2015

Randomness in completeness and space-bounded computations

Debasis Mandal
Iowa State University

Randomness in completeness and space-bounded computations

by

Debasis Mandal

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:

Pavan Aduri, Major Professor

David Fernández-Baca

Jack H. Lutz

Ryan Martin

Giora Slutzki

Iowa State University

Ames, Iowa

2015

DEDICATION

I would like to dedicate this dissertation to ma and baba for all the sacrifices they made over many years to make sure that my brother and I have a good education.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

It has been a pleasant journey of learning avidly, thinking deep and hard, and developing new ideas over the last seven years. This journey would not have been possible without the continuous support, guidance, and patience of my advisor Prof. Pavan Aduri. When I decided to delve into the world of complexity theory, I was a complete novice in the field. With his relentless patience, Pavan not only helped me learn and understand this fascinating area of Computer Science, but to mature as a researcher, and contribute. He taught me how to think deep, add rigor to work, and question even the published results. Thank you Pavan.

I thank Prof. David Fernández-Baca, Prof. Jack H. Lutz, Prof. Ryan Martin, and Prof. Giora Slutzki for serving as committee members on my Program of Study. I took courses with all of them as a graduate student and learned a great deal of theory and Mathematics from each of them. I would particularly like to mention the courses I took with Jack. Not only did I learn an enormous amount of theoretical computer science during my various interactions with him, his teaching style motivated me to inculcate some of them in me. From him I also learned one of the greatest ideologies as a researcher that doing science is a group activity. Special thanks is also due to Dr. Steve Kautz for agreeing to serve on my POS during the preliminary oral examination.

I thank my collaborators: Prof. Alan L. Selman, Prof. N. V. Vinodchandran, Andrew Hughes, Nathan Russell, and Rajeswari Venugopalan.

A special thanks to Prof. Bivas Mitra. I would not have pursued graduate study had you not instilled in me the passion for research during my undergrad days. A special thanks to Dr. Simanta Mitra and Dr. Jim Lathrop with whom I was TA for the undergrad algorithms course which gave me the opportunity to sharpen my teaching skills.

I was fortunate to share the theory lab with Adam, Xiang, Don, and Titus. I will miss the impromptu discussions with Adam about everything and the weekly brainstorming sessions in

the grad theory seminar. They were fun.

I thank Preetham, Harish, Akshay, Ujjal, and Upamanyu for being both great roommates and friends. Thanks to Eeshita, Oliva, Nivedita, Anusha, Tanmoy, Beas, Chiranjit, Arindam, Pratik, Priyanka, Soumya, Gauree, Raji, Swapnanjan, Sourajit, Tushi, Subhomoy, Arpa, Anindya, Hyun t, Jinu, Reddy for all the good time at Ames. Special thanks to Sugam ji, Hritu, and Sudipta for the delicious foods when I badly needed some. Thanks to Sandipan da and Gopal for being available for advice and guidance whenever I needed them. I still cherish the great friendship with Rajib, Jayita, Mrinmoy, and Gitika since my undergrad days. Thank you guys for being there.

Finally, I would like to extend my sincere gratitude to my parents, parents-in-law, my brother Dibyendu and his wife Swatilekha for their support and encouragement throughout. Saptaparni came into my life towards the end of my graduate days and has been the sole witness of my transition from a student life to a professional life. I sincerely thank her for all her patience in coping with it.

Last, but not the least, I greatly acknowledge the support of the National Science Foundation through the grants CCF: 0916797 and CCF: 1421163 that helped me continue my research.

ABSTRACT

The study of computational complexity investigates the role of various computational resources such as processing time, memory requirements, nondeterminism, randomness, nonuniformity, etc. to solve different types of computational problems. In this dissertation, we study the role of randomness in two fundamental areas of computational complexity: NP-completeness and space-bounded computations.

The concept of completeness plays an important role in defining the notion of 'hard' problems in Computer Science. Intuitively, an NP-complete problem captures the difficulty of solving any problem in NP. Polynomial-time reductions are at the heart of defining completeness. However, there is no single notion of reduction; researchers identified various polynomial-time reductions such as many-one reduction, truth-table reduction, Turing reduction, etc. Each such notion of reduction induces a notion of completeness. Finding the relationships among various NP-completeness notions is a significant open problem. Our first result is about the separation of two such polynomial-time completeness notions for NP, namely, Turing completeness and many-one completeness. This is the first result that separates completeness notions for NP under a *worst-case* hardness hypothesis.

Our next result involves a conjecture by Even, Selman, and Yacobi [ESY84, SY82] which states that there do not exist disjoint NP-pairs all of whose separators are NP-hard via Turing reductions. If true, this conjecture implies that a certain kind of probabilistic public-key cryptosystems is not secure. The conjecture is open for 30 years. We provide evidence in support of a variant of this conjecture. We show that if there exist certain secure one-way functions, then the ESY conjecture for the bounded-truth-table reduction holds.

Now we turn our attention to space-bounded computations. We investigate probabilistic space-bounded machines that are allowed to access their random bits *multiple times*. Our main conceptual contribution here is to establish an interesting connection between derandomization

of such probabilistic *space-bounded* machines and the derandomization of probabilistic *time-bounded* machines. In particular, we show that if we can derandomize a multipass machine even with a small number of passes over random tape and only $O(\log^2 n)$ random bits to deterministic polynomial-time, then BPTIME(n) $\subseteq$ DTIME($2^{o(n)}$). Note that if we restrict the number of random bits to $O(\log n)$, then we can trivially derandomize the machine to polynomial time. Furthermore, it can be shown that if we restrict the number of passes to $O(1)$, we can still derandomize the machine to polynomial time. Thus our result implies that any extension beyond these trivialities will lead to an unknown derandomization of BPTIME(n).

Our final contribution is about the derandomization of probabilistic time-bounded machines under branching program lower bounds. The standard method of derandomizing time-bounded probabilistic machines depends on various circuit lower bounds, which are notoriously hard to prove. We show that the derandomization of low-degree polynomial identity testing, a well-known problem in co-RP, can be obtained under certain branching program lower bounds. Note that branching programs are considered weaker model of computation than the Boolean circuits.

CHAPTER 1. INTRODUCTION

Solving 'hard problems' has been one of the important measures of developments of human knowledge since ancient times. The notion of this 'hardness' is informally characterized by the number of serious but futile attempts by problem-solvers over many years. A problem such as Fermat's last theorem is considered to be a hard problem, as it resisted any solution for nearly 350 years, until mathematician Andrew Wiles finally solved it in 1994. Similarly, Goldbach's conjecture is also considered to be a hard problem as it has been open for over a quarter of a millennium. However, in the computational world, there is a formal way to characterize the computational difficulty of a problem through the study of computational complexity. In particular, given a problem, we ask the following questions: how much processing time does it take to solve the problem, or how much memory does it require, or can we solve it faster if we introduce randomness in the computation or if nondeterminism is used? Thus the objective of computational complexity is to determine the necessary and sufficient resources (such as time, memory, randomness, etc.) required to solve computational problems.

Problems in computer science are classified into various complexity classes based on the amount of resources required to solve them. Two problems are said to be computationally equivalent if they belong to the same complexity class and thus require similar amount of resources. Arguably, the two most important complexity classes in computer science are P and NP. P represents the set of problems that can be *solved* by deterministic Turing machines in polynomial time and NP is the set of problems whose solutions can be *verified* by deterministic Turing machines in polynomial time. Analogous to the above time-complexity classes, L (deterministic logspace) and NL (nondeterministic logspace) are defined for the space-bounded computations to classify problems based on their memory requirements.

Processing time and memory requirements (space) have been considered important re-

sources since the early days of computation. However, the formal study involving randomness in computation is relatively new. In 1977, Gill [Gil77] introduced a new model of computation called *probabilistic Turing machines* which are Turing machines with the ability to toss an unbiased coin. Naturally, these machines are allowed to make mistakes in their computations. Based on their error probability, Gill formulated several polynomial-time randomized complexity classes such as BPP, RP, and ZPP. Since then there has been a plethora of work in complexity theory involving randomness [MR96, MU05, AB09, MM11]. New areas of study inside complexity theory have evolved: randomness extraction, pseudorandomness, derandomization to name a few. The area of cryptography, sampling, probabilistic proof systems, and the interactive proof systems solely depend on the existence of randomness in computation. Today randomness is considered an essential resource in the study of computational complexity.

The focus of this dissertation is to investigate the role of randomness as a computational resource in the study of two specific areas of complexity theory: NP-completeness and space-bounded computations.

NP-completeness

The notion of NP-completeness, introduced in early seventies by Cook [Coo71], Karp [Kar72], and Levin [Lev73], is central to understanding the nature of hard problems in NP. Intuitively, they capture the computational difficulty of all problems in NP. If we can solve any NP-complete problem in polynomial time, the entire NP class collapses to the "easy" class P. However, there is no single notion of NP-completeness and this makes it even more interesting. The key concept in defining the NP-completeness is the notion of polynomial-time reductions. Informally, reductions translate instances of one problem to instances of another problem; a problem A is polynomial-time reducible to a problem B if A can be solved in polynomial-time by making queries to problem B. By varying the manner in which the queries are made, we obtain a wide spectrum of reductions. At one end of the spectrum is *Cook/Turing reduction* where multiple queries are allowed and a query may depend on answers to previous queries. On the other end is the most restrictive reduction, *Karp-Levin/many-one reduction*, where each positive instance of problem A is mapped to a positive instance of problem B, and so are the

negative instances. In between are *truth-table/non-adaptive reductions*, and *bounded truth-table reductions*. It is easy to see that the many-one reduction implies truth-table reduction and the truth-table reduction implies the Turing reduction.

With the above polynomial-time reductions at hand, we can now formally define various notions of NP-completeness. A language L is called NP-hard via many-one reduction if every language in NP many-one reduces to L. The language L is called *many-one complete* for NP if L is also in NP. Even though it is standard to define completeness using many-one reductions, one can similarly define completeness using Turing, truth-table, or bounded truth-table reductions. Note that the seminal paper of Cook used Turing reduction to define NP-completeness, whereas the works of Karp and Levin used many-one reductions. Turing reductions are arguably more appropriate to define completeness to capture the intuition that if a complete problem for NP is "easy" then the entire class is easy. However, all known natural languages, like traveling salesperson problem or Hamiltonian circuit problem, turn out to be complete under many-one reductions. At this point, it is natural to ask: What is the relationship between many-one completeness and Turing completeness for NP? Since many-one reduction implies Turing reduction, any many-one complete language for NP, like the above two mentioned, is also a Turing complete language for NP. Does the converse hold? What is the relationship between NP-hardness via many-one reduction and NP-hardness via Turing reduction?

In the first part of this dissertation, we use randomness as a tool to investigate the relationships among various notions of NP-completeness and NP-hardness. Note that the definition of NP-completeness does not involve randomness in any way. Polynomial-time randomized reductions play an important role in all of these proofs. We would like to mention that randomized reductions have been used in some of the seminal results in complexity theory since early days; for example to show that interactive protocol is same as polynomial space [Sha92, LFKN92], worst-case hardness reduces to average-case hardness for EXP [BFL91, BFNW93, IW97], and a problem in polynomial hierarchy can be solved in polynomial-time given access to #SAT oracle [Tod91b].

Probabilistic Space-bounded Computations

The next part of this dissertation is focused on the study of probabilistic space-bounded computations and the derandomization (removing randomness from computation). Our main objective is to investigate the different types of access-mechanisms of the random tape by the probabilistic machines and how it affects their computations. In the traditional definition of probabilistic space-bounded computations, a probabilistic machine can access its random tape in a *one-way*, read-only manner and the random tape does not count towards the space complexity of the probabilistic machine. In particular, the machine cannot reread the random bits unless they are stored in its work tapes. This access mechanism is the most natural one as it corresponds to modeling probabilistic machines as coin-tossing machines, originally defined by Gill [Gil77]. The complexity class BPL is the class of languages accepted by such bounded-error probabilistic machines that use logarithmic space and halt in polynomial time. Note that we only consider probabilistic machines that halt on all inputs on all settings of the random tape. If we do not require the machines to halt, then we get a potentially larger class of languages [KV85, Sak96]. The class RL is the one-sided error counterpart of BPL. Whether BPL or even RL can be derandomized to deterministic logspace is one of the central questions in the study of space-bounded computation. In spite of clear and steady progress in space-bounded derandomization, this question is far from settled [Nis92, INW94, SZ99, RR99, RTV06, Rei08, CRV11].

On the other extreme, there are two-way probabilistic space-bounded machines that have two-way access to the random tape. Two-way machines are considered to be significantly more powerful than their one-way counterpart [BCP83]. For example, Nisan [Nis93] showed that a two-way zero-error probabilistic logspace machine can simulate a one-way two-sided bounded error probabilistic logspace machine, but the reverse is not known to hold. Similarly, BPL is known to be in logspace with a linear amount of advice [FK06], whereas the best we know about the 2-*way*BPL machine is that they can be simulated by a logspace machine with polynomial amount of advice [Adl78]. In between the standard one-way model and the two-way access model are the *multipass* models, where the machines make multiple passes

over the random tape and in each pass they access their random tape in a traditional one-way manner. Clearly, the number of passes in the multipass machines act as a parameter to move from the one-way access to the two-way access models. This model was first considered by David, Papakonstantinou, and Sidiropoulos [DPS11]. We investigate the consequences of *derandomizing* both the multipass machines and the two-way machines. Our motivation is the following: Is there any implication of derandomizing these machines to the derandomization of the probabilistic *time-bounded* classes?

Given a $t(n)$ time-bounded probabilistic Turing machine, a deterministic machine can simulate it by cycling through all possible random strings of length at most $t(n)$ and taking the majority vote. Such trivial way of derandomizing any probabilistic machine takes exponentially more time than the time taken by the original probabilistic machine. The natural question is: Is there a way to simulate probabilistic time-bounded machines deterministically without an exponential blow-up in time? This is one of the central open questions in complexity theory. In complexity-theoretic terms, this question can be phrased as whether BPP (or Promise-BPP) is in deterministic subexponential time. It has been established that existence of problems (in exponential time) with high circuit complexity (intuitively, this means the size of the Boolean circuit to compute that language is large) can be used to obtain a faster deterministic simulation of probabilistic time-bounded computations. The celebrated result of Impagliazzo and Wigderson [IW97] states that if there are languages in E with $2^{\epsilon n}$ circuit complexity, then Promise-BPP is in P. Weaker circuit complexity hardness assumptions yield weaker simulations. For example, if EXP does not have polynomial-size circuits, then BPP is in sub-exponential time. These "hardness versus randomness" trade-offs [BFNW93, NW94, Uma03, ISW06, KvMS11] crucially rest on the notion of efficient *pseudorandom generator*—easily computable functions that stretch short random seeds to long seeds that appear random to time-bounded machines.

Though the existence of problems in EXP with high circuit complexity is a believable and well accepted hypothesis, we are far from proving such circuit lower bounds. For example, even showing that NEXP has languages that do not have polynomial-size circuits will be a major breakthrough. Given this, it is natural to ask if there are alternate routes to achieve derandomization that do not involve proving circuit lower bounds. This is exactly where the

derandomization of the aforementioned multi-access probabilistic space-bounded models come into play.

Now we provide a summary of the main results in this dissertation.

Summary of Results

Our first contribution in this dissertation is to address the following question from the previous discussion: Is there a Turing complete language for NP that is not many-one complete for NP? This will separate Turing and many-one completeness for NP. This question was first raised by Ladner, Lynch, and Selman [LLS75] in their seminal paper where they introduced several polynomial time reductions such as truth-table reduction, bounded-truth-table reductions, etc. It is easy to see that a positive answer to the question will separate P from NP, the most outstanding question in computer science. They conjectured that the reverse also holds, i.e., if P $\neq$ NP, then Turing and many-one completeness are different for NP. The question remained open nearly for two decades. The first major progress was made by Lutz and Mayordomo [LM96] who showed that Turing and many-one completeness are different for NP if a stochastic property about NP (called *measure hypothesis*) holds. Since then, efforts have been made [ASB00, PS02, PS04, HPV08, GHP11] to prove the separation between these completeness notions in NP under a weaker hypothesis. However, all of these hypotheses involved either the almost-everywhere hardness of NP (which says that NP is hard on all but finitely many inputs) or a combination of average-case hardness and worst-case hardness. But the natural problems are believed to be worst-case hard or average-case hard (for example, satisfiability is believed to be average-case hard). Thus the question of whether we can separate many-one and Turing completeness for NP only under a worst-case hardness hypothesis still remained open. We resolve this question in Chapter 3.

Our next result is about an interesting conjecture about promise problems. A promise problem is a disjoint pair—a pair of disjoint sets (Π_y, Π_n), Π_y is called the set of "yes" instances and Π_n is the set of "no" instances. Their union $\Pi_y \cup \Pi_n$ is called the *promise*. For a promise problem (Π_y, Π_n), one is interested in the following computational question: Is there an efficient algorithm that tells whether an instance x lies in Π_y or not, under the promise

that x is in $\Pi_y \cup \Pi_n$. The algorithm may give an arbitrary answer if the promise does not hold, i.e., $x \notin \Pi_y \cup \Pi_n$. More formally, a *solution/separator* of a promise problem is any set S that includes Π_y and is disjoint from Π_n. A promise problem is considered easy if it admits a solution in P and is hard if every solution is computationally difficult. The ESY conjecture, due to Even, Selman, and Yacobi, concerns the computational difficulty of disjoint NP-pairs. They conjectured that there do not exist promise problems all of whose solutions are NP-hard [SY82, ESY84] via Turing reductions. If true, the ESY conjecture implies that NP $\neq$ co-NP and NP $\neq$ UP [1], and this is one reason obtaining evidence for the ESY conjecture has been hard for the last 30 years. In Chapter 4, we consider variants of the ESY conjecture and show that under some reasonable hypotheses these variants follow.

