\delta}) + 4.
\end{aligned}$$

In order to make $\ln(\Delta c^{1-\delta}) + 4 < 0$, it is enough to have $c = 0.01$. This means $g(i)$ is decreasing in interval $[\sqrt[3]{n}, cn]$, which implies $f(i)$ is also decreasing in the same interval.

Thus

$$\begin{aligned}
\Pr[\bigvee_{i=\sqrt[3]{n}+1}^{cn} A_i] &\leq (cn-\sqrt[3]{n})f(\sqrt[3]{n}) \\
&\leq cn\cdot\frac{(\Delta n)^{\Delta n+\frac{1}{2}}}{(\Delta n-\sqrt[3]{n})^{\Delta n-\sqrt[3]{n}+\frac{1}{2}}\cdot\sqrt[3]{n}^{\sqrt[3]{n}+\frac{1}{2}}} \\
&\quad\cdot\frac{n^{n+\frac{1}{2}}}{(n-(1+\delta)\sqrt[3]{n})^{n-(1+\delta)\sqrt[3]{n}+\frac{1}{2}}\cdot((1+\delta)\sqrt[3]{n})^{(1+\delta)\sqrt[3]{n}+\frac{1}{2}}}\cdot\left(\frac{(1+\delta)\sqrt[3]{n}}{n}\right)^{3\sqrt[3]{n}} \\
&\leq cn\cdot\left(\frac{\Delta n}{\Delta n-\sqrt[3]{n}}\right)^{\Delta n+\frac{1}{2}}\cdot(\Delta n)^{\sqrt[3]{n}}\cdot n^{-\sqrt[3]{n}/3-1/6} \\
&\quad\cdot\left(\frac{n}{n-(1+\delta)\sqrt[3]{n}}\right)^{n+\frac{1}{2}}\cdot n^{(1+\delta)\sqrt[3]{n}}\cdot(1+\delta)^{(2-\delta)\sqrt[3]{n}-1/2}\cdot n^{-(7+\delta)\sqrt[3]{n}/3-1/6} \\
&= \frac{cn\cdot\left(\frac{\Delta n}{\Delta n-\sqrt[3]{n}}\right)^{\Delta n+\frac{1}{2}}\cdot\Delta^{\sqrt[3]{n}}\cdot\left(\frac{n}{n-(1+\delta)\sqrt[3]{n}}\right)^{n+\frac{1}{2}}\cdot(1+\delta)^{(2-\delta)\sqrt[3]{n}-1/2}}{n^{(2-2\delta)\sqrt[3]{n}/3+1/3}} \\
&= o(1)
\end{aligned}$$

which proves (4.2).

In order to show (4.3), for the range $cn \leq i \leq \Delta n$, let $i = tn$ where $c \leq t \leq \Delta$. In this case,

$$\begin{aligned}
f(i)=f(tn) &= \frac{(\Delta n)^{\Delta n+\frac{1}{2}}}{(\Delta n-tn)^{\Delta n-tn+\frac{1}{2}}\cdot(tn)^{tn+\frac{1}{2}}} \\
&\quad\cdot\frac{n^{n+\frac{1}{2}}}{(n-(1+\delta)tn)^{n-(1+\delta)tn+\frac{1}{2}}\cdot((1+\delta)tn)^{(1+\delta)tn+\frac{1}{2}}}\cdot\left(\frac{(1+\delta)tn}{n}\right)^{3tn} \\
&= \frac{\sqrt{\Delta}}{nt\sqrt{(\Delta-t)\cdot(1-(1+\delta)t)\cdot(1+\delta)}}\cdot\left[\frac{\Delta^{\Delta}\cdot(1+\delta)^{(2-\delta)t}\cdot t^{(1-\delta)t}}{(\Delta-t)^{(\Delta-t)}\cdot(1-(1+\delta)t)^{1-(1+\delta)t}}\right]^n
\end{aligned}$$

Define

$$h(t)=\frac{\Delta^{\Delta}\cdot(1+\delta)^{(2-\delta)t}\cdot t^{(1-\delta)t}}{(\Delta-t)^{(\Delta-t)}\cdot(1-(1+\delta)t)^{1-(1+\delta)t}}.$$

Using Mathematica, one can show that there exists some constant $0 < \gamma < 0.970$ such that $h(t) \leq \gamma$ holds when $c \leq t \leq \Delta$. Therefore

$$\begin{aligned}
\Pr[\bigvee_{i=cn+1}^{\Delta n} A_i] &\leq \Pr[\bigvee_{i=cn+1}^{\Delta n-1} A_i]+\Pr[A_{\Delta n}] \\
&\leq \frac{(\Delta-c)\sqrt{\Delta n}}{c\sqrt{(1-(1+\delta)\Delta)\cdot(1+\delta)}}\cdot\gamma^n+\binom{n}{(1+\delta)\Delta n}\left(\frac{(1+\delta)\Delta n}{n}\right)^{3\Delta n} \\
&\leq \frac{(\Delta-c)\sqrt{\Delta n}}{c\sqrt{(1-(1+\delta)\Delta)\cdot(1+\delta)}}\cdot\gamma^n+\left(\frac{e^{1-(1+\delta)\Delta}((1+\delta)\Delta)^{3\Delta}}{(1-(1+\delta)\Delta)^{1-(1+\delta)\Delta}}\right)^n=o(1)
\end{aligned}$$

which shows (4.3). □

Lemma 4.7.4. *Let ψ be a 3-formula over n variables with clause density Δ such that ψ is a $(\Delta, 1+\delta)$-expander for some $\delta > 0$. Let $U \subseteq \psi$ be a set of disjoint clauses with $|U| = \lambda n$ for some $\lambda > 0$. Let Ψ be a set of variables and f be a bijection from Ψ to U such that for all $x \in \Psi$, x appears in the clause $f(x)$. Then there exists $\epsilon > 0$ such that there is an SDR σ on ψ for which at least ϵn of the variables from Ψ are not in the range of σ.*

Proof. Let $\epsilon = \lambda\delta$. By Lemma 4.5.11, we need to show that for any $S' \subseteq \psi$, $|S'| - |var(S')\backslash\Psi| \leq (\lambda - \epsilon)n$.

By the expansion of ψ and the fact that there is exactly one variable from Ψ in each clause from U, we have that $|var(S')\backslash\Psi| \geq (1+\delta)|S'| - \min(|S'|, \lambda n)$.

First suppose that $|S'| \geq \lambda n$. Then we have that

$$\begin{aligned}
|S'| - |var(S')\backslash\Psi| &\leq |S'| - ((1+\delta)|S'| - \lambda n) \\
&= \lambda n - \delta|S'| \\
&\leq \lambda n - \delta\lambda n \\
&= \lambda n(1-\delta) \\
&= (\lambda - \epsilon)n
\end{aligned}$$

Now suppose that $|S'| < \lambda n$. Then we have that

$$\begin{aligned}
|S'| - |var(S')\backslash\Psi| &\leq |S'| - ((1+\delta)|S'| - |S'|) \\
&= (1-\delta)|S'| \\
&\leq \lambda n(1-\delta) \\
&= (\lambda - \epsilon)n
\end{aligned}$$

Thus in either case $|S'| - |var(S')\backslash\Psi| \leq (\lambda - \epsilon)n$

□

Proof of Theorem 4.7.1. Let $\Delta > 0.91$, and let $\phi = (C_1, C_2, \ldots, C_{\Delta n})$ be a random 3-XOR formula on n variables with clause density Δ.

By Theorem 4.5.8, with high probability ϕ will be unsatisfiable.

Let $\psi = (C_1, C_2, \ldots, C_{0.6n})$ be the 3-XOR formula that is the first $0.6n$ clauses of ϕ. We will show that there exists $\epsilon > 0$ such that with high probability $\text{OBDD}_\pi(\psi) \geq 2^{\epsilon n}$ for *any* order π. This implies that $S^+(\phi) \geq 2^{\epsilon n}$, proving the theorem.

Suppose the conclusions of Lemma 4.7.2, Lemma 4.7.3, and Lemma 4.6.4 hold with respect to ψ (which occurs with high probability).

By Lemmas 4.7.2 and 4.7.3, there exists a δ such that ψ is a $(0.6, 1+\delta)$-expander, and for any order π there exists t_π such that more than δn clauses from ψ are split by t_π according to π. For a given π, let S be a set of δn clauses from ψ such that every clause in S is split by t_π according to π.

By Lemma 4.6.4, there exists $\gamma > 0$ and $T \subseteq S$, with $|T| = \gamma n$, such that the clauses of T are disjoint. Either $\frac{\gamma n}{2}$ of the clauses from T are left-split by t_π, or at least $\frac{\gamma n}{2}$ of the clauses from T are right-split by t_π. We now divide our proof into two cases depending on which of these is true.

First assume that there exists $U \subseteq T$, with $|U| \geq \frac{\gamma n}{2}$ such that every clause in U is left-split by t_π. By Lemma 4.7.4, there exists $\epsilon > 0$ and $V \subseteq U$, such that

1. $|V| = \epsilon n$
2. There exists an SDR σ on ψ such that no element of left_V is in the range of σ, where $\text{left}_V = var(V) \cap \text{left}_U$

(Here U and left_U correspond to U and Ψ from Lemma 4.7.4 respectively.)

Due to the properties of σ, any assignment of the variables of left_V can be extended to satisfy the formula ψ: Define a function $f : \{0,1\}^{|V|} \to \{0,1\}^n$ in such a way that $f(\mathbf{x}) = (\mathbf{x}, \mathbf{y}, \mathbf{z})$, where

1. $\mathbf{x}$ is an assignment to the variables of left_V.
2. $\mathbf{y}$ is an assignment to the variables $var_{t_\pi,\pi} \backslash \text{left}_V$, where $var_{t_\pi,\pi}$ are the first t_π variables of ψ according to the order π.
3. $\mathbf{z}$ is an assignment to $var(\psi) \backslash var_{t_\pi,\pi}$ (i.e. the last $n - t_\pi + 1$ variables according to the order π.)

4. $\psi(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 1$

For $\mathbf{x}, \mathbf{x}' \in \{0,1\}^{|V|}$ with $\mathbf{x} \neq \mathbf{x}'$, let $f(\mathbf{x}) = (\mathbf{x}, \mathbf{y}, \mathbf{z})$ and $f(\mathbf{x}') = (\mathbf{x}', \mathbf{y}', \mathbf{z}')$. We will show that $\psi(\mathbf{x}, \mathbf{y}, \mathbf{z}) \neq \psi(\mathbf{x}', \mathbf{y}', \mathbf{z})$. This implies that $|\{\psi|_{\pi,\mathbf{w}} : \mathbf{w} \in \{0,1\}^{t_\pi}\}| \geq 2^{|V|}$, which by Theorem 4.5.3 implies that $|\text{OBDD}_\pi| \geq 2^{|V|} = 2^{\epsilon n}$.