Our first result in the domain of space-bounded computations involves an interesting connection between the derandomization of the multipass machines and the derandomization of the linear probabilistic time-bounded machine. As our main result in Chapter 5, we show the following (informally): If every language decided by a bounded-error probabilistic logspace machine that uses $O(\log^2 n)$ random bits and makes a non-constant number of passes over its random tape is in deterministic polynomial time, then linear probabilistic time can be derandomdized to deterministic sublinear exponential time. Our main conceptual contribution is that derandomizing such probabilistic space-bounded machines leads to derandomization of probabilistic *time-bounded* classes. There are two reasons why we think this is a fruitful avenue to explore. First, the pseudorandom generators for the space-bounded computations [Nis92, INW94, NZ96] are unconditional compared to the pseudorandom objects for the time-bounded classes whose existence depends on the hard-to-prove circuit lower bounds. Second, the logspace probabilistic machine can be derandomized to an $O(\log^{3/2} n)$-space-bounded deterministic machine [SZ99] and the belief in the theory community is actually that BPL = L.

Our final contribution in this dissertation (in Chapter 6) is the derandomization of probabilistic time-bounded classes under branching program lower bounds (instead of circuit lower bounds). In particular, we show that if the linear exponential time E does not have $2^{\epsilon n}$-size branching programs, then a particular problem in co-RP, low-degree polynomial identity test-

[1] UP stands for Unambiguous nondeterministic polynomial-time.

ing over the field $\mathbb{Q}$, can be solved deterministically in time $n^{O(\log n)}$. The motivation is that branching programs are believed to be weaker nonuniform model than Boolean circuits and thus showing lower bounds against them should be easier.

CHAPTER 2. PRELIMINARIES

In this chapter, we introduce the notation and terminology used throughout the rest of this dissertation. We also review few basic concepts, computational models, and the complexity classes from computational complexity. We assume the familiarity with the standard notation and definitions in complexity theory [Pap94, Gol08, AB09, MM11].

2.1 Languages and Basic Model of Computation

All languages are defined over the the binary alphabet $\Sigma = \{0, 1\}$, Σ^n denotes the set of all binary strings of length n. We use $|x|$ to denote the length of a string x. We assume the standard lexicographic order on strings. We use $x - 1$ to denote the immediate predecessor of x in this order. Given a language L and a string x, $L(x)$ denotes the characteristic function of L, and $L|x$ is defined as $L(\lambda)L(0)L(1)\cdots L(x-1)$.

We also assume that the computations are done using floor function of x, $\lfloor x \rfloor$, wherever necessary. All the time functions $t : \mathbb{N} \to \mathbb{N}$ we consider are time-constructible and all the space functions are assumed to space-constructible. Further, $\log^c n$ or polylog(n) represents $(\log n)^c$ for some integer $c > 0$.

We use both uniform and nonuniform models of computation. Turing machine is the choice of our uniform model of computation. We use deterministic, nondeterministic, and probabilistic Turing machines both for time- and space-bounded computations. Boolean circuits and branching programs are considered the nonuniform models of computation and used throughout this dissertation. An O-oracle circuit C, denoted by C^O, is a Boolean circuit with access to oracle O. We assume the familiarity with these standard models of computation [AB09].

Non-deterministic double-exponential time is defined by NEEXP $= \bigcup_{c>1} \text{NTIME}(2^{2^{n^c}})$

and co-NEEXP is its complement class. We say that a nedeterministic machine is a NEEXP machine, if its runtime is bounded by $2^{2^{n^c}}$ for some $c > 1$. A language L is in ZPTIME($t(n)$), if there is a probabilistic machine Z running in time $O(t(n))$ such that for every x, $\Pr[Z(x) = L(x)]$ is atleast $1/4$, and the probability that Z outputs an incorrect answer is zero. The machine Z may output $\perp$ with certain probability (at most $3/4$).

The class QuasiNP is the set of languages that can be accepted by nondeterministic Turing machines running in quasi-polynomial-time, i.e., QuasiNP $= \cup_{c>0}$NTIME$(2^{\log^c n})$.

The following machine is a variant of standard definition of Turing machine and computes a function rather than behaving as a decider.

Definition 2.1. Suppose N is a nondeterministic machine accepting a language S. We say that a *$t(n)$-time bounded, zero-error, probabilistic machine computes accepting computations of* N if there exists a probabilistic machine Z such that

(a) For every $x \in S$, for every choice of random bits $Z(x)$ either outputs a string from Σ^* or outputs the special symbol $\perp$.

(b) for every $x \in S$, $\Pr[Z(x)$ is an accepting computation of $N(x)] > 1/4$, and

(c) for every $x \in S$, $\Pr[Z(x) \neq \perp$ and is not an accepting computation of $N(x)] = 0$.

2.2 Polynomial-time Reductions

Consider two languages A and B. A is *polynomial-time Turing reducible* to B, denoted by $A \leq_T^{\mathrm{P}} B$, if there is a polynomial-time oracle Turing machine M such that $A = L(M^B)$. Note that M can make at most polynomially many queries to B and they can be *adaptive*. The language A is *polynomial-time truth-table reducible* to B, denoted by $A \leq_{tt}^{\mathrm{P}} B$, if there is a pair of polynomial time computable functions $\langle f, g\rangle$ such that for every $x \in \Sigma^*$, (1) $f(x)$ is query set $Q = \{q_1, q_2, \cdots, q_k\}$ and (2) $x \in A \iff g(x, B(q_1), B(q_2), \cdots, B(q_k)) = 1$. We call f the *query generator* and g the *truth-table evaluator*. Given a polynomial time reducibility $\leq_r^{\mathrm{P}}$, a set B is $\leq_r^{\mathrm{P}}$*-complete* for NP if B is in NP and for every set $A \in$ NP, A is $\leq_r^{\mathrm{P}}$ reducible to B.

A language A is *k-truth-table reducible* to a language B (denoted $A \leq^{\mathrm{P}}_{ktt} B$) if there exist two polynomial-time computable functions f and t such that for every x,

$$f(x) = \langle q_1, \cdots, q_k \rangle \text{ and } t(x, B(q_1), \cdots, B(q_k)) = A(x).$$

We say that A is *bounded-truth-table reducible* to B (denoted $A \leq^{\mathrm{P}}_{btt} B$) if there exists an integer $k > 0$ such that $A \leq^{\mathrm{P}}_{ktt} B$.

Definition 2.2. A function $f\colon \Sigma^* \to \Sigma^*$ is SNP *computable* if there is a nondeterministic polynomial-time bounded Turing machine M such that for every x at least one path of $M(x)$ outputs $f(x)$ and no path outputs a wrong answer. Some paths may output $\perp$.

We will also consider strong nondeterministic reductions. These reductions were originally defined by Adleman and Manders [AM77]. We slightly modify their definition to suit our purposes.

Definition 2.3. Let A and B be two languages. We say that A is *strong nondeterministic k-truth-table reducible* to B (denoted $A \leq^{\mathrm{SNP}}_{ktt} B$), if there is a polynomial-time computable function f and an SNP computable function t such that for every x, $f(x) = \langle q_1, \cdots, q_k \rangle$, and $t(x, B(q_1), \cdots, B(q_k)) = A(x)$. We say that A is *strong nondeterministic bounded-truth-table reducible* to B ($A \leq^{\mathrm{SNP}}_{btt} B$) if there exists a $k > 0$ such that $A \leq^{\mathrm{SNP}}_{ktt} B$.

Remark. Note that in this definition, the reduction does not use nondeterminism to produce the queries. The original definition of Adleman and Manders [AM77] allows the query generator f also to be SNP computable. Similarly, we can define the notion of strong nondeterministic quasi-polynomial-time reductions.

Definition 2.4. We say that A is reducible to B via *length-increasing, strong nondeterministic, k-truth-table reductions (denoted $\leq^{\mathrm{SNP}}_{ktt,li}$)* if $A \leq^{\mathrm{SNP}}_{ktt} B$ and the length of every query is bigger than the length of the input.

Notions of length-increasing are defined similarly for $\leq^{\mathrm{P}}_{ktt}$, $\leq^{\mathrm{P}}_{btt}$, and $\leq^{\mathrm{SNP}}_{btt}$-reductions.

2.3 P-selective Sets

We use the notion of P-selective sets introduced by Selman [Sel79].

Definition 2.5. A set $S \subseteq \Sigma^*$ is P-*selective* if there is a polynomial time computable function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that for all strings $x, y \in \Sigma^*$, (1) $f(x, y) \in \{x, y\}$, and (2) If either of x and y is in S, then $f(x, y)$ is in S. The function f is called the P-*selector* of S.

The well-known example of P-selective sets are the *left-cut* sets $L(r) = \{x \mid x < r\}$, where r is an infinite binary sequence, and $<$ is the dictionary order with $0 < 1$. The following lemma is due to Toda [Tod91a].

Lemma 2.1. *For every* P*-selective set L, there is a polynomial time algorithm that given any finite set of strings Q as input, outputs a sequence $x_1, \cdots, x_m$ such that $\{x_1, \cdots, x_m\} = Q$, such that for some integer p, $0 \leq p \leq m$, $Q \cap L = \{x_i \mid i \leq p\}$ and $Q \cap \bar{L} = \{x_i \mid i > p\}$.*

2.4 Unpredictability

Now we define the notion of *unpredictability* which is similar to the notion of *genericity* considered elsewhere in the literature.

Definition 2.6. We say that a nondeterministic machine M is *strong* if for every input x, exactly one of the following conditions hold:

1. at least one path of M accepts x and no path rejects,
2. at least one path of M rejects x and no path accepts.

Some paths of the machine may output $\perp$.

Definition 2.7. Let M be a strong nondeterministic machine and L be a language. We say that M is a *predictor* for L if for every $x \in L$, M accepts $\langle x, L|x\rangle$ and for every $x \notin L$, M rejects $\langle x, L|x\rangle$.

Definition 2.8. Let $t(n)$ be any time bound. We say that a language L is SNTIME($t(n)$)-*unpredictable* if for every strong nondeterministic machine M that predicts L, every path of M runs for more than $t(n)$ time for *all but finitely many* inputs of the form $\langle x, L|x\rangle$.

Definition 2.9. Let A and L be two languages. We say that L is $\mathrm{SNTIME}(t(n))$-*unpredictable within* A if $L \subseteq A$ and for every strong nondeterministic machine M that predicts L, for all but finitely many x from A, M runs for more than $t(n)$ time on inputs of the form $\langle x, L|x\rangle$.

We use the following theorem from [HPRS12] concerning the existence of unpredictable sets within $\overline{\mathrm{SAT}}$.

Theorem 2.1. *For every $k > 0$, there is a set R such that R is* $\mathrm{SNTIME}(2^{\log^k n})$-*unpredictable within* $\overline{\mathrm{SAT}}$.

2.5 Secure One-way Functions and Cryptographic Generators

In this section, we define the notions of secure one-way functions against O-oracle Boolean circuits and cryptographically-secure pseudorandom generators.

Definition 2.10. A family of functions $\{f\} : \Sigma^n \to \Sigma^{\ell(n)}$ is *one-way, $s(n)$-secure against oracle* O, if the function f is uniformly computable in time polynomial in n and for every non-uniform circuit C^O of size at most $s(n)$ and for sufficiently large n,

$$\Pr_{x \in \Sigma^n}[C^O(f(x)) \in f^{-1}(f(x))] \leq \frac{1}{s(n)}.$$

Given any function $f : \{0,1\}^* \to \{0,1\}$ and oracle O, the *circuit complexity of f relative to O-oracle at length n*, denoted by $C_f^O(n)$, is the size of the smallest O-oracle circuit that computes f on every input of size n. We say that a function $f : \{0,1\}^* \to \{0,1\}$ has circuit complexity $s(n)$ relative to an oracle O, if for all but finitely many n, $C_f^O(n) \geq s(n)$.

Definition 2.11. A *pseudorandom generator* is a function $G : \Sigma^{m(n)} \to \Sigma^n$ such that for every circuit C of size at most $O(n)$,

$$\left| \Pr_{x \in \Sigma^n}[C(x) = 1] - \Pr_{y \in \Sigma^{m(n)}}[C(G_n(y)) = 1] \right| \leq \frac{1}{8}.$$

Given an oracle O, G is said to be *secure against O-oracle* if the above inequality holds for all O-oracle circuits C^O of size at most $O(n)$, for almost all n.

All known constructions of the cryptographically-secure pseudorandom generators are based on some hardness assumptions on the circuit complexity of a function. We need the following result due to Klivans and van Melkebeek [KvM02] (Theorem 3.4 in their paper.)

Theorem 2.2 ([KvM02])**.** *There exists a positive constant c such that the following holds for any oracle O, any function $f \in$ EXP, and any time constructible function $\ell : \mathbb{N} \to \mathbb{N}$: If for every n, the O-oracle circuit complexity of f at length $\ell(n)$ is at least n^c, then there is a pseudorandom generator $G : \Sigma^{O(\ell^2(n))} \to \Sigma^n$ that is secure against O-oracle. The running time of G is $2^{\ell^{2d}(n)}$ for some constant $d > 0$.*

We use the following instantiation of the above result obtained by taking $\ell(n)$ to be $(c \log n)^{1/\epsilon}$.

Theorem 2.3 ([KvM02])**.** *Let O be any language. If there is a constant $\epsilon > 0$ and a function f in* EXP *with circuit complexity at least 2^{n^ϵ} relative to O-oracle, then there is a constant $a > 0$ and a pseudorandom generator $G : \Sigma^{\log^a n} \to \Sigma^n$ that is secure against O-oracle. The running time of G is $2^{\log^b n}$ for some constant $b > 0$.*

2.6 Multipass Machines

In the last two chapters of this dissertation, we study the probabilistic space-bounded machines that can access random bits multiple times. For such machines, (often called *offline model*) the random bits appear on a special read-only tape called the *random tape*. In addition to the random tape, these machines have one read-only input tape and few read-write work tapes, as standard in space-bounded machine models. The total space used by the work tapes signify the space bound of such a machine. We also assume that all the probabilistic space-bounded machines halt in time at most exponential in their space bounds. Thus, the number of random bits used by them is at most exponential in their space-bounds. More formally, our multipass machines are defined as follows:

Definition 2.12. A language L is in $k(n)$-*pass* BPSPACE$[s(n), r(n)]$ if there exists an $O(s(n))$-space bounded probabilistic Turing machine M such that on any input x of length n:

(a) M makes $k(n)$ passes over the random tape, during each pass it accesses the contents of random tape in a *one-way* manner,

(b) M uses at most $O(r(n))$ random bits, and

(c) the probability that M incorrectly decides x is at most $1/3$.

In our notation, BPL = 1-*pass* BPSPACE$[\log n, n^{O(1)}]$. In Chapter 5, we observe that a constant factor in the number of passes does not add power to the model and hence in our notation, $O(1)$-*pass* BPSPACE$[\log n, n^{O(1)}]$ is also same as BPL. Also, the result of Nisan and Zuckerman [NZ96] can be stated as BPSPACE$[s(n), s^{O(1)}(n)] =$ DSPACE$(s(n))$.

Since we assume that every space-bounded probabilistic machine halts on all settings of random tape on all inputs, the running time of the multipass machine is bounded by $2^{O(s(n))}$ where $s(n)$ is the space bound. Thus, this machine can only access random tape of length $2^{O(s(n))}$. Indeed, when the number of random bits is exponential in space, *i.e.*, $r(n) = 2^{O(s(n))}$, we simply write the above class as $k(n)$-*pass* BPSPACE$(s(n))$. Further, when the the space of the $k(n)$-pass BPSPACE machine is bounded by $O(\log n)$, we simply write the class as $k(n)$-*pass* BPL$[r(n)]$.

2.7 Expander Graphs

Definition 2.13. An undirected multigraph G_N on N vertices is called a γ-*expander* if for every set of vertices S of size at most $N/2$, it has at least $(1+\gamma)|S|$ neighborhood vertices. We define a family of γ-expanders by $\{G_N\}_{N\geq 1}$, where each G_N is a γ-expander. We say that an expander is d-regular, if there exist a constant $\gamma > 0$ such that G_N is γ-expander and every vertex in G_N has degree d. We index each of the d neighbors for each node by the integers from 1 to d.

We will use the following constant-degree expander construction due to Margulis [Mar75] and Gabber-Galil [GG81].

Theorem 2.4. *Let N be an integer, $\mathbb{Z}_m$ be the set of integers modulo m, and $m = \sqrt{N}$. Consider an undirected multigraph G_N whose vertices and edges are defined as follows:*

- *each vertex is represented by a pair* $(x, y) \in \mathbb{Z}_m \times \mathbb{Z}_m$ *and*
- *each vertex* (x, y) *is connected to only five other vertices:* $(x, y), (x, x + y), (x, x + y + 1), (x + y, y), (x + y + 1, y)$ *(under modulo* m *operations).*

Then G_N *is a 5-regular expander.*

2.8 Arithmetic Circuits and Polynomial Identity Testing

Now we define arithmetic circuits as another model of computation and the corresponding polynomial identity testing problem. Let $\mathbf{R}$ denote a ring, $\mathbb{Z}$ denote the set of integers, and $\mathbb{Q}$ denote the set of rationals.

Definition 2.14. An *arithmetic circuit* C with ℓ-inputs over $\mathbf{R}$, denoted C_ℓ, is a labeled, directed acyclic graph such that

(a) there exist exactly one node whose out-degree is zero (output node),

(b) there exist at least ℓ nodes whose in-degree is zero (input nodes),

(c) all other nodes are labeled with operations $+$ and $\times$, and

(d) input nodes are labeled either by $x_1, \cdots, x_\ell$ or by constants from $\mathbf{R}$.

Size, Degree, and Depth

The *size* of an arithmetic circuit is the size of the underlying labeled graph consisting of the nodes and edges. An arithmetic circuit C_ℓ naturally and succinctly represents a multivariate polynomial $p(x_1, \cdots, x_\ell)$. An important parameter is its degree. In general, the degree of the polynomial represented by C_ℓ could be exponential in size of C_ℓ. We consider arithmetic circuits that represent *low-degree polynomials*. We say an arithmetic circuit C_ℓ of size n has *low-degree* if the polynomial represented by C_ℓ has total degree at most n.

Let C_ℓ be an arithmetic circuit of size n, that has $\ell \leq n$ inputs. Even though the polynomial represented by C_ℓ is over ℓ variables, we can also view it as the following n-variate polynomial: $p(x_1, \cdots, x_\ell) \times (x_{\ell+1} \cdots x_n)^0$. Thus, from now onward, we will assume that n represents both the

size of the arithmetic circuit C_n as well as the number of variables in the n-variate polynomial represented by it.

Depth of an arithmetic circuit is the length of the longest path from any input gates to the output gate. Fan-in of a gate g is the number of input gates to g. We are interested in arithmetic circuits with bounded depth and unbounded fan-in. Let $k > 0$ be a constant and C_n be an unbounded fan-in arithmetic circuit with depth k. By our notation, C_n is a circuit of size n representing an n-variate polynomial $p(x_1, \cdots, x_n)$. Note that the degree of $p(x_1, \cdots, x_n)$ is bounded by a polynomial $q(n)$ ($q(n)$ may depend on k). By padding C_n with few additional gates (for example, considering a circuit representing an identically zero polynomial and feeding it to an addition gate), we can obtain a circuit D of size $q(n)$ that is equivalent to the circuit C_n. This padding can be done in logspace. Thus C_n is equivalent to an arithmetic circuit D of size $q(n)$ and degree $q(n)$. Hence D represents a low degree polynomial. Even though D represents a polynomial over n variables, we can still view it as a polynomial over $q(n)$-variables. Thus, without loss of generality, we make the following assumption in the rest of this dissertation: *A depth-k arithmetic circuit C of size n has low-degree and represents an n-variate polynomial.*

Definition 2.15. Low-degree Polynomial Identity Testing (PIT) over ring $\mathbf{R}$ is the following computational problem: Given a low-degree arithmetic circuit over $\mathbf{R}$, determine if the multivariate polynomial represented by it is identically zero. Let $k > 0$ be a constant. *Depth-k PIT* over $\mathbf{R}$ is the problem of determining whether the polynomial represented by a depth k, arithmetic circuit (over $\mathbf{R}$) is identically zero.

2.9 Hitting Sets and Black-box Derandomization

We use two different notions of hitting sets for the space-bounded probabilistic class $\mathcal{C}$ in this dissertation.

Definition 2.16. We say that $\mathcal{C}$ has *polynomial-time computable hitting sets* if for every machine M in $\mathcal{C}$, there exists a constant $c > 0$ and a family of functions $\{f_n\}_{n\geq 1}$, $f_n : \Sigma^{c \log n} \to \Sigma^n$, uniformly computable in time polynomial in n, such that for every $n \geq 0$ and for every $x \in \Sigma^n$,

$$x \in L \Leftrightarrow \exists s \in \{0,1\}^{c \log r(n)} \text{ so that } M(x, f_{r(n)}(s)) = 1$$

where $r(n)$ denotes the number of random bits used by M on inputs of length n.

The above type of derandomization applies to a complexity class and is standard in the study of derandomization. The next definition, due to Agrawal and Vinay [AV08], applies to the specific problem PIT defined in the previous section.

Definition 2.17 ([AV08])**.** Let $k > 0$ be a constant. Depth-k, PIT over ring $\mathbf{R}$ has a *polynomial-time computable hitting set* if there exists a family of functions $\{g_n\}_{n\geq 1} : \mathbb{N} \to (\mathbf{R}^n)^*$ uniformly computable in time polynomial in n such that for every depth-k arithmetic circuit C_n (over $\mathbf{R}$) of size n, C_n is identically zero if and only if $C_n(a_1, \cdots, a_n) = 0$, for every tuple $(a_1, \cdots, a_n)$ in the range of $g_n(1^n)$.

Next, we define what we mean by black-box derandomization for the complexity class $\mathcal{C}$.

Definition 2.18. We say that $\mathcal{C}$ has polynomial-time *black-box derandomization* if for every machine M in $\mathcal{C}$, there exists a constant $c > 0$ and a family of functions $\{f_n\}_{n\geq 1}, f_n : \Sigma^{c \log n} \to \Sigma^n$ uniformly computable in time polynomial in n, such that for every $n \geq 0$ and for every $x \in \Sigma^n$,

$$\left| \Pr_{r \in U_{r(n)}} [M(x, r) = 1] - \Pr_{s \in U_{c \log r(n)}} [M(x, f_{r(n)}(s))] = 1 \right| \leq \frac{1}{n},$$

where $r(n)$ denotes the number of random bits used by M on inputs of length n and U_m denotes the uniform distribution over Σ^m.

The family of functions $\{f_n\}$ in the above definition is called the *pseudorandom generator* for the complexity class $\mathcal{C}$.

CHAPTER 3. SEPARATION OF NP-COMPLETENESS NOTIONS

The concept of NP-completeness plays a pivotal role in understanding the computational difficulty of various problems that arise in practice. The notion of polynomial-time reductions is at the heart of defining whether a problem is NP-complete or not. Interestingly, there is no single notion of polynomial-time reduction. There is a wide spectrum of such notions, for example, many-one reduction, bounded truth-table reduction, truth-table reduction, Turing reduction, etc. Each of these polynomial-time reductions defines a notion of completeness. Given a polynomial-time reduction r, a language L is called r-complete for NP if the language L is in NP and every language in NP r-reduces to L. Whether these reductions, particularly many-one reduction and Turing reduction, are different in NP has drawn a keen interest in the complexity theory community. Ladner, Lynch, and Selman [LLS75] showed that if the two completeness notions arising from the many-one and Turing reductions are different in NP then the complexity class P is different from its non-deterministic counterpart NP. They famously conjectured that if P $\neq$ NP, then the Turing and many-one completeness notions are different in NP. Their conjecture prompted a large body of research on differences among completeness notions for various complexity classes.

Understanding the differences between many-one reductions and Turing reductions is one of the fundamental problems in complexity theory. Compared to many-one reductions, our knowledge about Turing reductions is limited. Extending certain assertions that are known to be true for many-one reductions to the case of Turing reductions yield much sought after separation results in complexity theory. For example, it is known that polynomial-time many-one complete sets for EXP are not sparse [TFL93]. Extending this result to the case of Turing reductions implies that EXP does not have polynomial-size circuits. In the context of resource-bounded measure, it is known that "small span theorem" holds for many-one reduc-

tions. Establishing a similar result for Turing reductions separates EXP from BPP [JL95]. In addition, Turing reductions are crucial to define the polynomial-time hierarchy.

Before we mention the progress on the aforementioned conjecture, it is important to understand the differences between separating various polynomial-time reductions in NP and separating various NP-completeness notions. Simon and Gill [SG77] showed that if P $\neq$ NP, there is a language in NP that is hard for NP via Turing reductions, but is not hard for NP via many-one reductions. Thus the conjecture of Ladner, Lynch, and Selman holds for NP-hardness, but is not known to hold for NP-completeness. Further, Selman [Sel79] showed that if NE $\cap$ co-NE does not equal E, then there exist languages A and B in NP such that A polynomial-time Turing reduces to B, but does not polynomial-time many-one reduce to B. Aida, Schuler, Tsukiji, and Watanabe [ASTW01] showed a similar result for the average-case world; if P does not equal NP, then there are distributional problems (A, μ_A) and (B, μ_B) in DistNP such that (A, μ_A) Turing reduces to (B, μ_B) but does not many-one reduce to (B, μ_B). Pavan and Selman [PS04] showed that even under a weaker hypothesis, that is, existence of tally languages in UP $-$ P, it's possible to separate truth-table reduction from bounded truth-table reduction and Turing reduction from truth-table reduction in NP. Moreover, the differences between Turing and truth-table reductions have been studied extensively in the context of random self-reductions and coherence [FFLS92, FFLN96, HNOS96, BL99]. For example, Feigenbaum, Fortnow, Lund, and Spielman [FFLS92] showed that if nondeterministic triple exponential time is not in bounded-error, probabilistic triple exponential time, there exists a function in NP that is Turing random self-reducible, but not truth-table random-self reducible.

Previous Work

It is interesting to note that the question whether many-one completeness and Turing completeness are different has been completely resolved for the complexity classes EXP and NEXP. Works of Ko and Moore [KM81] showed that there is a language that is Turing complete for EXP, but not many-one complete for EXP. Watanabe [Wat87] went one step further and showed that each of the 1-truth-table, bounded truth-table, truth-table, and Turing completeness notions are different from each other for EXP. On the contrary, Homer, Kurtz,

and Royer [HKR93] showed that the completeness notions induced by many-one and 1-truth-table reductions are in fact identical for EXP. Separation results similar to Watanabe's were obtained for the complexity class NEXP as well by Buhrman, Homer, and Torenvliet [BHT91]. Survey articles [BT94, Hom97] contain the details and proof ideas of these results.

The progress towards separating Turing and many-one completeness for NP, however, has been very slow. One of the early results in this direction was by Longpre and Young [LY90] who showed that for every polynomial $t(n)$, there is a Turing complete language in NP that is not many-one complete via a $t(n)$ time many-one reduction. However, this still does not solve the aforementioned problem as the definition of many-one reduction allows it to run for time any polynomial in n. The first result that achieved a separation between Turing and many-one completeness for NP, under a reasonable hypothesis, is due to Lutz and Mayordomo [LM96]. They showed that if NP does not have P-measure 0 (known as *measure hypothesis*), then Turing completeness for NP is different from many-one completeness [1]. Subsequently, completeness notions under other polynomial-time reductions too were shown to be different under even weaker hypotheses. Ambos-Spies and Bentzien [ASB00] achieved a finer separation (by separating almost all of the bounded truth-table completeness notions from each other and truth-table completeness) for NP under a weaker hypothesis known as *genericity hypothesis*, which states that NP has a P-generic language. This is still the weakest known hypothesis that separates each of the k-truth-table completeness notions for NP. Both the above hypotheses involve the stochastic properties of NP. Pavan and Selman [PS02] showed that the separations can be achieved under combinatorial properties of NP as well. They separated Turing and many-one completeness for NP under the assumption that NP $\cap$ co-NP contains a 2^{n^ϵ}-bi-immune language [2] (called NP *machine hypothesis*). Pavan and Selman [PS04] showed that the existence of a 2^{n^ϵ}-bi-immune language in NP suffices to separate 3-truth-table from many-one completeness for NP. The separation between any two adaptive reductions under combinatorial hypotheses were still open, until they further separated truth-table completeness

[1] In fact, they achieved a stronger result: Under measure hypothesis, 3-truth-table reduction is different form many-one reduction for NP.

[2] An infinite language L is called 2^{n^ϵ}-*immune* if no infinite subset of L belongs to DTIME(2^{n^ϵ}). A language L is called 2^{n^ϵ}-*bi-immune* if both L and $\overline{L}$ are 2^{n^ϵ}-immune.

from Turing completeness for NP under the hypothesis that UP $\cap$ co-UP has a 2^{n^ϵ}-bi-immune language (called UP *machine hypothesis*). They also improved their previous result to show that NP machine hypothesis is able to separate 1- and 2-truth-table completeness for NP as well. For the interesting relationships among the aforementioned hypotheses, refer to the survey of Pavan [Pav03] and in-depth study by Hitchcock and Pavan [HP08].

Our Contribution

All of the above mentioned hypotheses are known as *almost everywhere hardness hypotheses*. Informally, these hypotheses assert that there exists a language in NP such that every algorithm that decides L must take more than subexponential time on *all but finitely many inputs*. Even though we believe that NP is subexponentially hard, we do not have any candidate languages in NP that are almost everywhere hard. All known natural problems have an infinite set of instances that can be decided in polynomial time. Thus these hypotheses are considered "strong hypotheses". It was open whether a separation among NP-completeness notions can be achieved using a *worst-case hardness* hypothesis (such as P $\neq$ NP, or NE $\neq$ E) or *average-case hardness* hypothesis (such as the existence of one-way functions). The first work in this direction was by Hitchcock, Pavan, and Vinodchandran [HPV08] who showed the separation between Turing and many-one completeness for NP under two certain average-case hardness hypotheses called *partial bi-immunity* hypothesis and *scaled dimension* hypothesis. Later, Gu, Hitchcock, and Pavan [GHP11] used a combination of average-case and worst-case hardness hypotheses to separate Turing completeness from truth-table completeness for NP. The average case hypothesis is "there exist one-way permutations" and the worst case hypothesis is "there exists a language in NEEE $\cap$ co-NEEE that cannot be solved in deterministic triple exponential time with logarithmic advice". But the separation between any two polynomial-time completness notions for NP purely under worst-case (in uniform setting) hardness hypothesis was still open.

The main contribution in this chapter is a separation of Turing completeness from truth-table completeness (hence, many-one completeness) for NP under only a worst-case hardness hypothesis. This is the first result of this nature.

Organization of this Chapter

The rest of this chapter is organized as follows. We formally state the main theorem of this chapter in the next section. Section 3.2 provides the proof of the main theorem. Section 3.3 explains the power of our hypothesis by relating it to the previously studied hypotheses, mentioned before. Finally, Section 3.4 concludes this chapter with future direction on this work.

Notation

The functions of the form $2^{2^{f(n)}}$, that are used in many places throughout this chapter, are not visually appealing; from now we represent such functions as $\tau(f(n))$. Let $\tau : \mathbb{N} \to \mathbb{N}$ be a function defined as $\tau(n) = 2^{2^n}$. Then $\tau(\delta f(n))$ represents $2^{2^{\delta f(n)}}$. We use $\tau^\epsilon(n)$ to denote $(\tau(n))^\epsilon$.

3.1 Separation Theorem

First, we formally state our hypothesis.

Hypothesis W. There exist a constant $\delta > 0$ and an NEEXP machine N_1 accepting Σ^* that runs in time $t(n)$ such that no $2^{t(n)^\delta}$-time bounded, zero-error, probabilistic machine can compute the accepting computations of N_1. Here $t(n) = 2^{2^{n^c}}$ for some constant $c > 1$.

In this chapter we prove the following main result.

Theorem 3.1. *If Hypothesis* W *holds, then there is a Turing complete language for* NP *that is not truth-table complete for* NP*.*

Before we provide a formal proof, we first describe a proof outline.

Proof Sketch

Our proof proceeds in four steps. Note that Hypothesis W is a "worst-case hardness hypothesis". This means that for every probabilistic, $2^{t(n)^\delta}$-time bounded, machine Z_1 there exists *infinitely many* inputs x such that the probability that $Z_1(x)$ computes an accepting computation of $N_1(x)$ is very small. This is equivalent to the following: there exist *infinitely many* input

lengths n for which there exists *at least one string x of length n* so that the probability that $Z_1(x)$ is an accepting computation of $N_1(x)$ is very small. In the first step (Section 3.2.1), we amplify the hardness of N_1 and obtain an NEEXP machine N_2 with the following property: For every $2^{t(n)^\delta}$-time bounded, probabilistic machine Z_2, there exist *infinitely many input lengths n* at which *for every string x of length n* the probability that $Z_2(x)$ is an accepting computation of $N_2(x)$ is small.

In the second step (Section 3.2.2), we first define a padding function $pad : \Sigma^* \to \mathbb{N}$. Via standard padding arguments we obtain an NP-machine N running in time $p(n)$ that accepts a tally set $T = \{0^{pad(x)} \mid x \in \Sigma^*\}$. For $\ell \geq 0$, let $T_\ell = \{0^{pad(x)} \mid x \in \Sigma^\ell\}$. The NP-machine N has the following hardness property: For every $f(n)$-time bounded, probabilistic machine Z (for an appropriate choice of f) there exist infinitely many integers ℓ such that Z fails to compute accepting computations on *every string* from T_ℓ.

Using the NP-machine N, we define the Turing complete language L in step three (Section 3.2.3). The language L is formed by taking disjoint union of two NP languages L_1 and L_2. The language L_1 consists of tuple of the form $\langle x, a\rangle$ so that $x \in C$ (for some NP-complete language C), and a is an accepting computation of $N(0^n)$ (for some n that depends on x). In L_2, we encode accepting computations of N using a P-selective set. It follows that C can be Turing reduced to L by first obtaining an accepting computation of N (by making queries to L_2) and then by making one query to L_1. The idea of forming L_1 is borrowed from [PS02], and encoding accepting computations of an NP-machine as a P-selective sets is well known. For example see [HNOS96].

Finally, in step four (Section 3.2.4), we show that if L is truth-table complete, then there is a probabilistic machine Z such that for every ℓ there exists atleast one string in T_ℓ so that Z computes an accepting computation of N on that string with high probability. Using this, we in turn show that there exists a probabilistic machine Z_2 so that for every input length ℓ, there exists atleast one string $x \in \Sigma^\ell$ such that $Z_2(x)$ outputs an accepting computation of the NEEXP machine $N_2(x)$. This will be a contradiction. Technically, this step is the most challenging part of the proof.