By definition we have that $\psi(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 1$. $\mathbf{x}$ and $\mathbf{x}'$ differ on their assignments to some variable in left_V. Let $C \in V$ be the clause containing the variable in left_V on which $\mathbf{x}$ and $\mathbf{x}'$ differ. The other two variables in C are assigned equally in $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and $(\mathbf{x}', \mathbf{y}', \mathbf{z})$, since their values are determined by the assignment $\mathbf{z}$. Therefore C is not satisfied as a 3-XOR clause by $(\mathbf{x}', \mathbf{y}', \mathbf{z})$, and $\psi(\mathbf{x}, \mathbf{y}, \mathbf{z}) \neq \psi(\mathbf{x}', \mathbf{y}', \mathbf{z})$.

Now we consider the second case, where there exists $U \subseteq T$, with $|U| \geq \frac{\gamma n}{2}$ such that every clause in U is right-split by t_π.

By Lemma 4.7.4, there exists $\epsilon > 0$ and $V \subseteq U$, such that

1. $|V| = \epsilon n$

2. There exists an SDR σ on ψ such that no element of right_V is in the range of σ, where $\text{right}_V = var(V) \cap \text{right}_U$

Due to the properties of σ, any assignment of the variables of right_V can be extended to satisfy the formula ψ: Define a function $f : \{0,1\}^{|V|} \to \{0,1\}^n$ in such a way that: $f(\mathbf{z}) = (\mathbf{x}, \mathbf{y}, \mathbf{z})$, where

1. $\mathbf{z}$ is an assignment to the variables of right_V.

2. $\mathbf{y}$ is an assignment to the variables $var(\psi) \backslash (var_{t_\pi,\pi} \cup \text{right}_V)$ (i.e. the last $n - t_\pi + 1$ variables according to the order π, not including variables from right_V).

3. $\mathbf{x}$ is an assignment to the variables of $var_{t_\pi,\pi}$.

4. $\psi(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 1$

For $\mathbf{z}, \mathbf{z}' \in \{0,1\}^{|V|}$ with $\mathbf{z} \neq \mathbf{z}'$, let $f(\mathbf{z}) = (\mathbf{x}, \mathbf{y}, \mathbf{z})$ and $f(\mathbf{z}') = (\mathbf{x}', \mathbf{y}', \mathbf{z}')$. We will show that $\psi(\mathbf{x}, \mathbf{y}, \mathbf{z}) \neq \psi(\mathbf{x}', \mathbf{y}, \mathbf{z})$. This implies that $|\{\psi|_{\pi,\mathbf{w}} : \mathbf{w} \in \{0,1\}^{t_\pi}\}| \geq 2^{|V|}$, which by Theorem 4.5.3 implies that $|\text{OBDD}_\pi| \geq 2^{|V|} = 2^{\epsilon n}$

By definition we have that $\psi(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 1$. $\mathbf{z}$ and $\mathbf{z}'$ differ on their assignments to some variable in right_V. Let $C \in V$ be the clause containing the variable in right_V on which $\mathbf{z}$ and $\mathbf{z}'$ differ. By definition $\psi(\mathbf{x}', \mathbf{y}', \mathbf{z}') = 1$, so C is satisfied as a 3-XOR clause by the assignment $(\mathbf{x}', \mathbf{y}', \mathbf{z}')$. The only variable in C that is assigned by $\mathbf{y}'$ or $\mathbf{z}'$ is the variable right_C, so the clause C is not satisfied as a 3-XOR clause by $(\mathbf{x}', \mathbf{y}, \mathbf{z})$, and therefore $\psi(\mathbf{x}, \mathbf{y}, \mathbf{z}) \neq \psi(\mathbf{x}', \mathbf{y}, \mathbf{z})$.

$\square$

4.8 Future Work

The obvious open problem is to prove lower bounds for refuting random 3-CNF or 3-XOR formulas in an OBDD-based refutation system where neither the variable order nor the order in which clauses are processed in the refutation is constrained. Although it might seem that one could tweak our techniques to get this result, it may be that this is more difficult than appears at first glance.

For instance, suppose we tried to use the same approach of focusing in on a particular OBDD in the refutation of a random 3-CNF formula such that the OBDD represents the conjunction of about δn clauses, for some appropriately chosen fixed δ, in the hopes of showing that the OBDD must be of exponential size. In the restricted systems from this chapter, we were able to choose δ to be an arbitrarily small constant. However, in the unrestricted system (still without weakening), we would be forced to choose δ to be greater than about $1/6$, because a random formula with clause density just above the threshold *does* contain sub-formulas with about $n/6$ clauses that have small OBDD representations for some variable order. (For instance, one can look for a large set of disjoint clauses and then choose a variable order where the variables from each clause are adjacent in the order). It is much more difficult to reason about sub-formulas in this regime; for instance, to the best of our knowledge it has not even been proved that a random 3-CNF formula with clause density just above the threshold with high probability does not contain an unsatisfiable sub-formula consisting of $n/6$ clauses (let alone that all sub-formulas slightly larger than this must have large OBDD representations).

Certainly one cannot rely on the existence of SDRs when considering sub-formulas with clause density this close to the threshold.

Making progress will probably require using a more sophisticated analysis of the structure of random formulas than we do in this chapter. A large amount of research has been spent investigating the structure of random CNF and XOR formulas with densities below the respective satisfiability thresholds, including understanding the solution space structure of such formulas and the occurrence of various phase transitions. (See for example [Ach09] for a survey of this work, along with more general information about SAT solving and random formulas). Finding a way to leverage this type of knowledge in this context is probably a key step towards achieving these more difficult lower bounds.

Chapter 5

A General Framework for Proving Proof Complexity Lower Bounds Using Random Formulas

In the previous chapter we focused on variants of a particular OBDD-based proof system and proved lower bounds for this proof system using random formulas. However, the high-level approach was not really exclusively geared towards random formulas and the methods used were specifically tailored to work with OBDD-based systems in a way that was not obviously generalizable. As mentioned in Section 4.2, it would be good to have methods that were adapted specially to work with random CNF formulas and that potentially could be used to attack strong proof systems.

In this chapter we introduce such a general framework for working with random 3-CNF formulas. Conceptually, the high-level approach is simple: The goal is to show that given a short refutation of an unsatisfiable formula in a proof system P, the description of the formula can be compressed more than one can compress a random formula, by algorithmically constructing a suitable encoding. Doing so implies that random formulas cannot have short refutations in the system P. In essence we are using a sophisticated counting argument that is specifically tailored to work with random formulas.

After introducing the general framework, in Sections 5.3 and 5.4 we demonstrate its use by proving lower bounds for treelike resolution, a restricted version of the resolution system. We should emphasize from the outset that this result itself is not new; exponential lower bounds for random k-CNFs are already known for treelike resolution, and in fact for much stronger systems such as general resolution and the RES(k) system.[1]

[1]RES(k) is a generalization of resolution whose lines are k-DNFs instead of just clauses [BSW01, Ale05]. A clause is a 1-DNF and resolution is equivalent to RES(1). RES(k) for constant $k > 1$ is strictly stronger than resolution and strictly weaker than constant-depth Frege systems.

Furthermore, the bounds we derive are not optimal – for the clause density we choose the best known bounds are of the form $2^{\Omega(n)}$, and we derive bounds of 2^{n^ϵ} for some ϵ.[2]

However, it is our hope that the techniques introduced generalize better than current methods and could be useful for attacking stronger proof systems for which no good methods currently exist. Wigderson and Ben-Sasson showed that most of the known resolution lower bounds, including those for random formulas, can be explained in terms of size-width trade-offs [BSW01]. In this framework, one shows that any short refutation can be transformed to another refutation where the width of all clauses in the new refutation (defined as the number of literals in the clause) is small. Then one derives lower bounds on the size of refuting certain formulas by showing that any such refutation must contain a clause of large width. This approach is very elegant, but is hard to generalize since the notion of width is specific to clauses and does not apply to proof systems that have greater expressive power than resolution. The basic framework we establish in this paper can be applied to any proof system, and the lower bounds we derive for treelike resolution do not directly refer to clause width at all.

One other way in which the methods used to prove our treelike resolution lower bounds seem to be qualitatively different than previous methods is that they are algorithmic in nature. At the most abstract level, in order to establish that random formulas do not have short refutations, one must identify a property Q that random formulas have but that formulas with short refutations do not have. The previous resolution lower bounds use static properties Q based on the idea of expansion. Given a CNF formula, we can form a bipartite graph where one set of vertices consists of the clauses of the formula and the other set of vertices consists of the variables of the formula, and there is an edge between a clause and a variable if the variable appears in the clause. For a random formula the resulting bipartite graph will have some type of explicitly stateable expansion properties that formulas with short resolution refutations

[2]Although if instead of choosing constant clause density we consider random formulas with $n^{1.5-\epsilon}$ clauses, in this case the lower bound we can derive using these techniques converges to the optimal value.

provably cannot have. On the other hand, in our lower bound proof the property Q is intrinsically tied to the main coding algorithm of the proof. Our Q also is capturing some idea related to expansion, but it seems impossible to extract it from the main algorithm and to state it explicitly as a static property. Perhaps the ability to use these implicit properties will be useful in dealing with more powerful proof systems.

5.1 Random Formulas and the General Framework

In this section we redefine random formulas in a slightly different way and prove two simple lemmas that establish our general framework applicable to any proof system.

The following notation will be used throughout the chapter. For a clause C, $lits(C)$ denotes the set of literals in C, and $vars(C)$ denotes the set of variables in C, ignoring the sign of the literal. For a k-CNF formula φ, the *size* of φ, denoted by $|\varphi|$, is the number of clauses in φ. We write $\phi \subseteq \varphi$ if ϕ is a subset of the clauses of φ.

Definition 5.1.1 (Random 3-CNF Formulas). *A random 3-CNF formula ϕ on n variables with clause density Δ is a set of Δn clauses, where each clause is chosen uniformly at random without replacement from all of the $2^3\binom{n}{3}$ possible clauses.*

The main difference between this definition of a random formula and the one from the previous chapter is that here we demand that no two clauses of the random formula can be the same. (Here we also view a random formula as a set of clauses as opposed to an ordered list of clauses). For large n and constant clause density, if we choose clauses of a random formula with replacement, with high probability the formula will contain distinct clauses, and therefore for our purposes the two definitions of a random formula coincide. Nonetheless, explicitly including this condition that clauses of a formula are distinct will simplify the proofs that follow.

Our first lemma establishes a simple counting method for proving that random formulas do not have short refutations in a proof system. Recall from Theorem 4.5.7 that $\Delta^* = 4.51$ is an upper bound on the satisfiability threshold for random 3-CNF formulas.

Lemma 5.1.2. *Let P be an arbitrary proof system, and c be a constant such that $c > \Delta^*$. Let Φ be the set of all 3-CNF formulas with at most n variables and exactly cn distinct clauses. Let $\Psi \subseteq \Phi$ be the set of all such formulas that are unsatisfiable and have refutations of size at most $t(n)$ in P. Let $\{0,1\}^{\leq k}$ be the set of all binary strings of length at most k. If there exists an onto function $\eta : \{0,1\}^{\leq k} \rightarrow \Psi$ with*

$$k = \log\left(\binom{8\binom{n}{3}}{cn}\right) - \omega(1)$$

then with high probability a random 3-CNF formula $\mathcal{F}$ with n variables and clause density c will be unsatisfiable and have no P refutation of size at most $t(n)$.