3.2 Proof of Main Theorem

This section provides the proof of the main theorem in this chapter. The first step is to amplify the hardness of the NEEXP machine N_1 from the hypothesis to obtain a new NEEXP machine N_2.

3.2.1 Hardness Amplification

Lemma 3.1. *Suppose that the hypothesis* W *holds. Then there exist an* NEEXP *machine* N_2 *accepting* Σ^* *and running in time* $O(2^n\tau(n^c))$ *and a constant* $\beta < \delta$ *such that for every probabilistic machine* Z_2 *that runs in time* $\tau(\beta 2^{n^c})$*, there exist* infinitely many input lengths $n > 0$ *such that for* every $x \in \Sigma^n$*,*

$$\Pr[Z_2(x) = \text{ an accepting computation of } N_2(x)] \leq 1/4.$$

Proof. Let N_1 be the nondeterministic machine from Hypothesis W whose running time is bounded by $O(t(n))$, where $t(n) = \tau(n^c)$ (for some $c > 1$). Length of every accepting computation of $N_1(x)$ is bounded by $O(t(|x|))$. Consider a machine N_2 that behaves as follows: On an input x of length n, it runs $N_1(y)$ on every string y of length n (in a sequential manner). The running time of N_2 is $O(2^n \times t(n))$. Since N_1 accepts Σ^*, the machine N_2 also accepts Σ^*. We claim that N_2 has the required property.

Suppose not. Then there is a probabilistic machine Z_2 that runs in time $O(\tau(\beta 2^{n^c}))$ (for some $\beta < \delta$) such that for for all but finitely many n, there exists a string $y_n \in \Sigma^n$ such that

$$\Pr[Z_2(y_n) = \text{ an accepting computation of } N_2(y_n)] > 1/4.$$

By the definition of N_2, the accepting computation of $N_2(x)$ encodes the accepting computation of $N_1(y)$ for every y whose length is same as the length of x. Consider a machine Z_1 that on any input x of length n behaves as follows:

> It runs $Z_2(y)$ on every y of length n. It verifies that the output of $Z_2(y)$ is an accepting computation of $N_2(y)$, and if the verification succeeds, then it extracts the accepting computation of $N_1(x)$ and outputs it. If $Z_2(y)$ does not output an accepting computation of $N_2(y)$, then Z_1 outputs $\perp$.

Let x be any input of length n. By our assumption, there exists a $y_n \in \Sigma^n$ such that $Z_2(y_n)$ outputs an accepting computation of $N_2(y_n)$ with probability at least $1/4$. The above machine clearly runs $Z_2(y_n)$ on input x. Since an accepting computation of $N_1(x)$ can be retrieved from an accepting computation of $N_2(y_n)$, the above machine outputs an accepting computation of $N_1(x)$. Thus for all but finitely many n, for every $x \in \Sigma^n$, Z_1 outputs an accepting computation of $N_1(x)$ with probability at least $1/4$. The running time of Z_1 is clearly $O(2^n \times \tau(\beta 2^{n^c}))$, which is less than $\tau(\delta 2^{n^c})$ (as $\beta < \delta$). This contradicts Hypothesis W. □

3.2.2 Defining an NP Machine

Now we define an NP machine N from the above NEEXP machine N_2. Fix $\epsilon < \beta$. Consider the following padding function $pad : \Sigma^* \to \mathbb{N}$, defined by

$$pad(x) = \lfloor \tau^{\epsilon}(\log^c r_x) \rfloor,$$

where r_x is the rank of string x in the standard lexicographic order of Σ^*, so that $2^{\ell} - 1 \leq r_x \leq 2^{\ell+1} - 2$, for every $x \in \Sigma^{\ell}$. Note that pad is 1-1 and so $pad^{-1}(n)$ (if exists) is well defined. To keep the calculation simple, we drop the floors henceforth. Now we define the following tally language based on the padding function:

$$T = \left\{ 0^{pad(x)} \mid x \in \Sigma^* \right\},$$

Our NP machine N that accepts a tally language behaves as follows: On input 0^m, it computes $x = pad^{-1}(m)$. Upon finding such x, it runs $N_2(x)$. If no such x is found, then N rejects. Note that $|x| < (\log \log m^{2/\epsilon})^{1/c}$. So running time of N is bounded by $m^{3/\epsilon}$. Thus N is an NP machine. Note that N accepts the tally language T.

3.2.3 Turing-complete Language

At this point, we are ready to define the language L in NP that we prove to be Turing complete, but not truth-table complete for NP.

Let L_T be the range of the padding function pad.

$$L_T = \{\tau^{\epsilon}(\log^c i) \mid i \in \mathbb{N}\}.$$

By definition, N accepts only those tally strings whose length is in the set L_T. We use n_i to denote $pad(i)$. Given a length $n \in L_T$, define a_n to be the lexicographically maximum accepting computation of $N(0^n)$. Let a be the infinite binary string $a_{n_1}a_{n_2}a_{n_3}\cdots$ where $n_i \in L_T$ and $n_1 < n_2 < n_3 < \cdots$. Let $|a_n|$ denotes the length of the accepting computation a_n. Let SAT$'$ consist of the SAT formulas with lengths only in L_T, *i.e.*,

$$\text{SAT}' = \text{SAT} \cap \{x \in \Sigma^* \mid |x| \in L_T\}.$$

Since there exists a polynomial p such that $n_{i+1} \leq p(n_i)$, it can be shown via padding that SAT many-one reduces to SAT$'$ and thus SAT$'$ is NP-complete.

We define L_1 and L_2 as follows.

$$L_1 = \left\{\langle \phi, u\rangle \mid |\phi| = n,\ u \text{ is an accepting computation of } N \text{ on } 0^n,\ \phi \in \text{SAT}'\right\}$$

$$L_2 = L(a) = \{z \mid z < a\},$$

where $<$ is the dictionary order with $0 < 1$. Then our Turing-complete language L is the disjoint union of L_1 and L_2, *i.e.*,

$$L = L_1 \oplus L_2 = 0L_1 \cup 1L_2.$$

Note that both L_1 and L_2 are in NP, and so is L.

Lemma 3.2. *L is $\leq_T^{\text{P}}$-complete for* NP*.*

Proof. Reduce SAT$'$ to L: On input ϕ of length n, make adaptive queries to L_2 to find a_n. Accept ϕ if and only if $\langle \phi, a_n\rangle \in L_1$. □

3.2.4 L is Not Truth-table Complete

Now we show that L is not truth-table complete for NP. Before we proceed with the proof, we provide the intuition behind the proof.

Proof Sketch

Suppose that L is truth-table complete. We achieve a contradiction by exhibiting a procedure to compute accepting computations of NEEXP machine N_2. Since the NP-machine N is

padded version of N_2, it suffices to compute the accepting computations of N. We partition T into sets $T_1, T_2, \cdots$, where $T_\ell = \{0^{pad(x)} \mid x \in \Sigma^\ell\}$. Clearly, $|T_\ell| = 2^\ell$ and $T = \bigcup_\ell T_\ell$. Note that an accepting computation of $N_2(x)$ can be computed by computing an accepting computation of $N(0^{pad(x)})$, and if $|x| = \ell$, then $0^{pad(x)} \in T_\ell$.

Recall that N_2 has the following property: For every probabilistic machine Z_2 that attempts to compute its accepting computations, there exist infinitely many input lengths ℓ and Z_2 fails on every string at those lengths. Informally, this translates to the following hardness property of N: For every probabilistic machine Z that attempts to compute accepting computations of N, there exist infinitely many integers ℓ such that Z fails on every string from T_ℓ. Thus to achieve a contradiction, it suffices to exhibit a probabilistic procedure Z such that for all but finitely many ℓ, Z outputs an accepting computation of $N(0^n)$ for some $0^n \in T_\ell$, with non-negligible probability. We will now (informally) describe how to compute accepting computations of N.

For the sake of simplicity, let us first assume that the NP machine N has exactly one accepting computation on every input from T. The first task is to define a set S that encodes the accepting computations of the machine N. One way to define S as

$$S = \{\langle 0^n, i\rangle \mid i\text{th bit of accepting computation of } N(0^n) \text{ is } 1\}.$$

Since we assumed that N has exactly one accepting computation, deciding S is equivalent to computing accepting computations of N. Since S is in NP, there is a truth-table reduction from S to L. We make another simplifying assumption that all queries are made to L_1 part of L. Consider an input $\langle 0^n, i\rangle$ where $0^n \in T_\ell$ (for some $\ell > 0$). All the queries produced on this input are of the form $\langle \phi, u\rangle$. It is easy to check if u is an accepting computation of $N(0^m)$ for some m. If u is not an accepting computation, then $\langle \phi, u\rangle$ does not belong to L, and thus it is easy to decide the membership of $\langle 0^n, i\rangle$ in S. Suppose that u is an accepting computation of $N(0^m)$ for some m. Then there are two cases. First case is the "short query" case, where m is much smaller than n. In this case $\langle \phi, u\rangle$ is in L_1 only when $|\phi|$ equals m and $\phi \in \text{SAT}'$. Since $m << n$, we can decide whether $\phi \in \text{SAT}'$ using a brute force algorithm in time $O(2^m)$, this in turn enables us to decide the membership of $\langle 0^n, i\rangle$ in S. Thus if all the queries are small, we can decide the memberships of $\langle 0^n, i\rangle$ (for all i), and thus can compute accepting computation

of $N(0^n)$. The second case is the "large query" case: Suppose that for some query, m is not much smaller than n. In this case, we are in the following scenario: The reduction outputs accepting computation of $N(0^m)$ and m is somewhat large. In this case, we argue that for an appropriate choice of n, 0^m also lies in T_ℓ. This will enable us to design a procedure that outputs accepting computation of some string from T_ℓ. This is the gist of the proof.

The above argument assumed that N has exactly one accepting computation, which may not be true in general. We get around this problem by applying Valiant-Vazirani lemma [VV86] to isolate one accepting computation. Thus the definition of our language S will involve the use of isolation lemma. It is also very much possible that the reduction makes queries to L_2 also. Recall that L_2 is a P-selective set and it is known that if an NP-language A reduces to a P-selective set, then A must be "easy" [Tod91a, Sel79]. We use this in combination with the above mentioned approach. A technically involved part is to define the correct notion of "small" and "large" queries. There is a delicate interplay among the choice of pad function, notion of small query, and the runtime of probabilistic machine that computes the accepting computations of N.

Formal Proof

We now formalize the above intuition into the following lemma.

Lemma 3.3. *L is not $\leq^{\mathrm{P}}_{tt}$-complete for* NP*.*

Proof. For the sake of contradiction, assume that L is truth-table complete for NP. Consider the following set S.

$$S = \{\langle 0^n, k, r_1, r_2, \ldots, r_k, i\rangle \mid n \in L_T, 1 \leq k \leq |a_n|, r_i \in \Sigma^{|a_n|}, \text{ there is a } u \text{ such that } u \text{ is an accepting computation of } N(0^n), u \cdot r_1 = u \cdot r_2 = \cdots = u \cdot r_k = 0, \text{ and the } i\text{th bit of } u = 1\},$$

where $u \cdot r_i$ denotes the inner product over GF[2].

It is easy to see that S is in NP. Since L is $\leq^{\mathrm{P}}_{tt}$-complete for NP, S is $\leq^{\mathrm{P}}_{tt}$ reducible to L via polynomial time computable functions $\langle g, h\rangle$, where g is the query generator and h is the truth-table evaluator. Since g is polynomial-time computable, there exists a constant $b > 0$ such that every query generated by it is of length at most n^b.

At this point, our goal is to compute an accepting computation of N. We start with the following algorithm $\mathcal{A}$ that classifies all the queries of the query generator into two sets, "Large Query" and "Small Query".

Procedure $\mathcal{A}$

1: Input 0^n, where $n = \tau^\epsilon(\log^c i)$ for some $i \in \mathbb{N}$. Clearly, $n \in L_T$.

2: For $1 \leq j \leq n^2$ repeat the following:

- Pick k^j uniformly at random from $\{1, \cdots, |a_n|\}$.
- Pick each of $r_1^j, r_2^j, \ldots, r_{k_j}^j$ uniformly at random from $\Sigma^{|a_n|}$.

3: Let Q^j be the set of queries generated by g on inputs $\langle 0^n, k^j, r_1^j, \cdots, r_{k_j}^j, i\rangle$, $1 \leq i \leq |a_n|$. Compute Q^j for $1 \leq j \leq n^2$ and set $Q = \bigcup_j Q^j$. Note that the length of each query is bounded by n^b.

4: Partition Q into two sets Q_1 and Q_2 such that Q_1 is the set of all queries to L_1 and Q_2 is the set of all queries to L_2.

5: If Q_1 contains a query $\langle \phi, u_t\rangle$ for some t, where u_t is an accepting computation of $N(0^t)$ and

$$t > \tau^\epsilon(((\log\log n^{b/\epsilon})^{1/c} - 1)^c),$$

then print u_t, output "Large Query", and halt.

6: Otherwise, output "Small Query" and halt.

It is clear that the algorithm $\mathcal{A}$ runs in time polynomial in n.

Before we give our probabilistic algorithm to compute the accepting computations of N, we bound the probabilities of certain events of interest. T is partitioned into sets $T_1, T_2, \cdots$ each of cardinality 2^ℓ, where

$$T_\ell = \left\{0^{\tau^\epsilon(\log^c r_x)} \mid x \in \Sigma^\ell\right\}.$$

Fix $\ell > 0$. For a fixed $0^n \in T_\ell$ and j, $1 \leq j \leq n^2$, let $\mathcal{E}_{n,j}$ denote the following event:

There exists *exactly one* u such that

- u is an accepting computation of $N(0^n)$,

- $u \cdot r_1^j = u \cdot r_2^j = \cdots = u \cdot r_{k_j}^j = 0$.

By Valiant-Vazirani, we have that $\Pr[\mathcal{E}_{n,j}] \geq \frac{1}{n^2}$. Let $\mathcal{E}_n$ denote the event that for some j, $1 \leq j \leq n^2$, $\mathcal{E}_{n,j}$ occurs. The probability of $\mathcal{E}_n$ is at least $1 - \frac{1}{2^{n^2}}$. Finally, let $\mathcal{E}_\ell$ denote the event that for every $0^n \in T_\ell$, the event $\mathcal{E}_n$ occurs. Again, we have that $\Pr[\mathcal{E}_\ell] \geq 1 - \frac{1}{2^\ell}$.

Thus for every ℓ, the probability that the event $\mathcal{E}_\ell$ occurs is very high. Fix an ℓ. From now on, we assume that the event $\mathcal{E}_\ell$ has occurred.

Now our goal is to arrive at the machine that computes an accepting computation of atleast one string from T_ℓ. For this we will analyze the behavior of the above algorithm on a specific string $0^{V_\ell} \in T_\ell$, where

$$V_\ell = \tau^{\epsilon/b}(\log^c(2^{\ell+1} - 2)).$$

We stress that this *unique* string 0^{V_ℓ} depends only on the length ℓ. When we run algorithm $\mathcal{A}$ on 0^{V_ℓ}, either it outputs "Large Query" on it, or it outputs "Small Query".

Lemma 3.4 (Key Lemma)**.** *One of the following holds.*

1. *If $\mathcal{A}$ outputs "Small Query" on 0^{V_ℓ}, then there is an algorithm $\mathcal{B}_1$ that on input 0^{V_ℓ} runs in time polynomial in $\tau(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c})$, and correctly outputs an accepting computation of $N(0^{V_\ell})$.*

2. *If $\mathcal{A}$ outputs "Large Query" on 0^{V_ℓ}, there exist an algorithm $\mathcal{B}_2$ such that for every string in T_ℓ it runs in time polynomial in V_ℓ, and there exists a $0^t \in T_\ell$ for which $\mathcal{B}_2(0^t)$ outputs an accepting computation of $N(0^t)$.*

We defer the proof of this lemma to the end of this section and complete the proof of main theorem. Next, we describe a probabilistic machine that computes accepting computation of the NEEXP machine N_2 with non-trivial probability, with the help of $\mathcal{B}_1$ and $\mathcal{B}_2$.

Consider the probabilistic machine Z_2 that does the following on input $x \in \Sigma^\ell$:

Procedure Z_2

1: Compute V_ℓ. Run $\mathcal{A}$ on 0^{V_ℓ}.

2: If $\mathcal{A}(0^{V_\ell})$ outputs "Small Query",

- Verify if $x = pad^{-1}(V_\ell)$. If it is, then run $\mathcal{B}_1$ on 0^{V_ℓ} and if it outputs an accepting computation of $N(0^{V_\ell})$, then output that accepting computation. This is also the accepting computation of $N_2(x)$.

3: If $\mathcal{A}(0^{V_\ell})$ outputs "Large Query", do the following:

- For every string 0^i in T_ℓ, run the algorithm $\mathcal{B}_2$ on it. If it outputs the accepting computation of $N(0^t)$ for some 0^t, then verify if $x = pad^{-1}(0^t)$. If it is, then output that accepting computation. This is also the accepting computation of $N_2(x)$.

We analyze the behavior of Z_2 under the assumption that the event $\mathcal{E}_\ell$ happens. Recall that this happens with very high probability. If $\mathcal{A}(0^{V_\ell})$ outputs "Small Query", then by part (1) of Lemma 3.4, $\mathcal{B}_1$ outputs an accepting computation of $N(0^{V_\ell})$. Note that every accepting computation of $N(0^{V_\ell})$ is an accepting computation of $N_2(0^{pad^{-1}(V_\ell)})$. Since $pad^{-1}(V_\ell)$ is of length ℓ, there exists a string $x \in \Sigma^\ell$, on which Z_2 outputs an accepting computation of $N_2(x)$. Now consider the case where $\mathcal{A}(0^{V_\ell})$ outputs "Large Query", then by part (2) of Lemma 3.4, there exists a $0^t \in T_\ell$ such that $\mathcal{B}_2(0^t)$ outputs an accepting computation of $N(0^t)$. Thus Z_2 will find that 0^t through iteration. Similarly, $pad^{-1}(0^t) \in T_\ell$ is of length ℓ, thus there exists a x in Σ^ℓ on which Z_2 outputs an accepting computation of $N_2(x)$. Thus Z_2 always outputs an accepting computation of atleast one string x from Σ^ℓ.

We will now bound the runtime of Z_2. This is bounded by runtime of $A(0^{V_\ell})$, plus the runtime of $\mathcal{B}_1(0^{V_\ell})$, and the time taken in step 3 of the above algorithm. By part (1) of Lemma 3.4, the runtime of $\mathcal{B}_1(0^{V_\ell})$ is $\tau^d(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c})$ for some constant $d > 0$, which is bounded by

$$\tau^d(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c}) = \tau^d(\epsilon 2^{((\log^c(2^{\ell+1}-2)^{1/c}-1)^c}) = \tau^d(\epsilon 2^{((\log(2^\ell-1))^c}) < \tau^d(\epsilon 2^{\ell^c}).$$

Let p be a constant such that $\mathcal{A}(0^{V_\ell})$ runs in time V_ℓ^p and $\mathcal{B}_2(0^i)$, $0^i \in T_\ell$, runs in time V_ℓ^p.

Step 3 runs $\mathcal{B}_2$ on every string from T_ℓ, and there are 2^ℓ strings in T_ℓ. Thus the combined runtime of $\mathcal{A}(0^{V_\ell})$ in step 1 and step 3 is bounded by

$$2^{\ell+1}V_\ell^p = 2^{\ell+1}\tau^{p\epsilon/b}(\log^c(2^{\ell+1}-2)) \leq 2^{\ell+1}\tau^{p\epsilon/b}((\ell+1)^c) \leq \tau^q((\ell+2)^c)$$

for some constant $q > p$. Thus the total running time of Z_2 is bounded by $\tau(\beta 2^{\ell^c})$, as $\beta > \epsilon$.

Thus for all but finitely many ℓ, the machine Z_2 computes an accepting computation of $N_2(x)$ for atleast one string x from Σ^ℓ with non-trivial probability. This contradicts the hardness of NEEXP machine N_2 in Lemma 3.1. This completes the proof of Lemma 3.3. □

This also completes the proof of the main theorem.

3.2.5 Proof of Key Lemma

We prove the two parts of the Lemma 3.4 separately.

Proof of Part 1. Fix the input 0^{V_ℓ} from the hypothesis. Since we are operating under the assumption that the event $\mathcal{E}_\ell$ has occurred, there is a j, $1 \leq j \leq {V_\ell}^2$, such that $\mathcal{E}_{V_\ell,j}$ has occurred.

Recall that Q^j is the set of all queries made by the truth-table reduction $\langle g, h\rangle$ on inputs $\langle 0^n, k^j, r_1^j, \cdots, r_{k_j}^j, i\rangle$ for $1 \leq i \leq |a_{V_\ell}|$. Let Q_1^j be the set of all queries made to L_1 and let Q_2^j be the set of all queries made to the set L_2.

We will now describe how to decide answers to queries from Q_1^j. Note that each query $q \in Q_1^j$ is of the form $\langle \phi, z\rangle$, where ϕ is of length t. If z is not an accepting computation of $N(0^t)$, then $\langle \phi, z\rangle$ in not in L_1. Suppose z is an accepting computation of $N(0^t)$. Then it must be the case that $t \leq \tau^\epsilon(((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c)$. Now $\langle \phi, z\rangle$ is in L_1 if and only if $\phi \in \text{SAT}'$. Since $|\phi| = t$, this can be tested in time 2^t. Thus we can decide the membership of all queries from Q_1^j in time polynomial in $\tau(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c})$.

Now we deal with the queries to L_2; we cannot hope to decide membership of queries in Q_2^j in L_2 in general. However, note that L_2 is P-selective. Thus by Toda's lemma (Lemma 2.1) all elements of Q_2^j can be ordered as $q_1, q_2, \cdots, q_m$ and for some r, $1 \leq r \leq m$, all of $q_1, \cdots, q_r$ are in L_2 and none of $q_{r+1}, \cdots, q_m$ are in L_2. This ordering can be done in time polynomial

in m. We do not know the value of r. However, there are only m possibilities for r. For each $i \in [1, m]$ we set r as i and determine the possible membership of queries in L_2.

We are now in the following situation: For every query in Q_1^j we know the membership in L_1, and for every query in Q_2^j we know candidate memberships in L_2. Using these and evaluating h, we obtain a candidate for the ith bit of u, for $1 \leq i \leq |a_{V_\ell}|$. From this we construct a string $\hat{u}$ and if $\hat{u}$ is an accepting computation of $N(0^n)$ we output it. If u is not an accepting computation of $N(0^n)$, we proceed with next choice for r. By our assumption, there is an accepting computation of $N(0^{V_\ell})$ that is isolated. Thus this process will find that accepting computation. Note that the total time taken by this process is still some polynomial in $\tau(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c})$.

Finally, we do not know the value of j for which the isolation occurs. We repeat the above process for every possible choice of j, and there are only n^2 choices for j. Thus the total time taken by this algorithm is still a polynomial in $\tau(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c})$. □

Proof of Part 2. The length of every query generated by $\mathcal{A}(0^{V_\ell})$ is at most ${V_\ell}^b$. Then it can be seen from algorithm $\mathcal{A}$ that on input 0^{V_ℓ}, when it outputs "Large Query", it outputs an accepting computation u_t of $N(0^t)$ such that

$$\tau^\epsilon(((\log\log {V_\ell}^{b/\epsilon})^{1/c} - 1)^c) < t < {V_\ell}^b.$$

Now if we can show that 0^t lies in T_ℓ for all possible values of t for such a V_ℓ, then there is at least one 0^t in T_ℓ whose accepting computation can be computed from a string 0^{V_ℓ} in T_ℓ. Here $V_\ell = \tau^{\epsilon/b}(\log^c(2^{\ell+1} - 2))$. Recall that $T_\ell = \{0^{\tau^\epsilon(\log^c r_x)} \mid 2^\ell - 1 \leq r_x \leq 2^{\ell+1} - 2\}$. Then it's easy to verify that the upper and lower bounds on t gives the last and the first strings in T_ℓ, respectively. We use this observation to construct $\mathcal{B}_2$:

> On input $0^t \in T_\ell$, run $\mathcal{A}(0^{V_\ell})$ to get u_t. Then verify if u_t is an accepting computation of $N(0^t)$. If it is, then output u_t.

The above observation implies that there is such a 0^t in T_ℓ on which $\mathcal{B}_2$ outputs u_t. The algorithm runs in time polynomial in V_ℓ. □

3.3 Power of The Hypothesis

In this section, we establish some results that explain the power of Hypothesis W and also compare it to some of the previously studied hypotheses that are used to separate NP-completeness notions.

Even though Hypothesis W talks about the difficulty of computing accepting computations of NEEXP machines, our first result states that it can be related to the hardness of the complexity class NEEXP $\cap$ co-NEEXP.

Hypothesis 2. There exist $c > 1$ and $\delta < 1$ such that for every fully time-constructible $t(n) = 2^{2^{n^c}}$,

$$\text{NTIME}(t(n)) \cap \text{co-NTIME}(t(n)) \nsubseteq \text{ZPTIME}(2^{t(n)^\delta}).$$

Now we show that our hypothesis follows from this worst-case separation hypothesis.

Proposition 3.1. *Hypothesis 2 implies Hypothesis W.*

Proof. Suppose L is a language that satisfies the hypothesis. Let N_1 and N_2 be two machines that witness that L and $\overline{L}$ are in NTIME($t(n)$). Consider the following machine N: On input x, guess $b \in \{1, 2\}$ and run N_b. Clearly, N is an NTIME($t(n)$) machine and accepts Σ^*. Suppose there is a probabilistic machine M running in time $2^{t(n)^\delta}$, such that for all but finitely many x, $M(x)$ outputs an accepting computation of $N(x)$ with probability at least $1/4$. Note that by looking at an accepting computation of $N(x)$, we can decide whether $x \in L$ or not. Consider a machine Z that on input x does the following:

> Run $M(x)$. If the output of $M(x)$ is an accepting computation of N_1, then Z accepts x. If the output of $M(x)$ is an accepting computation of N_2, then Z rejects x. Otherwise, Z outputs $\perp$.

Clearly, Z is a zero-error machine whose running time is $O(2^{t(n)^\delta})$ and for every x, Z correctly decides L with probability at least $1/4$. By running this machine $O(1)$ times, we obtain that L is in ZPTIME($2^{t(n)^\delta}$). This contradicts Hypothesis 2. □

Pavan and Selman [PS02] showed that the NP-completeness notions differ under the following hypothesis.

Hypothesis 3. (NP-machine Hypothesis) There exist an NP machine N accepting 0^* and $\beta > 0$ for every 2^{n^β}-time bounded deterministic algorithm M, $M(0^n)$ does not output an accepting computation of $N(0^n)$ *for all but finitely many* n.

Note that the hypothesis requires that every machine that attempts to compute accepting computations of N must fail on *all but finitely many inputs*. This type of hardness hypothesis is called "almost every hardness hypothesis". In contrast, Hypothesis W requires that every machine that attempts to compute accepting computations of the NEEXP machine must fail on only *infinitely many strings*.

Ideally, we would like to show that NP-machine hypothesis implies Hypothesis W. However, NP-machine hypothesis concerns with hardness against deterministic algorithms whereas Hypothesis W concerns with hardness against probabilistic algorithms. If we assume well-accepted derandomization hypotheses, we can show Hypothesis W is weaker than the NP-machine hypothesis.

Proposition 3.2. *Suppose that* ZPP = P. *If* NP*-machine hypothesis holds, then Hypothesis W holds.*

Proof. Consider the NP machine N from the NP-machine hypothesis that runs in time n^d for some $d > 0$. Define a padding function $pad : \Sigma^* \to \mathbb{N}$ by

$$pad(x) = \tau((\log r_x - 1)^2),$$

where $c > 1$, and r_x is the lexicographic rank of x in Σ^*. Now define the NEEXP machine N_1 as follows: On input x, compute $m = pad(x)$ and run $N(0^m)$. Clearly, N accepts Σ^* and runs in time $\tau^d(n^2)$, for $x \in \Sigma^n$ (so $r_x \leq 2^{n+1} - 2$). Assume $t(n) = \tau^d(n^2)$.

Suppose there is a probabilistic machine Z_1 running in time $2^{t(n)^\delta}$, such that for all but finitely many x, $Z_1(x)$ outputs an accepting computation of $N_1(x)$ with probability atleast $1/4$. Consider a machine Z that on input 0^m does the following: Compute $x = pad^{-1}(m)$ and run $Z_1(x)$. Clearly, Z is a zero-error machine that runs in time

$$O(2^{t(n)^\delta}) = \tau(\delta d 2^{n^2}) < \tau(\beta 2^{n^2}) = 2^{m^\beta}$$

for some appropriate $\beta > \delta d$. Note that for every n, this time bound *holds* for the last string in Σ^n and *does not* hold for the first string in Σ^n. Clearly, there are infinitely many 0^m where Z correctly computes the accepting computation of $N(0^m)$ in time 2^{m^β}. If we assume that full derandomization is possible, then we can replace Z by a deterministic machine M that runs in time $poly((2^{m^\beta}))$ which is 2^{m^ϵ} for an appropriate $\epsilon > \beta$. Hence contradiction. □

Lutz and Mayordomo [LM96] achieved the separation of NP-completeness notions under the *Measure hypothesis*. Hitchcock and Pavan [HP08] showed that *Measure hypothesis* implies the NP-machine hypothesis. Thus we have the following.

Proposition 3.3. *Suppose that* ZPP = P. *Measure hypothesis implies Hypothesis W.*

Pavan and Selman [PS04] showed that if NP-contains 2^{n^ϵ}-*bi-immune* sets, then completeness in NP differ. Informally, the hypothesis means the following: There is a language L in NP such that every 2^{n^ϵ}-time bounded algorithm that attempts to decide L must fail on *all but finitely many strings*. Thus this hypothesis concerns with almost-everywhere hardness, whereas Hypothesis W concerns with worst-case hardness. We are not able to show that the bi-immunity hypothesis implies Hypothesis W (even under the assumption ZPP = P). However, we note that if NP ∩ co-NP has bi-immune sets, then Hypothesis W follows. Pavan and Selman [PS02] showed that if NP ∩ co-NP has a DTIME(2^{n^ϵ})-*bi-immune* set, then NP-machine hypothesis follows.

Proposition 3.4. *Suppose that* ZPP = P. *If* NP ∩ co-NP *has a* DTIME(2^{n^ϵ})-*bi-immune set, then Hypothesis W holds.*

3.4 Conclusions

This result, for the first time, shows that Turing completeness for NP can be separated from many-one completeness under a worst-case hardness hypothesis. Our hypothesis concerns with hardness of nondeterministic, double exponential time. An obvious question is to further weaken the hypothesis. Can we achieve the separation under the assumption that there exists a language in NE that cannot be solved in deterministic or probabilistic time $O(2^{\delta 2^n})$?

CHAPTER 4. THE ESY CONJECTURE ABOUT PROMISE PROBLEMS

Even, Selman, and Yacobi [SY82, ESY84] conjectured that there do not exist certain promise problems all of whose solutions are NP-hard via Turing reductions. When phrased in the language of disjoint pairs, this statement means that there do not exist disjoint NP-pairs all of whose separators are NP-hard via Turing reductions. This conjecture has fascinating (and largely believable) consequences, including that NP differs from co-NP and NP is not equal to UP. Even though this conjecture is 30 years old, we do not have any concrete evidence neither in support of the conjecture nor against it. In this chapter, we report some exciting progress on this conjecture. We consider variants of the conjecture and show that under some reasonable hypotheses, these variants of the conjecture hold.

Consequences of the ESY Conjecture

The ESY conjecture has numerous interesting consequences in various areas of complexity theory. For example, it has some interesting implication regarding the hardness of public-key cryptosystems. Even, Selman, and Yacobi [ESY84] observed that the problem of cracking a public-key cryptosystem may not formalize as a straightforward decision problem, and it is more natural to formulate it as a promise problem. They associated a promise problem (Π_y, Π_n) to a model of public-key cryptosystems such that both Π_y and Π_n are in NP. Then they showed that a public-key cryptosystem that fits the model cannot be deemed secure if the underlying promise problem admits at least one efficient (polynomial-time computable) solution. On the other hand, if every solution is NP-hard via Turing reduction then the system is NP-hard to crack. Thus the ESY conjecture implies that public-key cryptosystems that fit the model are not NP-hard to crack.

The ESY conjecture is also related to the study of propositional proof systems [Raz94, Pud01]. Razborov observed that every propositional proof system f can be identified with a canonical disjoint NP-pair $(\mathrm{SAT}^*, \mathrm{REF}_f)$ where REF_f is the set of all Boolean formulas that have short proofs of unsatisfiability with respect to f. Conversely, Glaßer, Selman, and Zhang [GSZ07] showed that for every disjoint NP-pair (A, B) there is a proof system f such that (A, B) is many-one equivalent to $(\mathrm{SAT}^*, \mathrm{REF}_f)$. Because of this equivalence between propositional proof systems and disjoint NP-pairs, several interesting questions regarding propositional proof systems are related to the structure of disjoint NP-pairs. One of the open questions on propositional proof systems is whether optimal proof systems exist and the belief is that they do not exist. This question is related to the ESY conjecture. It is known that if optimal proof systems do not exist, then a variant of the ESY conjecture holds [GSSZ04a].

In addition to connections with public-key cryptosystems and propositional proof systems, the ESY conjecture has several believable consequences in complexity theory. It is known that this conjecture implies the following results in complexity theory: NP differs from co-NP, NP differs from UP, and satisfying assignments of Boolean formulas cannot be computed by single-valued NP machines (NPSV) [ESY84, GS88]. However, we do not know yet if any of these consequences hold unconditionally. This is one of the reasons why the ESY conjecture is difficult to prove.

Our Contribution

Given its relation to public key cryptosystems, propositional proof systems, and complexity theory, it is important to understand the power of the ESY conjecture. Is there a reasonable hypothesis that implies the conjecture? To date, we do not know any reasonable hypothesis that implies the ESY conjecture. However, it is interesting to note that the analogue of the ESY conjecture to the computational world, that is for the computationally enumerable sets, is a known theorem [Sch60]. It seems difficult to formulate reasonable hypotheses that imply the ESY conjecture due to its wide range of consequences. Any hypothesis that implies the ESY conjecture immediately implies the aforementioned results in complexity theory. On the other hand, none of the standard hypotheses used in complexity theory, such as PH is infinite,

E has high circuit complexity, the measure of NP is not zero, etc. are known to imply all of the above mentioned consequences. This seems to be the root difficulty in proving the ESY conjecture.