Proof. Because $c > \Delta^*$, with high probability $\mathcal{F}$ will be unsatisfiable.

We have that

$$\log|\Phi| = \log\left(\binom{8\binom{n}{3}}{cn}\right)$$

Therefore, given the assumption, $|\Psi| \ll |\Phi|$, so with high probability $\mathcal{F}$ will be unsatisfiable and have no P refutation of size at most $t(n)$.

□

Lemma 5.1.2 shows that if we can "compress" any formula ψ that has a short refutation, then we can get lower bounds on the size of refutations of random formulas. Our next lemma says that we can focus our attention on finding some subset of the clauses of ψ that are compressible. Suppose we encode a 3-CNF formula with cn clauses by individually encoding each clause (in other words, writing down each of the three variables in the clause and the signs of the literals). Then altogether the encoding would be of size about $3cn \log n$, since writing down a variable takes about $\log n$ bits. Of course, by considering the clauses altogether instead of individually, we can save on our encoding; as it turns out it is possible to encode an arbitrary 3-CNF formula with cn clauses using about $2cn \log n$ bits, which is about $2 \log n$ bits per clause. (This calculation is based on counting the total number of such formulas and taking the logarithm of this number). The following lemma says that if given a formula ψ with a short refutation we are always able to find some subset of $m > 0$ clauses of ψ that can be encoded using $(2-\epsilon)m\log n$ bits, then this suffices to get the lower bounds of Lemma 5.1.2.

Lemma 5.1.3. *Let $P, c, n, \Phi, \Psi, \mathcal{F}$ be as in Lemma 5.1.2.*

Let Φ_m be the set of all sets of m clauses.

Suppose there exists a constant $\epsilon > 0$ and a family of functions $\{\rho_m\}, 1 \leq m \leq cn$, such that

1. *$\rho_m : \{0,1\}^{(2-\epsilon)m\log n} \to \Phi_m$*

2. *For all $\psi \in \Psi$ there exists $\varphi \subseteq \psi$ and ρ_m such that φ is in the range of ρ_m*

Then with high probability $\mathcal{F}$ will have no P refutation of size at most $t(n)$.

Proof. By Lemma 5.1.2, it suffices to construct an onto function $\eta : \{0,1\}^{\leq k} \to \Psi$ with

$$k = \log\left(\binom{8\binom{n}{3}}{cn}\right) - \omega(1)$$

Let $\psi \in \Psi$. By assumption there exists $1 \leq m \leq cn$ and $\varphi \subseteq \psi$ such that $\rho_m(x) = \varphi$ for some $x \in \{0,1\}^{(2-\epsilon)m\log n}$. Let ψ^- be the clauses of ψ that are not in φ. The total number of 3-CNF formulas with at most n variables and exactly $cn - m$ distinct clauses is

$$\binom{8\binom{n}{3}}{cn-m}$$

Therefore ψ^- can be uniquely represented by a binary string y of size

$$\log\left(\binom{8\binom{n}{3}}{cn-m}\right)$$

We will define $\eta(x \circ y) = \psi$, where $\circ$ is the concatenation function. η is well defined; as it turns out, the larger m is the shorter the string $x \circ y$ will be, so the length of the string $x \circ y$ contains the information about where the delimitation between x and y occurs.

Clearly η is onto, so all that remains is to show that

$$|x \circ y| \leq \log\left(\binom{8\binom{n}{3}}{cn}\right) - \omega(1)$$

We have that

$$\begin{aligned}|x \circ y| &= \log\left(\binom{8\binom{n}{3}}{cn-m}\right) + (2-\epsilon)m\log n \\ &= \log\left(\binom{8\binom{n}{3}}{cn-m}\right) + 2m\log n - \epsilon m \log n\end{aligned}$$

Therefore, it suffices to show that

$$\log\left(\binom{8\binom{n}{3}}{cn}\right) - \log\left(\binom{8\binom{n}{3}}{cn-m}\right) \geq 2m\log n - o(m\log n)$$

Let $(n)_k$ denote the falling factorial. We have that

$$\begin{aligned}&\log\left(\binom{8\binom{n}{3}}{cn}\right) - \log\left(\binom{8\binom{n}{3}}{cn-m}\right) \\ &= \log\left(\frac{((4/3)(n)_3)!}{(cn)!((4/3)(n)_3-cn)!}\right) - \log\left(\frac{((4/3)(n)_3)!}{(cn-m)!((4/3)(n)_3-cn+m)!}\right) \\ &= \log((cn-m)!) + \log(((4/3)(n)_3-cn+m)!) - \log((cn)!) - \log(((4/3)(n)_3-cn)!) \\ &= -\sum_{i=cn-m+1}^{cn}\log i + \sum_{i=(4/3)(n)_3-cn+1}^{(4/3)(n)_3-cn+m}\log i \\ &\geq -m\log(cn) + m\log((4/3)(n)_3-cn+1) \\ &= -m\log n + 3m\log n - o(m\log n) \\ &= 2m\log n - o(m\log n)\end{aligned}$$

□

5.2 Resolution and the DPLL system

In this section we formally define the resolution system and the DPLL system, which corresponds to treelike resolution.

A resolution refutation of an unsatisfiable CNF formula φ is a sequence of clauses $\mathcal{C} = C_1, C_2, \dots C_t$, where $C_t = \emptyset$, the empty clause, and each C_i is either a clause from φ or is derived from two clauses C_j, C_k, with $j,k < i$, using the following resolution rule

$$\frac{A \vee x^1, B \vee x^0}{A \vee B}$$

Here $C_j = A \vee x^1$, $C_k = B \vee x^0$, $C_i = A \vee B$, A and B are arbitrary sets of literals, and x is an arbitrary Boolean variable. Because the resolution rule is sound and the empty clause is trivially unsatisfiable, such a sequence of clauses is a proof that φ is unsatisfiable. One can also show that resolution is complete; any unsatisfiable 3-CNF formula has a resolution refutation of size at most 2^n, where the size of a resolution refutation is defined to be the number of clauses in the sequence.

It is natural to represent a resolution refutation π of a formula φ as a rooted directed acyclic graph π with nodes in π corresponding to the clauses in $\mathcal{C}$. In this representation, the root of π is the clause $\emptyset$, the leaves of π are clauses of φ, and there are edges going from C_j and C_k into C_i if C_i was derived from C_j and C_k. (The graph π cannot necessarily be uniquely determined from $\mathcal{C}$, since it is possible that a clause from $\mathcal{C}$ can be legally derived in multiple ways). If there exists a π corresponding to $\mathcal{C}$ such that the outdegree of every node in π other than the root is one, then we say that $\mathcal{C}$ is a *treelike* refutation. In some cases the smallest treelike resolution refutation of a formula φ can be exponentially larger than the smallest general resolution refutation of φ [BSIW04].

Tseitsin proved the first super-polynomial resolution lower bounds for a restricted form of resolution called regular resolution (a system strictly more powerful than tree-like resolution) in the late 1960's, using a class of contradictions based on the fact that the sums of degrees of an undirected graph must be even [Tse68]. Almost two decades passed before Haken proved the first super-polynomial bounds for general resolution, using a class of contradictions based on the pigeonhole principle [Hak85]. Soon after, Szémeredi and Chvátal proved the first resolution lower bounds for random CNFs using techniques adapted from Haken's proof [CS88]. Later, Ben-Sasson and Wigderson unified all of these previous results under the size-width trade-off framework [BSW01]. The best lower bounds for random 3-CNFs with n variables and clause density $c > \Delta^*$ are of the form $2^{\Omega(n)}$, which is tight up to the multiplicative factor in the exponent.

We now describe the DPLL proof system:

Definition 5.2.1 (DPLL system)**.** *A DPLL refutation π of an unsatisfiable 3-CNF formula φ is a binary rooted tree. Each interior node of π is labeled with a variable x, and the two edges entering such a node are respectively labeled with x^0 and x^1. For a* *given node a in π, we associate with a the set $\rho(a)$ that consists of the literals labeling the path from the root to a. Every leaf node b is labeled with a clause from φ whose literals are a subset of $\rho(b)$.*[3] *The size of π is the number of nodes in π.*

We will make a couple of assumptions about the structure of a DPPL refutation.

Let π be a DPLL refutation of a formula φ. For a node a in π, let *tree(a)* be the subtree of π with root a.

Suppose an interior node a is labeled with the variable x. Then $\mathit{child}_1(a)$ denotes the node in π connected to a by an edge labeled with x^1, and $\mathit{child}_0(a)$ denotes the node in π connected to a by an edge labeled with x^0. Also, $\mathit{tree}_1(a)$ denotes $\mathit{tree}(\mathit{child}_1(a))$, the subtree rooted at $\mathit{child}_1(a)$, and $\mathit{tree}_0(a)$ denotes $\mathit{tree}(\mathit{child}_0(a))$, the subtree rooted at $\mathit{child}_0(a)$.

Definition 5.2.2 (Normal form)**.** *We say a DPLL refutation π is in* normal form *if the following conditions are satisfied:*

1. *Along any path from the root of π to a leaf node, no two nodes are labeled with the same variable*
2. *Let a be an interior node in π labeled by a variable x. Then there exists some leaf node in $\mathit{tree}_1(a)$ labeled by a clause containing the literal x^1, and there exists some leaf node in $\mathit{tree}_0(a)$ labeled by a clause containing the literal x^0.*

Lemma 5.2.3. *If there exists a DPLL refutation of a formula φ of size s, then there exists a normal form DPLL refutation of φ of size at most s.*

Proof. Let π be a DPLL refutation of a formula φ that is not in normal form. Suppose that condition 1 is violated, so that along some path from the root to a leaf node there are two nodes a and b, each labeled with a variable x. Without loss of generality, assume that a is above b in π, and that b is in $\mathit{tree}_1(a)$. Then if we modify π by replacing the

[3]In the usual setup a leaf node b is labeled with a clause that only contains literals whose negations are in $\rho(b)$. This corresponds more closely to the DPLL algorithm, where one branches on variables and backtracks after finding a clause that is falsified by the current restriction of the variables. Our definition is equivalent and will ease notation.

subtree $tree(b)$ with the subtree $tree_1(b)$, π will still be a valid DPLL refutation of φ, and the size of π will decrease. We can iterate this process until condition 1 is satisfied.

Now suppose that condition 2 is violated. Let a be an interior node in π labeled by a variable x, and without loss of generality suppose that there is no leaf node in $tree_1(a)$ that contains the literal x^1. Then if we modify π by replacing the subtree $tree(a)$ with the subtree $tree_1(a)$, π will still be a valid DPLL refutation of φ and the size of π will decrease. We can iterate this process until condition 2 is satisfied. □

Our interest in the DPLL system stems from the following connection to treelike resolution (see [Kra95] for instance).

Theorem 5.2.4. *An unsatisfiable k-CNF formula φ has a DPLL refutation of size s if and only if it has a treelike resolution refutation of size s.*

Proof sketch. Given a graph π corresponding to a treelike refutation $\mathcal{C}$ of φ, we can create a normal form DPLL refutation π' of φ with the exact same node and edge structure as π. The leaf nodes of π' will be labeled the same as in π. Suppose an interior node a of π has edges entering it from nodes b and c, and that a, b, and c correspond to the clauses C_a, C_b, and C_c in $\mathcal{C}$ respectively. Furthermore, suppose that in $\mathcal{C}$ C_a is derived from C_b and C_c by resolving on the variable x, and that x^1 is in C_b and x^0 is in C_c. Then in π', a will be labeled with the variable x, the edge going from b to a will be labeled with x^1 and the edge going from c to a will be labeled with x^0. If we form π' this way it will be a valid normal form DPPL refutation of φ.

Similarly, if π' is a normal form DPLL refutation of φ, we can create a treelike resolution refutation $\mathcal{C}$ of φ such that $|\mathcal{C}| = |\pi'|$. We will build $\mathcal{C}$ inductively using the tree π'. For each leaf node in π' labeled with a clause C, we add C to $\mathcal{C}$. Suppose an interior node a in π' has edges going into it from nodes b and c, and that C_b and C_c are the clauses already in $\mathcal{C}$ corresponding to the nodes b and c. Furthermore, suppose a is labeled with the variable x. Then we add C_a to $\mathcal{C}$ where C_a is the clause derived from C_b and C_c by resolving on x. If we form $\mathcal{C}$ this way, it will be a valid treelike resolution refutation that has a corresponding graph π with the same node and edge structure as π'. □

5.3 Statement of the Main Theorem and Description of the Coding Algorithm

We are now ready to state our main theorem from this chapter.

Theorem 5.3.1. *Let $c > \Delta^*$ be any constant. There exists a δ such that, with high probability, if $\mathcal{F}$ is a random 3-CNF formula with n variables and cn clauses, then $\mathcal{F}$ does not have a DPLL refutation of size at most 2^{n^δ}.*

Corollary 5.3.2. *Let $c > \Delta^*$ be any constant. There exists a δ such that, with high probability, if $\mathcal{F}$ is a random 3-CNF formula with n variables and cn clauses, then $\mathcal{F}$ does not have a treelike resolution refutation of size at most 2^{n^δ}.*

Proof of Theorem 5.3.1. By Lemma 5.1.3, we can reduce the proof to the following problem. We are given some unsatisfiable 3-CNF formula φ that has at most n variables and exactly cn distinct clauses. Furthermore, we are guaranteed that φ has a DPLL refutation of size at most 2^{n^δ}. We must show that we can find some subset of m clauses of φ, for $1 \leq m \leq cn$, such that we can encode these m clauses using at most $(2-\epsilon)m \log n$ bits for some $\epsilon > 0$. Our code must be independent of φ in the sense that we must use the same code for any such input to our problem.

Let π be the lexicographically first normal form DPLL refutation of φ of size at most 2^{n^δ}. We have no time constraints, so finding π is not an issue.

Our solution will be to design a coding algorithm ENCODE that traverses the tree π and always outputs a transcript *code* that is of size at most $(2-\epsilon)m \log n$ bits and encodes $m > 0$ clauses of φ. ENCODE will work by repeating the following steps:

```
ENCODE($\pi$)
1  Get next clause $C$
2  Encode $C$
3  Make cache deletion decisions
4  Check halting condition
```

Let us first informally discuss the basic ideas behind ENCODE.

As part of the algorithm, each time ENCODE reaches line 1 it must produce a new clause to encode. In order to do this, ENCODE will start at the root and perform a depth-first traversal of π. Whenever there is a choice of whether to move to the left child or the right child, the traversal will go in the direction of the subtree of smaller size, breaking ties arbitrarily. When the traversal reaches a leaf node labeled by a clause C, if C has not yet been encoded previously during the algorithm ENCODE will return this clause C (and continue the depth-first traversal from this point the next time it must get a clause). Otherwise, ENCODE backtracks and continues the depth-first traversal until a clause C' is found that has not yet been encoded previously.

Once a clause C has been found that has not previously been encoded, ENCODE must encode C in line 2. Throughout its execution, ENCODE maintains a transcript *code* that will be the output of the algorithm. ENCODE also maintains a "cache" of variables that starts off empty and will be used to encode clauses. The following three types of operations are recorded in *code*:

1. Adding a new variable to the cache
2. Encoding a clause C of φ
3. Deleting a variable from the cache

In order for ENCODE to encode the clause C, it first must ensure that all the variables in $var(C)$ are in the cache. Let $X \subseteq var(C)$ be the variables that are not currently in the cache. ENCODE adds each $x \in X$ to the cache one at a time and records this in *code*. To record entering a variable x in the cache, ENCODE must write $\log n$ bits to *code* to describe x.[4]

Once all the variable in $var(C)$ are in the cache, ENCODE encodes the clause C by indexing the three variables in $var(C)$ in the cache and using three extra bits to specify whether each variable appears as a positive or negative literal in C. To do this ENCODE must write $3 \log |cache| + 3$ bits to *code*, where $|cache|$ is the current size of the cache.

[4]Also, whenever an operation is recorded in *code* the type of operation must be specified. But this will only take a constant number of extra bits per clause encoded and will not affect our analysis.

Next, in step 3, ENCODE makes a decision about whether or not to delete some variables from the cache. If ENCODE decides to delete a variable x from the cache, it does this and then records this operation in *code* by writing $\log |cache|$ bits to *code* specifying x.

Finally, in step 4, ENCODE decides whether or not to halt. If it halts then the algorithm ends and ENCODE outputs *code*. Otherwise, the algorithm continues again at step 1.

Note that at any time during the execution of ENCODE, the current state of *code* implicitly defines *cache*, the variables that are in the cache at the current time, and *clauses*, the set of clauses in φ that have already been encoded.

Let us now give some intuition behind how ENCODE can compress the description of a set of clauses of φ. Suppose ENCODE only halts after it has traversed the entire tree π. Also, suppose we are able to ensure that the size of *cache* stays small throughout the execution of ENCODE (let us say $|cache| \leq O(\log n)$), and we can ensure that whenever a variable x is entered into the cache, it is accessed twice during an encode operation before it is deleted from the cache.

Then, since every time a clause is encoded exactly three variables are accessed from the cache, we get that the number of times a variable is added to the cache is at most $(3/2)m$, where m is the number of clauses that are encoded. Also, because the cache never grows to be bigger than $O(\log n)$, recording all the encode and deletion operations to *code* will together take at most $O(m \log\log n) = o(m \log n)$ bits. Therefore, including the add variable operations, when ENCODE halts and outputs *code* it will have length at most $(3/2)m \log n + o \log(n)$ bits, which is at most $(2 - \epsilon)m \log n$ bits for an appropriately chosen ϵ. The assumptions we make here are unrealistic, but this will be the intuition underlying how we define the cache deletion decisions and halting condition of ENCODE.

In order to describe further details of ENCODE we will need some more definitions.

Definition 5.3.3 (Encoding definitions)**.** *Let a be an interior node of π. If during the depth-first traversal of ENCODE a has not yet been reached we say that a is* unreached.

If ENCODE is in the process of traversing either $tree_0(a)$ or $tree_1(a)$ but has not yet traversed the other subtree, we say that a is in phase 1*. If ENCODE is in the process of traversing either $tree_0(a)$ or $tree_1(a)$ and has already finished traversing the other subtree, we say that a is* in phase 2*. If ENCODE has finished traversing all of $tree(a)$, then we say that a is* finished*.*

Let a be an interior node of π labeled by a variable x. Then $edge_1(a)$ denotes the edge labeled by x^1 entering a, $edge_0(a)$ denotes the edge labeled by x^0 entering a, and we say that $edge_1(a)$ and $edge_0(a)$ are partner *edges.*

Let $leaf$ be a leaf node in π labeled by a clause C. Suppose $x^i \in lits(C)$. Then along the path from the root of π to $leaf$ there must be some node a labeled with the variable x. If at some point during the execution of ENCODE the depth-first traversal reaches $leaf$ and causes ENCODE to encode C, we say that $edge_i(a)$ was covered *by C, and from that point on we say that the edge $edge_i(a)$ has been* covered*. For some node b, if $leaf$ is in $tree(b)$, we say that $edge_i(a)$ was* covered during the traversal of *$tree(b)$. (Note that this does not imply that $edge_i(a)$ is in $tree(b)$.)*

As a first try, let us define the cache deletion decisions of ENCODE as follows. Suppose ENCODE has just encoded a clause C in step 2 containing the literals x^i, y^j, z^k that labels a leaf node $leaf$ in π. In step 3 ENCODE now has to decide which if any variables to delete from the cache. The only variables that ENCODE will consider deleting are the variables x, y, and z. To decide whether to delete each of the variables, ENCODE will consider them one at time, using the following protocol.

Suppose ENCODE is deciding whether to delete the variable x. Because $x^i \in lits(C)$, there must be a node a on the path from the root to $leaf$ that is labeled by the variable x. Because ENCODE just encoded the clause C, the depth-first traversal is in the process of traversing $tree_i(a)$, so a is in phase 1 or in phase 2. If a is in phase 2 then delete x from the cache, and if a is in phase 1 then do not delete x from the cache.

The logic behind such a protocol is as follows. We know that because π is in normal form (see Definition 5.2.2), there is some leaf node $leaf_1$ in $tree_1(a)$ labeled by a clause

C_1 containing the literal x^1. Similarly, there is some leaf node $leaf_0$ in $tree_0(a)$ labeled by a different clause C_0 containing the literal x^0. The hope is that if we wait to delete x from the cache until after both C_0 and C_1 have been encoded, then we can ensure that every time a variable is added to the cache it is accessed twice during an encode operation before it is deleted from the cache like in our hypothetical situation from before.

The problem with this strategy is that although we have the guarantee that $leaf_0$ and $leaf_1$ exist in $tree(a)$, we do *not* have the guarantee that when ENCODE begins traversing $tree(a)$ both C_0 and C_1 have not been encoded previously during the algorithm. Indeed, it may be the case using this cache deletion decision protocol that x is entered into the cache while a is in phase 1 but no clause containing x is ever encoded while a is in phase 2 or vice versa, in which case x may only be accessed once for an encode operation before it is deleted from the cache. However, in order for this to be the case, it must be that x was entered into the cache previously during the algorithm and then removed. This suggests a strategy for dealing with these bad situations: if we can predict when such a situation will occur, we should *not* delete x from the cache the first time around, so that it will still be waiting there when we need to access it again.

This motivates our second try at defining the cache deletion decisions of ENCODE. This time the protocol will be more complex. As part of the cache deletion decisions ENCODE will color edges of π in order to keep track of information that will be needed to make optimal decisions (and which will be used in our analysis of the algorithm). ENCODE will also maintain a counter *count* that starts at 0 and will be used to define a new halting condition.

The coloring is based on a case analysis and may seem arbitrary; it will be helpful to remember that the purpose of the coloring is to keep track of those situations where a variable may be entered into the cache and only accessed once for an encode operation before being deleted from the cache. In general, the colors have the following meaning:

- All edges start off colored *black*. If an edge is black, it means that it has not yet been covered during ENCODE.

- When an edge is covered during ENCODE, it is colored *green* if its partner edge has not been covered yet and would never be covered during ENCODE, even if ENCODE was allowed to traverse all of π (i.e. ENCODE did not stop in the middle of the traversal because of a halting condition)

- When an edge is covered during ENCODE, it is colored *white* if it has not been covered previously, and its partner edge has already been covered or would be covered during ENCODE at some point if ENCODE was allowed to traverse all of π.