The original ESY conjecture states that every disjoint NP-pair has a solution that is not NP-hard via *Turing reductions*. We can obtain variants of the conjecture by replacing Turing reductions with less restrictive adaptive and non-adaptive reductions. Given a reduction type r, the ESY-r conjecture states that every disjoint NP-pair has a solution that is not NP-hard via r-reductions. We know already that if we take r to be many-one reductions, then the ESY-$\leq^{\mathrm{P}}_m$ conjecture is equivalent to NP $\neq$ co-NP [GSSZ04a]. What if we take r to be truth-table reductions or bounded-truth-table reductions? In this chapter, we address the above question. More formally, we prove the following main theorem in this chapter: If there exist one-one, one-way functions that are hard to invert via circuits with an NP $\cap$ co-NP-oracle, then the ESY conjecture for $\leq^{\mathrm{P}}_{btt}$ reductions holds.

Organization of this Chapter

The rest of the chapter is organized as follows. First, we provide the formal definition of the ESY conjecture and some of its properties in the next section. Rest of the chapter is devoted to the proof of the main theorem. The starting point is the following theorem by Hughes, Pavan, Russell, and Selman [HPRS12]: If NP $\neq$ co-NP, then every disjoint NP-pair has a solution that is not NP-hard via length-increasing bounded-truth-table reductions (i.e., the ESY conjecture for *btt* length-increasing reductions holds). For completeness, we reproduce the proof of this theorem in Section 4.2. The main idea behind the proof of the above theorem is that by using a stronger hypothesis than that of Hughes *et al.*, we remove the length-increasing restriction from their hypothesis. Our hypothesis concerns with the existence of cryptographic one-way functions. This is the focus of Section 4.3 and Section 4.4. Finally, Section 4.5 concludes this chapter with references to future direction.

4.1 ESY Conjecture

The ESY conjecture concerns about the disjoint NP pairs. A *disjoint* NP*-pair* is a pair (A, B) of non-empty, disjoint sets A and B such that both A and B belong to NP. Let DisjNP denote the collection of all disjoint NP-pairs. We say that a set S is a *separator* or *solution* for the disjoint NP-pair (A, B) if $A \subseteq S$ and $B \subseteq \overline{S}$. Intuitively, the solution set S separates A from B.

We now state the original conjecture of Even, Selman, and Yacobi [ESY84].

ESY Conjecture. For every disjoint NP-pair, there is a separator that is not Turing hard for the complexity class NP.

Although the original conjecture talks about Turing hardness, we can generalize it to arbitrary polynomial-time reductions. Let r be a polynomial-time reduction. Then the ESY conjecture for this reduction can be stated as follows.

ESY-r Conjecture. For every disjoint NP-pair, there is a separator that is not r-hard for NP.

In this chapter, we take r to be bounded-truth-table reduction and prove evidence for the ESY-$\leq^{\mathrm{P}}_{btt}$ conjecture.

Although the ESY conjecture stipulates a condition about arbitrary pairs of sets in NP, the following observation from [HPRS12] tells us that we can always take one of the sets to be SAT.

Observation 4.1. *The* ESY-r *conjecture is equivalent to the following statement: For every set* B *in* NP *that is disjoint from* SAT*, there is a separator of* (B, SAT) *that is not* r*-hard for* NP*.*

Further, given any two types of reductions, if one reduction is stronger than the other, then there is a simple relation between the ESY conjecture for those two reductions.

Observation 4.2. *Let* r *and* r' *be two polynomial-time reductions such that* r'*-hardness for* NP *implies* r*-hardness for* NP*. If the* ESY-r *conjecture holds then the* ESY-r' *conjecture holds.*

This observation implies that if the ESY conjecture for Turing reduction holds, then so does the ESY conjecture for the truth-table reduction, bounded-truth-table reduction, or many-one reduction.

The next observation from [HPRS12] is crucial. It says that the ESY-$\leq^{\mathrm{P}}_{tt}$ conjecture has the same set of consequences as the original ESY conjecture. We reproduce below the proof for completeness.

Observation 4.3. *The* ESY-$\leq^{\mathrm{P}}_{tt}$ *conjecture implies the following:*

(a) NP $\neq$ UP*,*

(b) NP $\neq$ co-NP*, and*

(c) satisfying assignments of Boolean formulas cannot be computed by single-valued NP *machines.*

Proof. Assume that NP = UP. Thus SAT is in UP and so let R be a relation that witnesses that SAT is in UP. Consider the following two disjoint languages in NP:

$$A = \{\langle x, i\rangle \mid \exists w\ R(x, w) = 1 \text{ and the } i\text{th bit of } w \text{ is } 1\}$$

and

$$B = \{\langle x, i\rangle \mid \exists w\ R(x, w) = 1 \text{ and the } i\text{th bit of } w \text{ is } 0\}.$$

Let S be any separator for (A, B). Then below is a truth-table reduction from SAT to S.

> On input x, produce queries $\langle x, 1\rangle, \cdots, \langle x, m\rangle$ (where m is the number of Boolean variables in the propositional formula x). If $\langle x, i\rangle \in S$, then set $a_i = 1$, else set $a_i = 0$. Accept x if and only if $R(x, a_1 a_2 \cdots a_m) = 1$.

Therefore SAT is $\leq^{\mathrm{P}}_{tt}$-reducible to every separator of (A, B), so the ESY-$\leq^{\mathrm{P}}_{tt}$ conjecture does not hold.

A similar proof shows that if the ESY-$\leq^{\mathrm{P}}_{tt}$ conjecture holds, then satisfying assignments of Boolean values cannot be computed by single-valued NP-machines.

Lastly, assume that NP = co-NP. Then it follows that the ESY-$\leq^{\mathrm{P}}_{m}$ conjecture does not hold [GSSZ04b], so the ESY-$\leq^{\mathrm{P}}_{tt}$ conjecture also does not hold, by Observation 4.2. □

The observation suggests that providing evidence for even ESY-$\leq^{\mathrm{P}}_{tt}$ conjecture could be as difficult as providing evidence for the original conjecture. Here we provide the evidence for the ESY-$\leq^{\mathrm{P}}_{btt}$ conjecture.

4.2 ESY Conjecture for Length-increasing Bounded-truth-table Reductions

In this section we prove that if NP does not equal co-NP, then the ESY conjecture holds for length-increasing bounded-truth-table reductions. In fact, we will show that the conjecture holds even for reductions that use nondeterminism. Before we present the proof, we describe the ideas and intuition behind our proofs.

Proof Sketch

Let (B, SAT) be a disjoint NP-pair. Our goal is to exhibit a separator S that is not NP-hard. One trivial way to achieve this is by making S to be an easy set—a set in P. However, this approach is not feasible because if NP differs from UP or P does not equal NP $\cap$ co-NP, then (B, SAT) does not have separators in P (for some $B \in$ NP) [GS88]. Thus we look for a separator that is not in P. Our first observation is that there exist "computationally difficult" sets that are not NP-hard, thus we can achieve our goal by taking S to be a difficult set.

It is known that if H is an unpredictable set, then H does not reduce to $H \cup B$ [LM96, ASB00, PS04]. This suggests that we can take $H \cup B$ as our separator and claim that it is not NP-hard. However, we run into at least two major problems. The set $H \cup B$ may not be disjoint from SAT and thus cannot be a separator. In fact, one can show that an unpredictable set H must have an infinite intersection with SAT. We get around this problem by taking H as an unpredictable set within $\overline{\mathrm{SAT}}$. This ensures that S is a separator.

The second and the more serious problem is that showing H is not reducible to $H \cup B$ does not imply that $H \cup B$ is not NP-hard, as the set H may not be in NP. Instead of working with H, we will argue that SAT does not reduce to S. This argument makes a critical use of nondeterminism. We will show that if SAT does reduce to S, then either we can get a predictor for H, or by making use of nondeterminism, we can reduce the number of queries.

Our first observation is that any reduction from SAT to S must infinitely often produce *relevant queries*—these are queries whose answers, given answers to all other queries, uniquely determine the output of the reduction. We then show that these relevant queries must lie outside the set $B \cup \mathrm{SAT}$. If not, we can reduce the number of queries by making use of *strong nondeterminism*. Next we argue that if a query q is relevant, then knowing answers to all other queries help us determine the membership of $q \in S$, and if q lies outside of $B \cup \mathrm{SAT}$, then this contradicts the unpredictability of the set H.

Formal Proof

Now we formalize the above intuition.

Theorem 4.1. *If* NP $\neq$ co-NP, *then the* ESY-$\leq^{\mathrm{SNP}}_{btt,li}$ *conjecture is true.*

Proof. Suppose NP $\neq$ co-NP, but the ESY-$\leq^{\mathrm{SNP}}_{btt,li}$ conjecture is false. Let (B, SAT) be a disjoint NP-pair. By Observation 4.1, every separator of (B, SAT) is $\leq^{\mathrm{SNP}}_{btt,li}$-hard for NP. Let Q_1 and Q_2 be two polynomial-time computable relations for SAT and B respectively. Assume that the length of witnesses (for positive instances in SAT and in B) is bounded by n^r, $r > 0$. By Theorem 2.1 there is a set R that is $\mathrm{SNTIME}(2^{\log^{2r} n})$-unpredictable within $\overline{\mathrm{SAT}}$. Consider the separator $S = R \cup B$. Suppose that S is $\leq^{\mathrm{SNP}}_{ktt,li}$-hard for NP for some $k \geq 0$. We will achieve a contradiction to our hypothesis NP $\neq$ co-NP and to the fact that R is $\mathrm{SNTIME}(2^{\log^{2r} n})$-unpredictable within $\overline{\mathrm{SAT}}$. This will give us that S is not $\leq^{\mathrm{SNP}}_{ktt,li}$-hard for any $k \geq 0$ and therefore not $\leq^{\mathrm{SNP}}_{btt,li}$-hard for NP.

We prove this by induction. The base case is when the number of queries is zero. This means that there is an SNP computable function t such that $t(x) = \mathrm{SAT}(x)$. This implies that NP = co-NP, a contradiction.

As the inductive hypothesis, assume that S is not $\leq^{\mathrm{SNP}}_{(\ell-1)tt,li}$-hard. Now assume that SAT $\leq^{\mathrm{SNP}}_{\ell tt,li}$-reduces to S via $\langle f, t\rangle$, for contradiction. Given x, let $f(x) = \langle q_1, \cdots, q_\ell\rangle$. We assume that q_ℓ is the largest query and denote it with b_x. We say that a query q_i is *relevant* if the following holds:

$$t(x, S(q_1), \cdots, S(q_i), \cdots, S(q_\ell)) \neq t(x, S(q_1), \cdots, \overline{S(q_i)}, \cdots, S(q_\ell)).$$

In other words, if q_i is relevant then knowing answers to all the other queries still does not help us determine SAT(x). Then the following observation is easy to see.

Observation 4.4. *There exist infinitely many x such that b_x is relevant.*

Proof. Suppose not. Then for all but a finite number of x we can remove b_x from the list of queries, giving an $\leq^{\mathrm{SNP}}_{(\ell-1)tt,li}$-reduction from SAT to S, and this contradicts the induction hypothesis. □

Now let's define a set T for the largest query b_x such that

$$T = \{x \mid b_x \text{ is relevant}\}.$$

Then we prove the following result on the size of T.

Lemma 4.1. *There exist infinitely many $x \in T$ such that $b_x \notin B \cup \mathrm{SAT}$.*

Proof. Suppose not. For all but finitely many $x \in T$, the query b_x is relevant and belongs to $B \cup \mathrm{SAT}$. Now consider the following reduction $\langle f', t' \rangle$ from SAT to S: on input x, f' will first compute $f(x) = \langle q_1, q_2, \cdots, q_{\ell-1}, b_x \rangle$ and outputs the queries $\langle q_1, \cdots, q_{\ell-1} \rangle$. We now describe t':

Procedure t'

1: Let $b_1 = S(q_1), \cdots, b_{\ell-1} = S(q_{\ell-1})$.

2: Determine whether each b_x is relevant or not by comparing $t(x, b_1, \cdots, b_{\ell-1}, 0)$ with $t(x, b_1, \cdots, b_{\ell-1}, 1)$. If b_x is not relevant, then output $t(x, b_1, \cdots, b_{\ell-1}, 0)$.

3: Guess a witness $w \in \Sigma^{n^r}$. If $Q_1(b_x, w)$ holds, then output $t(x, b_1, \cdots, b_{\ell-1}, 0)$.

4: If $Q_1(b_x, w)$ does not hold, then guess a witness $u \in \Sigma^{n^r}$. If $Q_2(b_x, u)$ holds then output $t(x, b_1, \cdots, b_{\ell-1}, 1)$, else output $\perp$.

We claim that the above is an $\leq^{\mathrm{SNP}}_{(\ell-1)tt,li}$-reduction from SAT to S. Clearly, f' produces only $\ell - 1$ queries. If b_x is not relevant, then the reduction is correct. Suppose that b_x is relevant. By our assumption $b_x \in B \cup \mathrm{SAT}$. If $b_x \in \mathrm{SAT}$, then $b_x \notin S$. Thus $t(x, b_1, \cdots, b_{k-1}, 0) = \mathrm{SAT}(x)$.

If $b_x \in B$, then $b_x \in S$. Thus $t(x, b_1, \cdots, b_{k-1}, 1) = \mathrm{SAT}(x)$. Thus the reduction is always correct.

It remains to show that this is an SNP-reduction. Clearly all queries are produced by a deterministic polynomial-time process. Step 2 computes the function t. However t is SNP computable. So this step can be done via an SNP-machine. Suppose $b_x \in \mathrm{SAT}$. Then there is a $w \in \Sigma^{n^r}$ such that $Q_1(b_x, w)$ holds, and thus this path outputs the correct answer. Since SAT is disjoint from B, for every $u \in \Sigma^{n^r}$, $Q_2(b_x, u)$ does not hold. Thus no path outputs the wrong answer. A similar argument shows that when $b_x \in B$, at least one path outputs the correct answer and no path outputs the wrong answer.

Thus SAT $\leq^{\mathrm{SNP}}_{(\ell-1)tt,li}$-reduces to S. This contradicts our induction hypothesis. This completes the proof of the lemma. □

Now, we return to the proof of the theorem. Lemma 4.1 has the following corollary.

Corollary 4.1. *There exist infinitely many $y \notin B \cup \mathrm{SAT}$ with the following property: There exists an x, $|x| < |y|$ such that $y = b_x$ and y is relevant.*

This enables us to build the following predictor for R. Let M be a strong nondeterministic algorithm that decides R.

Procedure M

1: Input $\langle y, R|y\rangle$.

2: If $y \in B \cup \mathrm{SAT}$, then run $M(y)$ and output the result.

3: Search for an x such that $|x| < |y|$ and $b_x = y$. If no such x is found run $M(y)$ and output the result.

4: Let $f(x) = \langle q_1, \cdots, q_{\ell-1}, y\rangle$. Compute $b_i = S(q_i)$, $1 \leq i \leq \ell - 1$, by the following:

   (a) Decide the membership of $q_i \in B$ by running a brute force algorithm for B.

   (b) Decide the membership of $q_i \in R$ by looking at $R|y$.

5: Check if y is relevant or not by comparing $t(x, b_1, \cdots, b_{\ell-1}, 0)$ and $t(x, b_1, \cdots, b_{\ell-1}, 1)$. If y is not relevant, then run $M(y)$ and output the result.

6: Now we know that y is relevant. Compute $\mathrm{SAT}(x)$. Find the unique bit b such that $\mathrm{SAT}(x) = t(x, b_1, \cdots, b_{\ell-1}, b)$.

7: Accept if and only if b equals 1.

Then we make the following claim.

Claim 4.1. *The above predictor correctly predicts R and for infinitely many strings from $\overline{\mathrm{SAT}}$ runs in time $2^{\log^{2r} n}$.*

Proof. Let I be the set of all y for which the conditions of Corollary 4.1 holds. The above predictor runs $M(y)$ on any y that is not in I and thus is correct on all such y. Let $y \in I$. We know that $\mathrm{SAT}(x) = t(x, b_1, \cdots, b_{\ell-1}, S(y))$. Since y is relevant $\mathrm{SAT}(x) \neq t(x, b_1, \cdots, b_{\ell-1}, \overline{S(y)})$. Thus $b = S(y)$. Since $y \notin B \cup \mathrm{SAT}$, $y \in S$ if and only if $y \in R$. Thus the above predictor correctly decides every y in I.

Now we will show that for every $y \in I$, the above predictor halts in quasi-polynomial time. Let $|y| = m$, note that the length of x found in step 3 is at most m. Checking for membership of y in $B \cup \mathrm{SAT}$ takes $O(2^{m^r})$ time. Since $y = b_x$ is the largest query produced, $|q_i| \leq m$, $1 \leq i \leq \ell - 1$. Since B can be decided in time 2^{n^r}, Step a takes $O(2^{m^r})$ time. Since $y > q_i$, $1 \leq i \leq \ell - 1$, Step b takes polynomial time. Computing $\mathrm{SAT}(x)$ takes $O(2^m)$ time. The

predictor computes the function t. However, t is SNP computable. Thus the total time taken is $O(2^{m^{r+1}})$. Note that the run time of the predictor is measured in terms of length of $\langle y, R|y\rangle$ which is at least 2^m. Thus for every $y \in I$, the predictor runs in time $2^{\log^{2r} n}$ time. Since I is an infinite set and by definition it is a subset of $\overline{\text{SAT}}$, the claim follows. □

We have shown that S is not $\leq^{\text{SNP}}_{\ell tt,li}$-hard for NP. This completes the induction step. Thus S is not $\leq^{\text{SNP}}_{btt,li}$-hard for NP. This completes the proof of the Theorem. □

The following corollary of the above theorem is straightforward.