- When an edge is covered during ENCODE, if it is then colored *blue* it means the edge had already been covered previously during ENCODE.

Every time an edge is colored (or recolored) either green or blue, ENCODE will increment *count*. The new halting condition will be to halt if $count \geq 12n^{\delta}$, or if ENCODE finishes traversing all of π, whichever comes first. (This halting condition prevents the cache from getting too large, as we will see later in the analysis).

We now give the new cache deletion decisions protocol in full detail. Again the setup is as follows: we are assuming that ENCODE has just encoded a clause C containing the literal x^i that labels a leaf node $leaf$. ENCODE is now deciding whether or not to delete the variable x from the cache. We know there exists a node a on the path from the root of π to $leaf$ labeled with the variable x, and that a is either in phase 1 or phase 2. The edge $edge_i(a)$ has just been covered.

We will break up the protocol into two cases, depending on whether a is in phase 1 or phase 2.

First suppose a is in phase 1. We now consider the following sub-cases, which are listed in priority ordering since they are not completely disjoint. (For instance, both sub-case 1 and sub-case 2 could occur, in which case ENCODE follows the rules of sub-case 1).

1. Let $\mathcal{C}$ be the set of all clauses labeling a leaf node in $tree_{1-i}(a)$. Let $\mathcal{C}' \subseteq \mathcal{C}$ be the set of all such clauses that have not yet been encoded during ENCODE. Suppose

$\forall D \in \mathcal{C}'$, $x \notin var(D)$. This means that when ENCODE later traverses $tree_{1-i}(a)$, it will not encode any clause containing x. (Note that every clause labeling a leaf node in $tree_i(a)$ that contains x contains the literal x^i, and every clause labeling a leaf node in $tree_{1-i}(a)$ that contains x contains the literal x^{1-i}, so for the rest of the time a is in phase 1 whether or not $\mathcal{C}'$ contains a clause with the variable x will not change.) In this case ENCODE colors (or recolors) $edge_i(a)$ green, increments *count*, and removes x from the cache.

2. $edge_i(a)$ is black. ENCODE colors $edge_i(a)$ white and leaves x in the cache.

3. $edge_i(a)$ is not black. ENCODE colors (or recolors) $edge_i(a)$ blue, increments *count*, and leaves x in the cache.

Now suppose a is in phase 2. We consider the following sub-cases (this time the sub-cases are disjoint). Note that in all these sub-cases x is always removed from the cache.

1. $edge_{1-i}(a)$ is black. Since ENCODE has already traversed all of $tree_{1-i}(a)$, this means that $edge_{1-i}(a)$ will never be covered during ENCODE. In this case ENCODE colors (or recolors) $edge_i(a)$ green, increments *count*, and removes x from the cache.

2. $edge_i(a)$ is black and $edge_{i-1}(a)$ is not black. ENCODE colors $edge_i(a)$ white and removes x from the cache.

3. $edge_i(a)$ is not black and $edge_{1-a}(a)$ is not black. ENCODE colors (or recolors) $edge_i(a)$ blue, increments *count*, and removes x from the cache.

We are not quite done yet. ENCODE now is keeping track of the information it needs, but it is not using this information to make optimal cache deletion decisions yet.

The actual coding algorithm we will use will be the following algorithm ENCODE′. ENCODE′ first runs ENCODE as we have defined it above, but throws away its output. Let ENCODE1 denote this first execution of ENCODE. Let *colored* be the set of edges in π that are colored either green or blue during ENCODE1. Let *var-colored* be the set of variables that label one of the edges in *colored* (ignoring the sign of the literal).

ENCODE$'$ then runs ENCODE again. Let ENCODE2 denote this second execution of ENCODE. The traversal, coloring, and halting condition will be the same as in ENCODE1. Therefore ENCODE2 will halt at the same point that ENCODE1 does. The only difference between ENCODE2 and ENCODE1 is that ENCODE2 has the following modification to the cache deletion decisions protocol of ENCODE.

Suppose that ENCODE2 has just encoded a clause containing the literal x^i and is deciding whether to delete x from the cache. If $x \in$ *var-colored*, and later on in ENCODE2 an edge from *colored* labeled with x (ignoring the sign of the literal) will be colored (or recolored) green or blue, then ENCODE2 leaves x in the cache *regardless* of what the cache deletion decisions protocol calls for. This is well-defined, since the coloring is the same in ENCODE1 and ENCODE2 and they halt at the same time and we have already run ENCODE1. Otherwise ENCODE2 makes cache deletion decisions according to the protocol of ENCODE. The output of ENCODE$'$ is the output of ENCODE2.

5.4 Analysis of the Coding Algorithm

5.4.1 The Balanced DPLL Case

We continue the proof of Theorem 5.3.1 by analyzing ENCODE$'$. Our goal is to show that for some $\epsilon > 0$ the output of ENCODE2, *code*, has length at most $(2-\epsilon)m \log n$ bits, where m is the number of clauses encoded in *code*. (Note that from *code* we can reconstruct every clause that is encoded in *code*.)

In this section, to clarify the main ideas, we will do the analysis for a simplified case, where we assume that π is a *balanced* DPLL refutation, i.e. any path from the root of π to a leaf node has length at most $O(\log n)$. In the next section we will do the analysis for the more general case, where π may be unbalanced.

The main challenge in the unbalanced case will be to prove Lemma 5.4.2, which states that the cache never grows to be too large during ENCODE1. From there it will be easy to show that the cache never grows to be too large during ENCODE2, which will be the key to proving that the output of ENCODE$'$ is sufficiently short. In the balanced case, proving Lemma 5.4.2 is much simpler.

Let us break an execution of ENCODE into discrete time steps. We say that ENCODE is at time $t = 0$ at the beginning of the algorithm, and every time ENCODE performs an action such as taking a step in the depth-first traversal, adding a variable to the cache, encoding a clause, or deleting a variable from the cache, we increment t by 1. If ENCODE is at time t of its execution, then $\mathit{node}(t)$ denotes the last node that was visited during the depth-first traversal of ENCODE, ρ_t denotes the set of nodes along the path from the root of π to $\mathit{node}(t)$, and $\mathit{cache}(t)$ denotes the set of variables that are in the cache at time t. Also $\mathit{var}(\rho_t)$ denotes the set of variables that label some node in ρ_t.

The following lemma will help us prove Lemma 5.4.2.

Lemma 5.4.1. *Suppose ENCODE1 is at time t of its execution. Then*

$$y \in \mathit{cache}(t) \Rightarrow y \in \mathit{var}(\rho_t)$$

Proof. Let y be a variable such that $y \notin \mathit{var}(\rho_t)$. Suppose y is added to the cache at some time $t' < t$ in order for ENCODE1 to encode some clause C containing the literal y^i that labels a leaf node leaf. Then there exists a node a on the path from the root of π to leaf that is labeled by y, and $a \notin \rho_t$ since $y \notin \mathit{var}(\rho_t)$. Examining the cache deletion decision protocol of ENCODE1, we see that if y is not deleted from the cache immediately after C is encoded, it is because a is in phase 1 and there is another clause C' containing the literal y^{1-i} labeling some leaf node leaf' in $\mathit{tree}_{1-i}(a)$ that had not yet been encoded by time t'. (Remember that in ENCODE1 we use the cache deletion decisions protocol of ENCODE without the extra modifications of ENCODE2). Because $a \notin \rho_t$, ENCODE1 will have traversed all of $\mathit{tree}_{1-i}(a)$ by time t. So at some time t'' such that $t' < t'' < t$, ENCODE1 will encode C'. At this time a will be in phase 2, so, according to the cache deletion decisions of ENCODE1, y will immediately afterwards be removed from the cache. This proves that if $y \notin \mathit{var}(\rho_t)$, then $y \notin \mathit{cache}(t)$ □

Lemma 5.4.2. *During an execution of ENCODE1 the cache never contains more than* $14n^{\delta}$ *variables*

Proof. Suppose ENCODE1 is at time t of its execution. By lemma 5.4.1, $|cache(t)| \leq |var(\rho_t)|$. Because π is balanced, we have that $|var(\rho_t)| \leq O(\log n)$, so $|cache(t)| \leq O(\log n) \leq 14n^{\delta}$ for large n. $\square$

Corollary 5.4.3. *During an execution of ENCODE2 the cache never contains more than* $26n^{\delta}$ *variables*

Proof. The only difference between ENCODE1 and ENCODE2 is that variables from *var-colored* are potentially kept in the cache in ENCODE2 when they would be removed in ENCODE1. We have that $|var\text{-}colored| \leq |colored| \leq 12n^{\delta}$ due to the halting condition of ENCODE. Therefore the cache at time t in ENCODE2 can contain at most $12n^{\delta}$ more elements than the cache at time t in ENCODE1. $\square$

Corollary 5.4.4. *Let endcache be the set of variables in the cache when ENCODE2 halts. Then* $|endcache| \leq 14n^{\delta}$.

Proof. The cache deletion decision protocol of ENCODE2 is such that if ENCODE2 halts at time t, its cache at time t is the same as the cache of ENCODE1 at time t. $\square$

Suppose ENCODE2 has halted and outputted *code*. We are ready to bound $|code|$, the length of *code* in bits. Let $\mathcal{T} = t_1, t_2, \ldots t_l$ be a list of the instances during ENCODE2 in which a variable was written to the cache, ordered chronologically. Let $var(t_i)$ be the variable that was added to the cache at time t_i. Note that it is possible that $var(t_i) = var(t_j)$ for some $i \neq j$. Let $num(t_i)$ be the number of times the variable $var(t_i)$ was accessed from the cache for an encode operation after time t_i before being removed from the cache (including the time $var(t_i)$ is accessed immediately after being added to the cache at time t_i) Let *num-colored* be the number of times that an edge is colored (or recolored) green or blue during ENCODE2.

Lemma 5.4.5.

$$l \leq \frac{\sum_{i=1}^{l} num(t_i) - num\text{-}colored + |endcache|}{2}$$

Proof. Suppose at time t of its execution ENCODE2 had just encoded a clause C containing the literal x^i that labels a leaf node $leaf$. Then there exists a node a on the path from the root of π to $leaf$ that is labeled with the variable x. After encoding C, during the cache deletion decisions when ENCODE2 decided whether to delete x from the cache, ENCODE2 colored the edge $edge_i(a)$ some color.

Suppose first that ENCODE2 colored $edge_i(a)$ blue. This means that $x \in$ *var-colored* and $edge_i(a)$ had been covered previously, so some clause $C' \neq C$ containing x^i had been encoded at some time $t' < t$. Then, according to the cache deletion decisions protocol of ENCODE2, when C' was encoded the variable x must have been left in the cache and would still be there at time t when C was encoded.

Similarly, suppose ENCODE2 colored $edge_i(a)$ green. Again, this implies that $x \in$ *var-colored*. Also, because $edge_i(a)$ is colored green it means that at the time ENCODE2 began traversing $tree(a)$, $tree_{1-i}(a)$ did not contain any leaf node labeled with a clause C' containing the literal x^{1-i} that had not already been encoded. But because π is in normal form (see Definition 5.2.2), there must exist some leaf node in $tree_{1-i}(a)$ labeled with a clause C' containing the literal x^{1-i} that had already been encoded previously. Then, according to the cache deletion decisions protocol of ENCODE2, when C' was encoded the variable x was left in the cache and would still be there at time t when C was encoded.

Therefore, if $edge_i(a)$ was colored green or blue, $num(t_i)$ was incremented for some t_i without adding any new element to $\mathcal{T}$.

Now suppose that $edge_i(a)$ was colored from black to white, but that $x \notin endcache$. Then there exists a clause C' labeling a leaf node $leaf$ in $tree_{1-i}(a)$ that contains the literal x^{1-a} and had not been encoded by time t (let $leaf$ be the first node labeled by such a C' that ENCODE2 would reach during the traversal of $tree_{1-i}$). It may be that in order to encode the clause C ENCODE2 had to add x to the cache, in which case a new t_j would have been added to the list $\mathcal{T}$. But according to the cache deletion decision protocol of ENCODE2, x would have remained in the cache until C' was encoded, and since $x \notin endcache$, it must be that C' was encoded before ENCODE2 halted. Therefore $num(t_j) \geq 2$.

Putting all this information together, we get that

$$\sum_{i=1}^{l} num(t_i) \geq 2l + num\text{-}colored - |endcache|$$

from which the claim follows. $\square$

Let m be the total number of clauses encoded in *code*. Then $m = \sum_{i=1}^{l} num(t_i)/3$. The total length of *code* is the cost of documenting every addition and deletion of a variable to/from the cache, plus the cost of encoding the m clauses by indexing three elements of the cache per clause, plus a constant number of bits per clause that is encoded. There can be at most $3m$ variables added to the cache during ENCODE2, so altogether there can be at most $3m$ total deletion operations recorded in code. By Corollary 5.4.3, the cache never grows to be larger than $26n^{\delta}$ during ENCODE2, so altogether these deletion operations contribute at most $3\delta m \log n + O(m)$ bits to $|code|$. Similarly, the encode operations will together contribute at most $3\delta m \log n + O(m)$ bits to $|code|$ as well. Therefore in total, the deletion and encode operations will contribute at most $6\delta m \log n + O(m)$ bits to $|code|$.

Each time a variable is added to the cache it takes $\log n$ bits to document this, so the total number of bits needed to document all the cache insertions is $l \log n$. Therefore we have that $|code| \leq l \log n + 6\delta m \log n + O(m)$.

Now suppose that ENCODE traverses all of π before halting. In this case $|endcache| = 0$. Using Lemma 5.4.5,

$$\begin{aligned}
|code| &\leq l \log n + 6\delta m \log n + O(m) \\
&\leq \left(\frac{\sum_{i=1}^{l} num(t_i) - num\text{-}colored + |endcache|}{2} \right) \log n + 6\delta m \log n + O(m) \\
&\leq \frac{3}{2} m \log n + 6\delta m \log n + O(m) \\
&\leq \left(\frac{3 + 12\delta}{2} \right) m \log n + O(m)
\end{aligned}$$

Otherwise suppose that ENCODE halts before traversing all of π. In this case we have that $num\text{-}colored = 12n^{\delta}$ and, by Corollary 5.4.4, $|endcache| \leq 14n^{\delta}$. Therefore, using Lemma 5.4.5 and the fact that $m \geq \mathit{num\text{-}colored}/3 = 4n^\delta$ we get that

$$\begin{aligned}
|code| &\leq l \log n + 6\delta m \log n + O(m) \\
&\leq \left(\frac{\sum_{i=1}^{l} num(t_i) - \mathit{num\text{-}colored} + |endcache|}{2} \right) \log n + 6\delta m \log n + O(m) \\
&\leq \left(\frac{3m + 2n^\delta}{2} \right) \log n + 6\delta m \log n + O(m) \\
&\leq \left(\frac{7m}{4} \right) \log n + 6\delta m \log n + O(m)
\end{aligned}$$

Therefore,

$$|code| \leq max \left(\left(\frac{3 + 12\delta}{2} \right) m \log n + O(m), \left(\frac{7m}{4} \right) \log n + 6\delta m \log n + O(m) \right)$$

Choosing δ small enough we get that $|code| \leq (15/8) m \log n$, which is $(2 - \epsilon) m \log n$ for $\epsilon = 1/8$.

□

5.4.2 The Unbalanced DPLL Case

In this section we redo the analysis of the coding algorithm for the more general case where π is not assumed to be balanced.

The only difference between the balanced and the unbalanced case is that the proof of Lemma 5.4.2, which states that the cache never grows to be too large during ENCODE1, is far more complicated; the rest of the analysis goes through as before. In the balanced case we crucially used the fact that any path in π can contain at most $O(\log n)$ nodes, so that immediately we have that $|var(\rho_t)| \leq O(\log n)$. In the unbalanced case $var(\rho_t)$ can contain up to $\Omega(n)$ variables, so we must argue that despite this only a small number of these variables can be in the cache at the same time.

First we prove a combinatorial lemma that we will need later on.

Let T be a rooted binary tree, where each edge is colored black or white. We call T properly colored if for every interior node a of the tree, the two edges entering a are the same color.

Consider the following one-player coloring game: T originally starts off colored all black, except for two edges lying on some path P from the root of T to a leaf node in T that are colored white and are called the *initial edges* of the game. On each turn the player is allowed to color three black edges in T white provided they all lie on the same path from the root of T to a leaf node in T. The player wins if he is able to properly color T.

Lemma 5.4.6. *It is impossible to win the coloring game*

Proof. Call an interior node a in T bad if the two edges entering a are different colors. A proper coloring implies that there are no bad nodes in T. Let W be the set of black edges entering a bad node (every bad node has exactly one black edge entering it). We can assume without loss of generality that on each turn the player colors white some edge in W. To see this, let *edge* be an edge that is in W before move i. In order for the player to win, on some move $j \geq i$, *edge* must be colored white. Suppose the player wins, and let σ be the winning sequence of moves that the player plays. If the order of moves in σ is transposed in any way, it will still be a winning sequence of moves. Therefore we can exchange move j with move i, so that on move i *edge* is colored white.

We claim the following two invariants are maintained throughout the game:

1. Let $\{edge1, edge2\} \subseteq W$. Then there does not exist a path in T from the root to a leaf node that contains both $edge1$ and $edge2$.

2. For two edges $edge1$ and $edge2$, let $anc(edge1, edge2)$ denote the least ancestor of $edge1$ and $edge2$ (i.e. the farthest node from the root such that any path from the root that includes $edge1$ must pass through $anc(edge1, edge2)$ and any path from the root that includes $edge2$ must pass through $anc(edge1, edge2)$). Let $\{edge1, edge2\} \subseteq W$. Consider the two edges going into $anc(edge1, edge2)$. One of these two edges is colored white.

Let a and b denote the two bad nodes at the start of the game, and e and f the two elements of W at the start of the game. Because the two initial edges lie on the path P, invariant 1 will be satisfied at the beginning of the game. Also, $anc(e, f)$ will be either the node a or the node b, so invariant 2 will be satisfied at the beginning of the game as well.

Now suppose on some move the player colors white some $edge1 \in W$. By invariant 1, this move can only "fix" one bad node, so it will create two new bad nodes $bad1$ and $bad2$ in T. Let $bad1_B$ be the black edge entering $bad1$ after this move and $bad1_W$ the white edge entering $bad1$. Similarly let $bad2_B$ be the black edge entering $bad2$ after the move and $bad2_W$ the white edge entering $bad2$. ($bad1_B$ and $bad2_B$ will be the new members of W).

First consider the relationship between $bad1_B$ and $bad2_B$. Because there exists some path from the root to a leaf node in T that contains both $bad1_W$ and $bad2_W$, there will be no path from the root to a leaf node in T that contains both $bad1_B$ and $bad2_B$, so these two edges will not violate invariant 1 after the move. Also, because there exists some path from the root to a leaf node in T that contains both $bad1_W$ and $bad2_W$, either $bad1_W$ or $bad2_W$ will be going into $anc(bad1_B, bad2_B)$, so $bad1_B$ and $bad2_B$ will not violate invariant 2 after the move either.

Now consider some arbitrary edge $edge2 \in W$ that was in W before the move, and suppose for contradiction that after the move $edge2$ violates invariant 1 with one of the new members of W, say $bad1_B$. This means that $bad1_B$ and $edge2$ lie along the same path P'. $edge2$ cannot be above $bad1_B$ in P', because in this case $edge2$ and $edge1$ are both on some path, which would violate the fact that invariant 1 held before the move. Therefore $bad1_B$ is above $edge2$ in P'. We know that $edge1$ and $bad1_W$ lie on some path P'' since they were part of the same move. If $edge1$ is above $bad1_W$ in P'' then again $edge2$ and $edge1$ are both on some path, which would violate the fact that invariant 1 held before the move. Therefore $edge1$ is below $bad1_W$ in P''. But in this case $bad1$ is the node $anc(edge1, edge2)$, which is a contradiction since by invariant 2 one of the edges going into $anc(edge1, edge2)$ was already white before the move.

Now we show that $edge2$ and $bad1_B$ do not violate invariant 2 after the move either. Because every path from the root that goes through $edge1$ must pass through $anc(edge1, edge2)$, and $edge1$ and $bad1_W$ lie along some path from the root to a leaf node since they are part of the same move, after the move there will be some path P'

containing $bad1_W$ that goes through the node $anc(edge1, edge2)$. Suppose that $bad1_W$ is below $anc(edge1, edge2)$ in P'. Then $anc(edge2, bad1_B) = anc(edge1, edge2)$ so invariant 2 will not be violated by $edge2$ and $bad1_B$ after the move since invariant 2 held before the move. Otherwise suppose that $bad1_W$ is above $anc(edge1, edge2)$ in P'. Then $bad1_W$ is one of the edges going into $anc(edge2, bad1_B)$, so after the turn one of the edges going into $anc(edge2, bad1_B)$ is white and invariant 2 will not be violated by $edge2$ and $bad1_B$ in that case either.

By invariant 1, after every move $|W|$ increases by 2, so the player can never win. $\square$

We now give some definitions and prove a couple smaller lemmas that will allow up to prove Lemma 5.4.2 in the unbalanced case.

For a node $a \in \rho_t$ such that $a \neq node(t)$, we define $child_{\rho_t}(a)$ to be the child node of a that is in ρ_t, $edge_{\rho_t}(a)$ to be the edge connecting a and $child_{\rho_t}(a)$, and $tree_{\rho_t}(a)$ to be $tree(child_{\rho_t}(a))$. Similarly, we define $child_{\bar{\rho}_t}(a)$ to be the child node of a that is not in ρ_t, $edge_{\bar{\rho}_t}(a)$ to be the edge connecting a and $child_{\bar{\rho}_t}$ and $tree_{\bar{\rho}_t}(a)$ to be $tree(child_{\bar{\rho}_t}(a))$.