Theorem 4.2. NP $\neq$ co-NP *if and only if the* ESY-$\leq^{\text{P}}_{btt,li}$ *conjecture holds.*

Proof. Suppose NP $\neq$ co-NP. Since every length-increasing bounded-truth-table reduction is trivially an $\leq^{\text{SNP}}_{btt,li}$-reduction, by Theorem 4.1 our result holds.

Suppose NP = co-NP. Then $(\text{CNFSAT}, \overline{\text{CNFSAT}}) \in \text{DisjNP}$. The only separator is CNFSAT. We know that CNFSAT is $\leq^{\text{P}}_{m\text{-}li}$-complete for NP. This can be shown via the Cook-Levin reduction, which given input x outputs a formula ϕ of length $\Theta(p_i(|x|) \log p_i(|x|))$, where p_i is the runtime of N_i from the standard enumeration of NP machines. Therefore the ESY-$\leq^{\text{P}}_{m\text{-}li}$ conjecture doesn't hold, so the ESY-$\leq^{\text{P}}_{btt,li}$ conjecture also doesn't hold by Observation 4.2. □

4.3 Proof of Main Theorem

In this section, our goal is to prove the following main theorem of this chapter:

Theorem 4.3. *Suppose that there exist an* $\epsilon > 0$ *and a one-one, one-way function that is* 2^{n^ϵ}*-secure against* NP $\cap$ co-NP *oracle circuits. Then the* ESY-$\leq^{\text{P}}_{btt}$ *conjecture holds.*

First, we extend the notion of SNP-reductions from Chapter 2 to strong, nondeterministic, quasi-polynomial-time reductions. We call a function *polynomially-bounded* if the length of the output of the function is bounded by a polynomial in input length.

Definition 4.1. Let A and B be two languages. We say that A is *strong, nondeterministic, quasi-polynomial k-truth table reducible* to B (denoted $A \leq^{\text{SNQP}}_{ktt} B$) if there is a polynomially-

bounded, quasi-polynomial-time computable function f and a strong nondeterministic quasi-polynomial-time computable function t such that for every x,

$$f(x) = \langle q_1, \cdots, q_k \rangle \text{ and } t(x, B(q_1), \cdots, B(q_k)) = A(x).$$

We call a $\leq^{\mathrm{P}}_{ktt}$-reduction *weakly-length-increasing*, if for every input x, the reduction outputs at least one query whose length is larger than the length of x. Suppose every language that is $\leq^{\mathrm{SNP}}_{ktt}$-hard for NP is hard via weakly-length-increasing $\leq^{\mathrm{SNP}}_{ktt}$-reductions. We first observe that under this assumption, the ESY-$\leq^{\mathrm{P}}_{btt}$ conjecture holds if NP differs from co-NP.

Observation 4.5. *Suppose that every set A that is $\leq^{\mathrm{SNP}}_{ktt}$-hard for* NP *is hard for* NP *via weakly-length-increasing $\leq^{\mathrm{SNP}}_{ktt}$-reductions. Then* co-NP *is a not a subset of* NP *if and only if the* ESY-$\leq^{\mathrm{P}}_{btt}$ *conjecture holds.*

Proof. Consider the proof of Theorem 4.1. The only places where we require the length-increasing part of the $\leq^{\mathrm{SNP}}_{\ell tt,li}$-reduction are in the proof of Lemma 4.1 and in the runtime analysis of Claim 4.1. Suppose we have a $\leq^{\mathrm{SNP}}_{\ell tt}$-reduction from SAT to set S. Then by our assumption, since S is $\leq^{\mathrm{SNP}}_{\ell tt}$-hard for NP, we have a weakly-length-increasing $\leq^{\mathrm{SNP}}_{\ell tt}$-reduction $\langle f, t \rangle$ from SAT to S, such that f is a query generator that generates ℓ queries. Let f' be the query generator that produces all queries of f except the largest query. Thus, there is a t', as described in the proof of Lemma 4.1, such that $\langle f', t' \rangle$ is an $\leq^{\mathrm{SNP}}_{\ell-1tt}$-reduction from SAT to S. However, in this case $\langle f', t' \rangle$ is not necessarily a weakly-length-increasing reduction, but is still an $\leq^{\mathrm{SNP}}_{(\ell-1)tt}$-reduction. This gives us that S is $\leq^{\mathrm{SNP}}_{(\ell-1)tt}$-hard for NP. By our assumption, there must exist another $\leq^{\mathrm{SNP}}_{(\ell-1)tt}$-reduction $\langle f'', t'' \rangle$, where f'' produces at least one query whose size is larger than the input size. Then with these changes, the proof of Lemma 4.1 goes through, the statement of Corollary 4.1 still holds, and the runtime analysis of predictor M holds in Claim 4.1. □

Our next observation is the following. The proof of Observation 4.5 goes through if we replace strong nondeterministic *polynomial-time* reductions with strong nondeterministic *quasi-polynomial-time* reductions and strengthen the hypothesis to include co-NP is not a subset of QuasiNP.

Observation 4.6. *Suppose that every set A that is $\leq_{ktt}^{\mathrm{SNQP}}$-hard for* NP *is hard for* NP *via weakly-length-increasing $\leq_{ktt}^{\mathrm{SNQP}}$-reductions, and suppose that* co-NP *is not a subset of* QuasiNP*, then the* ESY-$\leq_{btt}^{\mathrm{P}}$ *conjecture holds.*

Next we will show that assuming the existence of a certain kind of one-way function implies both the hypotheses of Observation 4.6 hold.

Observation 4.7. *Suppose that there exist an $\epsilon > 0$ and a one-one, one-way function that is 2^{n^ϵ}-secure against* NP $\cap$ co-NP *oracle circuits, then* co-NP *is not a subset of* QuasiNP.

Proof. Every one-way function can be trivially inverted in polynomial-time with an NP-oracle. Suppose that co-NP is a subset of QuasiNP, then NP is a subset of both QuasiNP and co-QuasiNP. Thus every one-way function can be inverted in polynomial-time with a QuasiNP$\cap$ co-QuasiNP oracle. We now use a padding argument to show that every one-way function can be inverted by quasi-polynomial size circuits with an NP $\cap$ co-NP oracle. Let f be a one-way function and M be a polynomial-time algorithm that inverts f with access to a language O that is in QuasiNP $\cap$ co-QuasiNP. Let r be a constant such that both O and $\overline{O}$ can be decided by nondeterministic algorithms running in time $O(2^{\log^r n})$ time. Define

$$O' = \{\langle x, 0^m\rangle \mid |x| = n, m = 2^{\log^r n}, \text{ and } x \in O\}.$$

Clearly O' is in NP $\cap$ co-NP. Consider the following algorithm M' with O' as oracle: On any input y, simulate $M(y)$. When M makes a query q (of length m) to O, make a query $\langle q, 0^{2^{\log^r m}}\rangle$ to oracle O'. Note that the time taken to form this query is $O(2^{\log^r m})$, since m is polynomial in the input length and M makes at most polynomially many queries, thus the total running time of M' is $O(2^{\log^{r'} n})$ for some constant $r' > r$. We can convert this algorithm into a circuit of size $O(2^{\log^{r''} n})$ for some constant $r'' > r'$. This contradicts the hypothesis. □

Now we show that the existence of the above 2^{n^ϵ}-secure one-way functions implies the first hypothesis of Observation 4.6.

Theorem 4.4 (Key Theorem)**.** *Suppose that there exist an $\epsilon > 0$ and a one-one, one-way function that is 2^{n^ϵ}-secure against* NP$\cap$co-NP *oracle circuits. Then every $\leq_{ktt}^{\mathrm{SNQP}}$-hard language for* NP *is hard for* NP *via weakly-length-increasing $\leq_{ktt}^{\mathrm{SNQP}}$-reductions.*

The above theorem is the heart of this section. It can be easily seen that the main theorem (Theorem 4.3) of this chapter follows by Observations 4.6, 4.7 and Theorem 4.4.

The rest of this chapter is devoted to the proof of Theorem 4.4. Agrawal [Agr02] and Agrawal and Watanabe [AW09] showed that if one-one, one-way functions exist, then all NP-complete sets are complete via nonuniform, one-one, and length-increasing reductions. Our proof heavily relies on the ideas in those papers. For the sake of simplicity, we prove the theorem for polynomial-time reductions. It can be verified easily that the proof holds for quasi-polynomial-time reductions.

4.4 Proof of Key Theorem

First, we give the proof ideas of Theorem 4.4 for polynomial-time many-one ($\leq^{\mathrm{P}}_m$) reductions. The proof proceeds in three steps. Suppose f_o is a 2^{n^ϵ}-secure one-way function from the hypothesis and L is a $\leq^{\mathrm{P}}_m$-hard language for NP, via a $\leq^{\mathrm{P}}_m$ reduction g. Let A be a language in NP that reduces to L via g. Now informally, define a $\leq^{\mathrm{P}}_m$ reduction to be "sparse" on a set S if the number of strings that are mapped to any single string in L is sufficiently "small" (to be defined formally later). In general, the $\leq^{\mathrm{P}}_m$ reduction g need not be sparse. The first step in our proof is to show that there is a sparse $\leq^{\mathrm{P}}_m$-reduction from A to L, for every language A in NP. The notion of Goldreich-Levin *hardcore bit* [GL89] is critical in this proof. Next, we show that there is a randomized length-increasing $\leq^{\mathrm{P}}_m$-reduction h from A to L, for every language A in NP. To prove this, we pad every input string x with a randomly selected "sufficiently large" (to be defined later) string r and apply the above sparse $\leq^{\mathrm{P}}_m$ reduction on $\langle x, r\rangle$. The sparsity of the $\leq^{\mathrm{P}}_m$ reduction helps us to bound the number of random strings r for which the reduction $h(\langle x, r\rangle)$ is not length-increasing for every input x. Finally, we derandomize this reduction using an appropriate pseudorandom generator.

Now we give the proof details.

Proof of Theorem 4.4. Let $\{f_o\}_{n\geq 1} : \Sigma^n \to \Sigma^{\ell(n)}$ be a family of one-one, one-way functions from the hypothesis. Let L be an $\leq^{\mathrm{SNP}}_{ktt}$-hard language for NP. We will view an $\leq^{\mathrm{SNP}}_{ktt}$-reduction $\langle f, t\rangle$ as a function from Σ^* to $(\Sigma^*)^k \times T_k$, where T_k is the set of all truth-tables for k variables. Let

us denote this function by $F_{f,t}$.

Definition 4.2. A truth-table-reduction $\langle f, t\rangle$ is *α-sparse* on a set $S \subseteq \Sigma^n$ if for every x_0 in S,

$$\Big|\{x \in \Sigma^n \mid F_{f,t}(x) = F_{f,t}(x_0)\}\Big| \leq \frac{2^n}{2^{n^\alpha}}.$$

4.4.1 Sparse Reduction in NP

First, we will prove that from every language A in NP, there is a sparse $\leq^{\mathrm{SNP}}_{ktt}$-reduction to L. Define A^n to be the set of strings of length n from the set A.

Lemma 4.2. *There exists a $\gamma > 0$ such that for every language A in* NP*, there is an $\leq^{\mathrm{SNP}}_{ktt}$-reduction from A to L that is γ-sparse on A^n for every $n \geq 0$.*

Proof. For the sake of simplicity, we assume that the language A is defined only on even length strings. If that were not the case, then we can work with $A' = A \cdot A$. Using the one-way function f_o, we first define the hard-core function of Goldreich-Levin [GL89]. Given a $2n$ bit string xy such that $x, y \in \Sigma^n$, $f_{gl}(xy) = \langle f_o(x), y, x \oplus y\rangle$. Since f_o is one-one, f_{gl} is one-one. Then the following lemma can be obtained by relativizing the hard-core bit theorem [GL89, HILL99] (In particular, Proposition 4.5 from [HILL99]).

Lemma 4.3. *There exists a γ ($< \epsilon$) such that for every sufficiently large n, for every oracle O in* NP $\cap$ co-NP*, and for every O-oracle circuit D of size at most 2^{n^γ},*

$$\Big|\Pr_{x,y\in\Sigma^n}[D(f_o(x), y, x \oplus y) = 1] - \Pr_{x,y\in\Sigma^n, b\in\{0,1\}}[D(f_o(x), y, b) = 1]\Big| \leq \frac{1}{2^{(3n)^\gamma}}.$$

Let $B = \{f_{gl}(w) \mid w \in A\}$. Since f_{gl} is one-one and is length-increasing, we have that B is in NP. Since B is in NP, there is an $\leq^{\mathrm{SNP}}_{ktt}$-reduction $\langle g, t\rangle$ from B to L. Thus $\langle f, t\rangle$ is a reduction from A to L, where $f = g \circ f_{gl}$. Note that there is a language O in NP $\cap$ co-NP such that f can be computed in polynomial-time with oracle access to O.

Now we will establish that $\langle f, t\rangle$ is γ-sparse on A^{2n}. Suppose not, then there exists a string $w_0 \in A^{2n}$ such that the size of the following set S is bigger than $\frac{2^{2n}}{2^{(2n)^\gamma}}$,

$$S = \{w \in \Sigma^{2n} \mid F_{f,t}(w) = F_{f,t}(w_o)\}.$$

We make the following observation.

Observation 4.8. *Since f_{gl} is one-one, $|S| = |f_{gl}(S)|$ and $F_{g,t}(f_{gl}(S)) = \{F_{f,t}(w_0)\}$.*

Define an O-oracle circuit D that on an input string z of length $\ell(n) + n + 1$ behaves as follows: If $F_{g,t}(z) = F_{f,t}(w_0)$ then accept, otherwise reject. The next observation follows from the definition of S and Observation 4.8.

Observation 4.9. *The circuit D accepts a string $z \in \Sigma^{\ell(n)+n+1}$ if and only if $z \in f_{gl}(S)$.*

By Observations 4.8 and 4.9,

$$\Pr_{x,y\in\Sigma^n}[D(f_o(x), y, x \oplus y) = 1] = p_n \geq \frac{1}{2^{(2n)^\gamma}}.$$

On the other hand,

$$\begin{aligned}
& \Pr_{x,y\in\Sigma^n,b\in\{0,1\}}[D(f_o(x), y, b) = 1] \\
= & \Pr_{x,y\in\Sigma^n,b\in\{0,1\}}[b = x \oplus y] \times \Pr_{x,y\in\Sigma^n,b\in\{0,1\}}[D(f_o(x), y, b) = 1 \mid b = x \oplus y)] \\
& + \Pr_{x,y\in\Sigma^n,b\in\{0,1\}}[b = \overline{x \oplus y}] \times \Pr_{x,y\in\Sigma^n,b\in\{0,1\}}[D(f_o(x), y, b) = 1 \mid b = \overline{x \oplus y}] \\
= & \frac{1}{2}\Pr_{x,y\in\Sigma^n}[D(f_o(x), y, x \oplus y) = 1] + \frac{1}{2}\Pr_{x,y\in\Sigma^n}[D(f_o(x), y, \overline{x \oplus y}) = 1] \\
= & \frac{1}{2}p_n.
\end{aligned}$$

The last equality is as follows: For every x and y, the tuple $\langle f_o(x), y, \overline{x \oplus y}\rangle$ does not belong to $f_{gl}(\Sigma^*)$ and hence does not belong to $f_{gl}(S)$. Thus, by Observation 4.8, $F_{g,t}(\langle f_o(x), y, \overline{x \oplus y}\rangle) \neq F_{f,t}(w_0)$. So D does not accept $\langle f_o(x), y, \overline{x \oplus y}\rangle$. Then

$$\begin{aligned}
\left|\Pr_{x,y\in\Sigma^n}[D(f_o(x), y, x \oplus y) = 1] - \Pr_{x,y\in\Sigma^n,b\in\{0,1\}}[D(f_o(x), y, b) = 1]\right| &= \frac{1}{2}p_n \\
&\geq \frac{1}{2}\frac{1}{2^{(2n)^\gamma}}.
\end{aligned}$$

Finally, note that D is a polynomial size circuit with access to NP $\cap$ co-NP oracle O. This contradicts the hard-core bit Lemma 4.3. □

4.4.2 Randomized Reduction

We will now show that for any set A in NP there is a randomized, weakly-length-increasing $\leq^{\mathrm{SNP}}_{ktt}$-reduction from A to L. Later we will derandomize this reduction.

Claim 4.2. *Let A be any language in* NP. *There is an $\leq^{\mathrm{SNP}}_{ktt}$-reduction $\langle h, t\rangle$ such that*

1. *For every x and r, if $h(x,r) = \langle q_1, \cdots, q_k, t\rangle$, then $x \in A$ if and only if $t(L(q_1), \cdots, L(q_k)) = 1$.*

2. *For every $x \in A$ of length n,*

$$\Pr_{r \in \Sigma^m}[\textit{there is a } q_i \in Q_{x,r} \textit{ such that } |q_i| > n] \geq 3/4,$$

where $Q_{x,r}$ is the set of all queries produced by $h(x,r)$ and $m > (nk)^{\frac{1}{1-\gamma}}$.