Let $phase1(t) \subseteq \rho_t$ be the set of nodes in ρ_t that are in phase 1 at time t, and $phase2(t) \subseteq \rho_t$ be the set of nodes in ρ_t that are in phase 2 at time t. (All nodes in ρ_t will either be in phase 1 or phase 2 at time t.)

Note that the following two lemmas refer to ENCODE1, the first execution of ENCODE by ENCODE′ whose output is ignored.

Lemma 5.4.7. *At time t of ENCODE1, $|phase1(t)| \leq n^\delta$.*

Proof. Suppose for contradiction there are more than n^δ nodes in ρ_t in phase 1. Consider a node $a \in \rho_t$ that is in phase 1. Then, because the depth-first traversal always moves toward the subtree of smaller size, $size(tree_{\rho_t}(a)) \leq size(tree_{\bar{\rho}_t}(a))$. Therefore, because there are more than n^δ nodes in ρ_t in phase 1, π must have size greater than 2^{n^δ}, which contradicts the fact that π is supposed to have size at most 2^{n^δ}. $\square$

Definition 5.4.8. *For an interior node b in π, let $var(b)$ be the variable labeling b and $edges(b)$ be the set of edges that lie on the path from the root of π to b.*

Lemma 5.4.9. *Suppose ENCODE1 is at time t of its execution, and let $b \in phase2(t)$. Suppose $edge_{\bar{\rho}_t}(b)$ is covered at some point during the traversal of $tree_{\bar{\rho}_t}(b)$ by a clause C, and neither $edge_{\bar{\rho}_t}(b)$ nor any edge in $tree_{\bar{\rho}_t}(b)$ is colored green or blue during the traversal of $tree_{\bar{\rho}_t}(b)$. Then some edge in $edges(b)$ is covered during the traversal of $tree_{\bar{\rho}_t}(b)$.*

Proof. Suppose for contradiction that $edge_{\bar{\rho}_t}(b)$ is covered at some point during the traversal of $tree_{\bar{\rho}_t}(b)$, that neither $edge_{\bar{\rho}_t}(b)$ nor any edge in $tree_{\bar{\rho}_t}(b)$ is colored green or blue during the traversal of $tree_{\bar{\rho}_t}(b)$, and that no edge in $edges(b)$ is covered during the traversal of $tree_{\bar{\rho}_t}(b)$.

Define an instance of the coloring game from Lemma 5.4.6 as follows. $tree_{\bar{\rho}_t}(b)$ will be the tree T of the coloring game. The other two edges that are covered by C during the traversal of $tree_{\bar{\rho}_t}(b)$ are the initial edges of the game. Since by assumption no edge in $edges(b)$ is covered during the traversal of $tree_{\bar{\rho}_t}(b)$, they will both be in T. Also, because any edges covered by C must lie on a single path, they will both lie on a single path as required by the definition of the coloring game.

Let the player's moves be defined by the clauses that are encoded during the traversal of $tree_{\bar{\rho}_t}(b)$ other than C. Because $b \in phase2(t)$, by time t ENCODE1 has already traversed all of $tree_{\bar{\rho}_t}(b)$. For every clause C' that is encoded, the player plays a move where he colors white the three edges of T that correspond to the three edges covered by C'. Note that every time the player colors three edges white they lie on a single path, and by the assumption that no edge in $edges(b)$ is covered during the traversal of $tree_{\bar{\rho}_t}(b)$ and that $edge_{\bar{\rho}_t}(b)$ is not colored green or blue, they also all lie within T.

Also, no edge in $tree_{\bar{\rho}_t}(b)$ is ever colored blue, so the player does not ever illegally color the same edge twice. Furthermore, the player wins this coloring game; if there were a bad node a in T after the player was done making all his moves, then, according to the cache deletion decisions protocol of ENCODE1, either $edge_0(a)$ or $edge_1(a)$ would have to have been colored green during ENCODE1, which contradicts the assumption that no edge in $tree_{\bar{\rho}_t}(b)$ is colored green.

But this is a contradiction, since by Lemma 5.4.6 it is impossible to win the coloring game.

□

We are now ready to restate and prove the main lemma in the unbalanced case.

Lemma 5.4.10 (Restatement of Lemma 5.4.2)**.** *During an execution of ENCODE1 the cache never contains more than* $14n^{\delta}$ *variables*

Proof. Suppose ENCODE1 is at time t of its execution.

The high level strategy for proving Lemma 5.4.10 is as follows. By Lemma 5.4.1 a variable can only be in the cache at time t if it is $var(\rho_t)$. We will define a set of nodes that we will call *good*, and establish a one-to-one mapping η from good nodes to elements of $var(\rho_t)$ that cannot be in the cache at time t. Then, by lower bounding the number of good nodes, we also lower bound the number of elements of $var(\rho_t)$ that cannot be in the cache at time t, and thus we get an upper bound on the number of variables that can be in the cache at time t.

As part of this process we will need to define another one-to-one mapping ν from instances where an edge has been colored (or recolored) green or blue in ENCODE1 to nodes in π.

Now we give the details. Let $b \in phase2(t)$. We will consider a number of disjoint cases.

1. Suppose that $var(b)$ has not ever been entered into the cache by ENCODE1 during the traversal of $tree(b)$. In this case, we say b is a good node and define $\eta(b) = var(b)$.

2. Suppose that during the traversal of $tree_{\bar{\rho}_t}(b)$ an edge $edge_{\rho_t}(a) \in edges(b)$ was covered for some $a \in phase2(t)$ at time $t' < t$. Furthermore, suppose $edge_{\rho_t}(a)$ was black before it was covered. Then, because $a \in phase2(t)$, according to the cache deletion decisions protocol of ENCODE1, $var(a)$ will be removed from the cache, and it cannot be in the cache at time t. We will therefore call b a good node and define $\eta(b) = var(a)$. Note that once an edge is covered it will be colored something other than black and cannot ever become black again, and also that

for two nodes $b, b' \in \rho_t$, the trees $tree_{\bar{\rho}_t}(b)$ and $tree_{\bar{\rho}_t}(b')$ are disjoint. Therefore η can only map a single node to $var(a)$, so the property that η is one-to-one is maintained.

Suppose that in addition when $edge_{\rho_t}(a)$ was covered, it was colored from black to green. Then we will say that a is *responsible for coloring an edge* and define $\nu(edge_{\rho_t}(a), t') = a$. This is an important point and is part of why these definitions are so involved; notice that in this case both an edge is colored either green or blue *and* a variable is popped out of the cache. The calculations in our analysis will be delicate enough that we must take special care to associate these instances with two different nodes through η and ν, or else the math would not go through.

3. Suppose that during the traversal of $tree_{\bar{\rho}_t}(b)$ an edge $edge$ is covered at time $t' < t$, where $edge$ is either in $tree_{\bar{\rho}_t}(b)$ or $edge = edge_{\bar{\rho}_t}(b)$. Furthermore, suppose that after $edge$ is covered it is colored either green or blue. Then we say that b is responsible for coloring an edge and define $\nu(edge, t') = b$.

4. Suppose that during the traversal of $tree_{\bar{\rho}_t}(b)$ an edge $edge \in edges(b)$ was covered for some time $t' < t$. Furthermore, suppose $edge$ was not black before it was covered (i.e. it had been covered previously), so that after it was covered it was recolored either green or blue. Then we say that b is responsible for coloring an edge and define $\nu(edge, t') = b$.

5. Suppose that during the traversal of $tree_{\bar{\rho}_t}(b)$ an edge $edge_{\rho_t}(a) \in edges(b)$ was covered for some $a \in phase1(t)$ at time $t' < t$. Furthermore, suppose $edge_{\rho_t}(a)$ was black before it was covered, and then colored white immediately afterwards as part of the cache deletion decisions. Note that in this case, because a is in phase 1, the variable $var(a)$ will *not* be removed from the cache according to the cache deletion decisions protocol of ENCODE1. In this case we say that b is *responsible for covering a phase 1 edge.*

Note that as we have defined η and ν, they are both one-to-one mappings.

Now let us attempt to count the number of good nodes. Let $b \in phase2(t)$. Suppose that b is not responsible for coloring an edge and is not responsible for covering a phase 1 edge.

If $var(b)$ has not ever been entered into the cache during the traversal of $tree_{\bar{\rho}_t}(b)$, then by item 1 above b is a good node. Otherwise the edge $edge_{\bar{\rho}_t}(b)$ must have been covered during the traversal of $tree_{\bar{\rho}_t}(b)$. To see this, note that if $edge_{\bar{\rho}_t}(b)$ were not covered during the traversal of $tree_{\bar{\rho}_t}(b)$, then $edge_{\rho_t}(b)$ would have to have been covered during the traversal of $tree_{\rho_t}(b)$ in order for $var(b)$ to be in the cache. But then, according to the cache deletion decisions protocol of ENCODE1, $edge_{\rho_t}(b)$ would be colored green, and so by item 2 above, b is responsible for coloring an edge, which contradicts our assumption.

Also, neither $edge_{\bar{\rho}_t}(b)$ nor any edge in $tree_{\bar{\rho}_t}(b)$ was colored green or blue during the traversal of $tree_{\bar{\rho}_t}(b)$, otherwise by item 3 above b would be responsible for coloring an edge. Therefore, by Lemma 5.4.9, some $edge \in edges(b)$ was covered during the traversal of $tree_{\bar{\rho}_t}(b)$. $edge$ must have been black before it was covered, or else by item 4 above b would have been responsible for coloring an edge. Then we know that $edge = edge_{\rho_t}(a)$ for some $a \in phase2(t)$, otherwise b would be responsible for covering a phase 1 edge, which contradicts our assumption. Therefore, by item 2 above, b is a good node in this case as well.

We have that the number of nodes that are responsible for coloring an edge can be at most $12n^{\delta}$, due to the halting condition of ENCODE. Also, by Lemma 5.4.7 we have that $|phase1(t)| \leq n^{\delta}$, so there can be at most n^{δ} nodes responsible for covering a phase 1 edge.

Therefore we get the following lower bound on the number of good nodes

$$\begin{aligned}(\#\text{ of good nodes }) &\geq |phase2(t)| - 12n^{\delta} - n^{\delta}\\ &= (|\rho_t| - |phase1(t)|) - 13n^{\delta}\\ &\geq |\rho_t| - 14n^{\delta}\end{aligned}$$

By Lemma 5.4.1 a variable can only be in the cache at time t if it is in ρ_t. Because η is a one-to-one mapping between good nodes and elements of ρ_t that cannot be in the cache at time t, we get that

$$\begin{aligned}|cache(t)| &\leq |\rho_t| - (|\rho_t| - 14n^{\delta}) \\ &= 14n^{\delta}\end{aligned}$$

□

5.5 Open Questions

The main open question is whether these methods can be extended to prove lower bounds for stronger systems than treelike resolution. An important first step is to prove general resolution lower bounds; although exponential lower bounds for general resolution refutations of random CNFs already exist, this would be progress towards showing the validity of these new methods. So far the author has been unsuccessful at this attempt, although he is optimistic that it is doable.

The reason why the proof in this paper does not extend immediately to the general resolution case is that it relies crucially on the fact that every time ENCODE takes a step in its depth-first traversal, because it always moves towards the smaller subtree, it is "cutting off" at least half of the proof tree. This property is used to prove Lemma 5.4.7, which is subsequently used to prove the main lemma, Lemma 5.4.10, which says that the cache never grows to be too large. In the general resolution case, if we take a step in a depth-first traversal, it is possible that the majority of the proof graph is still reachable regardless of which child node we choose to go to, so there is no way to "cut off" at least half of the proof graph as before.[5]

The proof of Theorem 5.3.1 is messy at times, particularly in the description of the cache deletion decisions protocol of ENCODE and the proof of Lemma 5.4.10. Is it possible to simplify the coding algorithm or its analysis in the treelike resolution case? Is it possible to adjust the framework in order to match the optimal treelike resolution bounds? Perhaps these improvements would clarify how to extend the proof to the general resolution case.

[5]However, one can easily extend the proof of the balanced DPLL case to work for *balanced* resolution refutations, where the proof graph only contains paths of length at most $O(\log n)$.

It would also be good to better understand to what extent these techniques are qualitatively different from the standard techniques that have been used to prove resolution lower bounds. Is the idea of constructing coding algorithms to indirectly generate proof complexity lower bounds fundamentally new and potentially powerful, or is it in some way just a re-framing and obfuscation of older techniques? Are there limits to how strong a proof system we can potentially apply these techniques to?

References

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, 2009.

[ABFL12a] E. Allender, H. Buhrman, L. Friedman, and B. Loff. Reductions to the set of random strings: The resource-bounded case. 2012. Submitted for publication; an earlier version appeared as [ABFL12b].

[ABFL12b] E. Allender, H. Buhrman, L. Friedman, and B. Loff. Reductions to the set of random strings:the resource-bounded case. In B. Rovan, V. Sassone, and P. Widmayer, editors, *Proceedings of the 37th MFCS*, volume 7464 of *LNCS*, pages 88–99. Springer, 2012.

[ABK06a] E. Allender, H. Buhrman, and M. Koucký. What can be efficiently reduced to the Kolmogorov-random strings? *Annals of Pure and Applied Logic*, 138:2–19, 2006.

[ABK$^+$06b] E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35:1467–1493, 2006.

[Ach09] D. Achlioptas. Random satisfiability. *Handbook of Satisfiability*, pages 245–270, 2009.

[ADF$^+$12] E. Allender, G. Davie, L. Friedman, S. B. Hopkins, and I. Tzameret. Kolmogorov complexity, circuits, and the strength of formal theories of arithmetic. Technical Report TR12-028, Electronic Colloquium on Computational Complexity, 2012. Submitted for publication.

[AFG10] E. Allender, L. Friedman, and W. Gasarch. Exposition of the Muchnik-Positselsky construction of a prefix free entropy function that is not complete under truth-table reductions. Technical Report TR10-138, Electronic Colloquium on Computational Complexity, 2010.

[AGF13] E. Allender, W. Gasarch, and L. Friedman. Limits on the computational power of random strings. *Information and Computation (Special Issue on ICALP 2011)*, 222:80–92, 2013.

[Ajt94] M. Ajtai. The complexity of the pigeonhole principle. *Combinatorics 14*, 4:417–433, 1994.

[AKV04] A. Atserias, P. Kolaitis, and M. Vardi. Constraint propagation as a proof system. In *Tenth International Conference on Principles and Practice of Constraint Programming*, pages 77–91, 2004.

[Ale05] M. Alekhnovich. Lower bounds for k-DNF resolution on random 3-CNFs. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 251–256, 2005.

[AW08] S. Aaronson and A Wigderson. Algebrization: A new barrier in complexity theory. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 731–740. ACM, 2008.

[BFKL10] H. Buhrman, L. Fortnow, M. Koucký, and B. Loff. Derandomizing from random strings. In *25th IEEE Conference on Computational Complexity (CCC)*, pages 58–63. IEEE, 2010.

[BFNV05] H. Buhrman, L. Fortnow, I. Newman, and N. Vereshchagin. Increasing Kolmogorov complexity. In V. Diekert and B. Durand, editors, *STACS 2005*, volume 3404 of *Lecture Notes in Computer Science*, pages 412–421. Springer Berlin / Heidelberg, 2005.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the P =? NP question. *SIAM J. Comput.*, 4(4):431–442, 1975.

[BJ85] S. W. Bent and J. W. John. Finding the median requires 2n comparisons. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 213–216. ACM, 1985.

[BKS04] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[BM97] H. Buhrman and E. Mayordomo. An excursion to the Kolmogorov random strings. *J. Comput. Syst. Sci.*, 54(3):393–399, 1997.

[BP98] P. Beame and T. Pitassi. Propositional proof complexity: Past, present and future. Technical Report (98)067, Electronic Colloquium on Computational Complexity, 1998.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computing*, 35:677–691, 1986.

[BSG03] E. Ben-Sasson and N. Galesi. Space complexity of random formulae in resolution. *Random Structures and Algorithms*, 23(1):92–109, 2003.

[BSI99] E. Ben-Sasson and R. Impagliazzo. Random CNF's are hard for the polynomial calculus. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 415–421, New York, NY, 1999.

[BSIW04] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near optimal separation of treelike and general resolution. *Combinatorica*, 24(4):585–603, 2004.

[BSW01] E. Ben-Sasson and A. Wigderson. Short proofs are narrow — resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001.

[BW96] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

[CKS81] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[CR79] S.A. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979.

[CS88] V. Chvátal and E. Szémeredi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, 1988.

[Day09] A. Day. On the computational power of random strings. *Annals of Pure and Applied Logic*, 160:214–228, 2009.

[DBM03] O. Dubois, Y. Boufkhad, and J. Mandler. Typical random 3-sat formulae and the satisfiability threshold. Technical Report (10)003, Electronic Colloquium on Computational Complexity, 2003.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 7:394–397, 1962.

[DM02] O. Dubois and J. Mandler. The 3-XORSAT threshold. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 769–778, 2002.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Communications of the ACM*, 7:201–215, 1960.

[FR75] R. W. Floyd and R. L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165–172, 1975.

[Fri13] L. Friedman. A framework for proving proof complexity lower bounds on random CNFs using encoding techniques. Technical Report TR13-027, Electronic Colloquium on Computational Complexity, 2013.

[FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

[FX13] L. Friedman and Y. Xu. Exponential lower bounds for refuting random formulas using ordered binary decision diagrams. In *The 8th International Computer Science Symposium in Russia*, 2013. To appear.

[Gol11a] Oded Goldreich. In a world of P=BPP. In *Studies in Complexity and Cryptography*, volume 6650 of *Lecture Notes in Computer Science*, pages 191–232. Springer, 2011.

[Gol11b] Oded Goldreich. Two comments on targeted canonical derandomizers. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:47, 2011.

[GZ03] J. F. Groote and H. Zantema. Resolution and binary decision diagrams cannot simulate each other polynomially. *Discrete Applied Mathematics*, 130:157–171, 2003.

[Hak85] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–305, 1985.

[Hal35] P. Hall. On representatives of subsets. *J. London Math. Soc.*, 10:26–30, 1935.

[Has86] J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 6–20. ACM, 1986.

[HD04] J. Huang and A. Darwiche. Toward good elimination ordering for symbolic SAT solving. In *Proceedings of the Sixteenth IEEE Conference on Tools with Artificial Intelligence*, pages 566–573, 2004.

[Hel86] Hans Heller. On relativized exponential and probabilistic complexity classes. *Information and Control*, 71(3):231–243, 1986.

[Hit10] John M. Hitchcock. Lower bounds for reducibility to the Kolmogorov random strings. In *Proc. Computability in Europe (CiE)*, volume 6158 of *Lecture Notes in Computer Science*, pages 195–200. Springer, 2010.

[IW97] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proc. ACM Symp. on Theory of Computing (STOC) '97*, pages 220–229, 1997.

[IW01] R. Impagliazzo and A. Wigderson. Randomness vs time: Derandomization under a uniform assumption. *Journal of Computer and System Sciences*, 63:672–688, 2001.

[JL00] D. W. Juedes and J. H. Lutz. Modeling time-bounded prefix Kolmogorov complexity. *Theory of Computing Systems*, 33(2):111–123, 2000.

[Kra95] J. Kraj́icek. *Bounded arithmetic, propositional logic and complexity theory.* Cambridge University Press, 1995.

[Kra07] J. Krajíček. An exponential lower bound for a constraint propagation proof system based on ordered binary decision diagrams. Technical Report (07)007, Electronic Colloquium on Computational Complexity, 2007.

[Kum96] M. Kummer. On the complexity of random strings. In *Proc. of Symp. on Theo. Aspects of Comp. Sci. (STACS)*, volume 1046 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 1996.

[LV08] M. Li and P. Vitanyi. *Introduction to Kolmogorov Complexity and its Applications.* Springer, third edition, 2008.

[Mar66] D. Martin. Completeness, the recursion theorem and effectively simple sets. *Proc. Am. Math. Soc.*, 17:838–842, 1966.

[Mil13] J. Miller. Personal Communication, 2013.

[MP02] A. A. Muchnik and S. Positselsky. Kolmogorov entropy in the context of computability theory. *Theoretical Computer Science*, 271:15–35, 2002.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[PBI93] T. Pitassi, P. Beame, and R. Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational Complexity 3*, 2:97–140, 1993.

[Pud97] P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, 1997.

[PV04] G. Pan and M. Vardi. Search vs. symbolic techniques in satisfiability solving. In *The Seventh International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[Raz85] A. A. Razborov. Lower bounds on the monotone complexity of some boolean functions. *Dokl. Akad. Nauk. SSSR*, 281(4):798–801, 1985.

[Raz87] A. A. Razborov. Lower bounds on the size of bounded depth networks over a complete basis with logical addition. *Mathematical Notes of the Academy*, 41(4):598–607, 1987.

[Raz98] A. A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, 1998.

[Raz03] A. A. Razborov. Pseudorandom generators hard for k-DNF resolution and polynomial calculus resolution. *manuscript*, 2003.

[RR97] A. A. Razborov and S. Rudich. Natural proofs. *J. Comput. Syst. Sci*, 55(1):24–35, 1997.

[Seg07] N. Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 54:40–44, 2007.

[Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 77–82. ACM, 1987.

[SW93] D. Seiling and I. Wegener. NC-algorithms for operations on binary decision diagrams. *Parallel Processing Letters*, 3(1):3–12, 1993.

[Tse68] G.S. Tseitin. On the complexity of derivations in the propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, Part II, pages 115–125. Consultants Bureau, New York and London, 1968.

[TSZ09] O. Tveretina, C. Sinz, and H. Zantema. An exponential lower bound on OBDD refutations for pigeonhole formulas. In *Athens Colloquium on Algorithms and Complexity*, pages 13–21. Electronic Proceedings in Theoretical Computer Science, 2009.

[TSZ10] O. Tveretina, C. Sinz, and H. Zantema. Ordered binary decision diagrams, pigeonhole formulas and beyond. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:35–38, 2010.

[Wil11] R. Williams. Non-uniform ACC circuit lower bounds. In *Proc. IEEE Conf. on Computational Complexity*, 2011.

[Yao82] A. C. C. Yao. Theory and applications of trapdoor functions. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 80–91, 1982.