Proof. By Lemma 4.2, there is an $\leq^{\mathrm{SNP}}_{ktt}$-reduction $\langle h, t\rangle$ from $A \times \Sigma^*$ to L that is γ-sparse on $S = A^n \times \Sigma^m$ (for every n and m). Clearly this reduction satisfies property 1. Let R be the set of all tuples $\langle q_1, \cdots, q_k, t\rangle$ where each q_i is of length at most n and t is a truth-table over k variables. The total number of such tuples is at most $2^{(n+1)k} \times 2^{2^k}$. For a string x of length n, let us count the number of strings from $\{x\} \times \Sigma^m$ that are mapped to the elements of R. Note that every tuple from $\{x\} \times \Sigma^*$ can be encoded as a string of length $2(n+m)$. Since h is γ-sparse on S the total number of such strings is at most

$$\frac{2^{2n+2m}}{2^{(2n+2m)^\gamma}} \times 2^{(n+1)k} \times 2^{2^k} < 2^{(2m+2m)^{1-\gamma}+2nk} < 2^{6m^{1-\gamma}},$$

which is bounded by $2^m/4$ for sufficiently large n, $m > (nk)^{\frac{1}{1-\gamma}}$, and $\gamma > 0$. The first inequality in the above equation is due to the fact that $n < m$ and $k + 2^k < nk$ for large n. Thus for every x, the cardinality of the set $\{r \in \Sigma^m \mid h(x,r) \notin R\}$ is at least $\frac{3}{4}2^m$. Hence the claim follows. □

4.4.3 Derandomization

We will now derandomize the above reduction. Note that our hypothesis implies the existence of a hard language in EXP whose NP ∩ co-NP-oracle circuit complexity is 2^{n^δ} (for some $\gamma > \delta > 0$). Then Theorem 2.3 implies that we can construct a pseudorandom generator $G : \Sigma^{\log^a m} \to \Sigma^m$ that is secure against NP ∩ co-NP-oracle circuits of size $O(m)$. Thus for every $x \in A$ of length n, we have the following:

$$\Pr_{r \in \Sigma^{\log^a m}}[\text{at least one query of } h(x, G(r)) \text{ is of length bigger than } n] \geq 1/2. \tag{4.1}$$

The desired reduction from A to L works as follows: We describe the query generator. Given an input x of length n, set $m > (nk)^{\frac{1}{1-\gamma}}$, cycle through all strings r of length $\log^a m$ and compute $h(x, G(r)) = \langle q_1, \cdots, q_\ell \rangle$. If at least one q_i is of length bigger than n, then output this tuple and stop. If for every r, every query of $h(x, G(r))$ is of length at most n, then by inequality (4.1), it must be the case that x is not in A. In this case the reduction simply outputs $\langle 0^{n+1}, \cdots, 0^{n+1}, F \rangle$, where F is the truth-table that always evaluates to false. Note that the running time of the query generator is deterministic quasi-polynomial. So this is a strong nondeterministic, quasi-polynomial-time, k-tt reduction.

This completes the proof of Theorem 4.4. □

4.5 Conclusions

In this chapter we provided the evidence that the ESY conjecture holds when we restrict the power of the reduction. Hughes, Pavan, Russell, and Selman showed that the ESY-conjecture for length-increasing, $\leq^{\mathrm{P}}_{btt}$-reductions is equivalent to NP $\neq$ co-NP [HPRS12]. They also showed that by using a stronger hypothesis, namely, if NP has SNTIME(n^2) unpredictable set, we can remove the length-increasing restriction. We provide further evidence in support of it, that is, our Theorem 4.3 removes the length-increasing restriction by using a different but stronger hypothesis that involves the existence of secure one-way functions.

An obvious question is whether we can weaken the hypotheses. The hypothesis of 4.3 requires the existence of one-one, one-way functions that are hard for circuits with NP $\cap$ co-NP oracles. The one-way functions studied in most of the literature (henceforth called "standard one-way functions") only require hardness against subexponential-size circuits (having no oracles). Can we show that if standard one-way functions exist, then the ESY-$\leq^{\mathrm{P}}_{btt}$ conjecture holds? We note that this is unlikely. Recall that the ESY-$\leq^{\mathrm{P}}_{btt}$ conjecture implies NP $\neq$ co-NP (equivalently, NP differs from NP$\cap$co-NP). Thus a positive answer to our question immediately shows that if standard-one way functions exist, then NP differs from NP $\cap$ co-NP. Intuitively, the existence of standard one-way functions imply that NP is hard against subexponential time/circuits, not hard against NP $\cap$ co-NP. We cannot hope to prove that the existence of standard one-way functions separates NP from co-NP.

Note that the existence of one-way functions that are hard against NP $\cap$ co-NP-oracle circuits, implies that NP is *hard on average* for NP$\cap$co-NP. This raises the following interesting question: If NP is average-case hard for NP $\cap$ co-NP, then does the ESY-$\leq^{\mathrm{P}}_{btt}$ conjecture hold? Another question is whether we can replace $\leq^{\mathrm{P}}_{btt}$-reductions with reductions that make $O(\log n)$ (or n^{ϵ}) nonadaptive queries. We believe that the techniques used in this chapter can be extended to work for $O(\log n)$ queries.

CHAPTER 5. MULTIPASS PROBABILISTIC SPACE-BOUNDED MACHINES

In this chapter, we investigate probabilistic space-bounded Turing machines that are allowed to access their random bits multiple times. There are several ways to access the random tape multiple times. We can restrict the number of access to any individual bit on the random tape; such machines are called *multi-read* machines. Impagliazzo, Nisan, and Wigderson [INW94] considered such machines and showed that their pseudorandom generator can fool any constant-read probabilistic space-bounded machine. The second type is the *two-way access* to the random tape where the tape head can move left or right based on the outcome of an unbiased coin toss. This is the most unrestricted access to the random tape. We denote by 2-*way*BPL the class of languages that are accepted by bounded-error probabilistic logspace machines with *two-way access* to the random tape. While we know that BPL is in deterministic polynomial time, it is not known whether 2-*way*BPL is even in deterministic sub-exponential time (it's in BPP). While one-way access machines can be characterized using certain graph reachability problem, we do not have such a nice combinatorial characterization for two-way access machines.

Our focus in this chapter is a third type of multi-access which lies between one-way and unrestricted two-way access to the random tape—we call such machines *multipass* machines, where the machines make multiple passes over the random tape and in each pass the machines can access the random tape only one-way.

Previous Work

Our understanding of multiple-access models is limited. While investigating deterministic simulations of probabilistic space-bounded machines, Borodin, Cook, and Pippenger [BCP83] raised the question whether two-way probabilistic $s(n)$-space-bounded machines can be sim-

ulated deterministically in $O(s^2(n))$ space. Karpinsky and Verbeek [KV85] showed that the answer is negative in general. They showed that two-way logspace probabilistic machines that are allowed to run for $2^{n^{O(1)}}$ time can simulate PSPACE with zero error probability. In another relevant result, Nisan [Nis93] showed that BPL can be simulated by zero-error, probabilistic, space-bounded machines that have a two-way access to the random tape. While investigating the power of two-way machines, Nisan showed that 2-*way*BPL is same as almost-logspace, where almost-logspace is the class of languages accepted by deterministic logspace machines relative to a random oracle.

It is also interesting to note that allowing two-way access to the random tape for a space-bounded machine makes the corresponding nonuniform classes closer to randomized circuit complexity classes. It is known that logspace uniform NC_1 is in deterministic logspace, where NC_1 is the class of languages accepted by polynomial-size, bounded fan-in $O(\log n)$-depth circuits. However, a randomized version of this inclusion is not known to hold. That is, we do not know whether (uniform) BPNC_1 is contained in BPL. However, it is a folklore that (uniform) BPNC_1 is in 2-*way*BPL (for example, see [Nis93]).

Probabilistic space-bounded machines that can make multiple passes over the random tape was first considered by David, Papakonstantinou, and Sidiropoulos [DPS11]. They showed that any pseudorandom generator that fools traditional $k(n)s(n)$-space bounded machines can also fool $k(n)$-pass $s(n)$-space bounded machines. As a corollary, they obtain that polylog-pass, randomized logspace is contained in deterministic polylog-space. David, Nguyen, Papakonstantinou, and Sidiropoulos [DNPS11] further considered probabilistic space-bounded machines that have access to a stack and a two-way/multipass access to the random tape and compared their power to the traditional deterministic and probabilistic classes.

Our Contributions

In this chapter, we prove results that indicate that it is fruitful to study multipass probabilistic space-bounded machines. As our main result in this chapter, we connect derandomization of multipass machines with the derandomization of time-bounded probabilistic complexity classes. Showing that $\mathrm{BPTIME}(n)$ is a subset of $\mathrm{DTIME}(2^{o(n)})$ is a significant open problem.

The best unconditional derandomization of BPTIME(n) till date is by Santhanam and van Melkebeek [SvM05] who showed that any bounded-error linear time probabilistic machine can be simulated in DTIME($2^{\epsilon n}$), where $\epsilon > 0$ is a constant that depends on the number of tapes and the alphabet size of the probabilistic machine. As the main result of this chapter, we show that derandomizing an $O(\log^{(k)} n)$-pass probabilistic space-bounded machine that uses only $O(\log n \log^{(k+3)} n)$ random bits to polynoimal time yields a non-trivial derandomization of BPTIME(n). Our result is actually stronger. The theorem holds for any non-constant number of passes. Notice that if we restrict the number of random bits from $O(\log n \log^{(k+3)}(n))$ to $O(\log n)$, then the corresponding set of languages is trivially in P. If we restrict the number of passes from $O(\log^{(k)} n)$ to $O(1)$, we can still show that the set of languages accepted is in P. Thus, the above theorem states that any extension of these simulations will lead to a non-trivial and unknown derandomization of BPTIME(n).

Our next set of results reveal the connection between multipass machines and the traditional probabilistic time- and space-bounded machines. First, we show that any $k(n)$-pass, $s(n)$-space, probabilistic machine can be simulated by traditional $k(n)s(n)$-space bounded probabilistic machines. Thus, in particular, a constant number of passes do not add power to the traditional one-way machines. Using the well-known derandomization results of Saks and Zhou [SZ99], we further show that the language from our hypothesis in the previous paragraph is contained in DTIME($n^{(\log \log n)^{1/2+\epsilon}}$), which is slightly higher than our desired polynomial-time. Next we ask the following question: Can we derandomize a logspace multipass machine to a deterministic logspace machine along the line of Adleman [Adl78]? Adleman showed that 2-*way*BPL can be simulated by a deterministic logspace machine with a polynomial amount of advice. Fortnow and Klivans [FK06] showed that if we restrict our attention to BPL, it can be simulated by a deterministic logspace machine with only a linear amount of advice. We improve their result. We show that even a probabilistic machine with polylog(n) number of passes on its random tape can be simulated by the same deterministic logspace machine with linear advice.

Next, we focus on the randomness-efficient deterministic amplification of the multipass probabilistic machines. Reducing the error probability of probabilistic machines is an important problem and has applications in derandomization. One way to achieve this is to run the

algorithm multiple times with fresh set of random strings and take the majority vote. However, this increases the number of random bits required for the computation. Is there a way to reduce the error, while keeping the number of random bits used by the machine at check? The method that achieves this is called the deterministic amplification. There are many deterministic amplification methods [AKS87, CW89, IZ89, Nis92] known for the time-bounded probabilistic complexity classes. The basic idea is to use an expander graph whose each node acts as a source of randomness and we can follow a random walk on the expander to pick up those nodes. However, all of these methods use more computational space than we can afford for the probabilistic space-bounded computations. Bar-Yossef, Goldreich, and Wigderson [BYGW99] provided a space-efficient encoding for the constant-degree expanders to obtain a deterministic amplification of the space-bounded machines. Specifically, they showed that any $s(n)$-space-bounded probabilistic machine that uses $R(n)$ random bits and has an error probability $\epsilon < 1/2$ can be amplified deterministically to a $O(t(n)s(n))$ space-bounded probabilistic algorithm that uses $R(n) + O(t(n))$ random bits and errs with probability $\epsilon^{\Omega(t(n))}$, for any function $t(n)$. We prove a similar theorem for multipass machines in this chapter.

Finally, we show a hierarchy theorem for the multipass machines using a result of Kinne and van Melkebeek [KvM09]. Note that the hierarchy theorems for the time- and space-bounded probabilistic machines are not known unconditionally and all the partial results require at least one advice bit [Bar02, FS04, FS06, FS07, FST05]. Using this result we show that linear time probabilistic class with a single bit of advice is a *strict subset* of $O(\log^{(3)} n)$-pass linear space probabilistic class with a single bit of advice. This result can be compared with the seminal result of Hopcroft, Paul, and Valiant that the DTIME($t(n)$) is strictly in DSPACE($t(n)$), for any function $t(n)$.

Organization of this Chapter

Rest of this chapter is organized as follows. We establish results that connect derandomization of multipass machines to derandomization of probabilistic time classes in the next section. In Section 5.2, we consider the problem of simulating multipass, probabilistic, space-bounded machines with the traditional one-pass machines. Next, we provide a space-efficient determinis-

tic non-uniform simulation of the multipass machines. In Section 5.4, we obtain a randomness-efficient deterministic amplification for the multipass machines. Section 5.5 provides a hierarchy theorem for the multipass machines. Finally, Section 5.6 concludes this chapter with the future direction of this line of work.

5.1 Derandomization of Probabilistic Time

As our main result of this section, we show that a time-efficient derandomization of probabilistic logspace machines that use very few random bits and make very few passes over their random tape, yields a non-trivial derandomization of probabilistic time. In particular, we show the following theorem.

Theorem 5.1. *If for some constant $k > 0$, $O(\log^{(k)} n)$-pass* BPL $\left[\log n \log^{(k+3)} n\right]$ *is in* P, *then* BPTIME$(n) \subseteq$ DTIME$(2^{o(n)})$.

Remark 5.1. We use the iterated logarithmic function for simplicity, but the above theorem can be proved with any "nice" slowly growing function $f(n) \in \omega(1)$.

We establish the theorem by first proving that every BPTIME(n) machine can be simulated by a bounded-error probabilistic space-bounded machine that makes $O(\log^{(k)} n)$ passes over the random tape and uses $o(n)$ space, on inputs of size n. There is a trade-off between the number of passes and space used by the simulating machine and this trade-off is essential in the proof. More formally, we prove the following theorem.

Theorem 5.2. *For every constant $k > 0$,*

$$\text{BPTIME}(n) \subseteq O(\log^{(k)} n)\text{-pass BPSPACE}\left[n/\log^{(k+3)} n, n\right].$$

Remark 5.2. The above theorem may be of independent interest. In a seminal result, Hopcroft, Paul, and Valiant [HPV77] showed that DTIME$(n) \subseteq$ DSPACE$(o(n))$[1]. The analogous inclusion relationship for probabilistic classes is not known. That is, we do not know *unconditionally* if BPTIME(n) is a subset of BPSPACE$(o(n))$ (see [LV03, KLV03] for conditional results in this direction). The above theorem can be viewed as a partial solution in this direction; if we allow

[1] In fact, they showed that DTIME$(t(n)) \subseteq$ DSPACE$(t(n)/\log t(n))$, for any time-constructible function $t(n)$.

the probabilistic space-bounded machine to have a slightly non-constant number of passes, then BPTIME(n) can be simulated in $o(n)$ probabilistic space.

We first give the proof of Theorem 5.1 assuming Theorem 5.2. The proof is trivial and uses a simple padding argument.

Proof of Theorem 5.1. Let $k > 0$ be a constant for which the hypothesis in the statement of Theorem 5.1 holds. Let L be a language in BPTIME(n). Then by Theorem 5.2, we know that the language L is in $O(\log^{(k-1)} n)$-*pass* BPSPACE $\left[n/\log^{(k+2)} n, n\right]$. Let

$$L' = \left\{ \langle x, 0^{2^{n/\log^{(k+2)} n} - n} \rangle \;\mid\; x \in L, |x| = n \right\}.$$

It is easy to see that L' is in $O(\log^{(k)} n)$-*pass* BPL $\left[\log n \log^{(k+3)} n\right]$. By our hypothesis, L' is in P. So $L' \in$ DTIME(n^ℓ) for some $\ell > 0$. From this it follows that for some $\ell > 0$, L is in DTIME $\left(2^{\frac{\ell n}{\log^{(k+2)} n}}\right)$ and thus in DTIME($2^{o(n)}$). □

Now we move on to proving Theorem 5.2. The proof relies on the classical result of Hopcroft, Paul, and Valiant [HPV77] who showed that every deterministic machine running in time $O(n)$ can be simulated by a deterministic machine that uses $O(n/\log n)$ space. If we adopt their proof to the case of probabilistic machines, we obtain that every bounded-error probabilistic machine running in time $O(n)$ can be simulated by a bounded-error, probabilistic machine that uses space $O(n/\log n)$; however, the simulating machine makes an exponential number of passes over the random tape. We observe that the number of passes can be greatly reduced at the expense of a little increase in space. This is essentially achieved by using a careful choice of parameters than those used in [HPV77].

To proceed with the proof, we need the notions of *block-respecting Turing machines* and *pebbling games*, introduced by [HPV77].

Definition 5.1. Let M be a multi-tape Turing machine running in time $t(n)$. Let $b(n)$ be a function such that $1 \leq b(n) \leq t(n)/2$. Divide the computation of M into $a(n)$ time segments so that each segment has $b(n) = t(n)/a(n)$ steps. Also divide each tape of M into $a(n)$ blocks so that each block has exactly $b(n)$ cells. We say that the machine M is $b(n)$-*block respecting* if

during each time segment every tape head of the machine M visits cells of exactly one block. I.e, a tape head can cross a block boundary only at time step $c \cdot b(n)$ for some integer $c > 0$.

Then it's easy to see the following. Refer to their paper and text by Balcázar, Diaz, and Gabarró [BDG90] for a proof.

Lemma 5.1. *Every ℓ-tape Turing machine running in time $t(n)$ can be simulated by a $(\ell+1)$ tape $b(n)$-block respecting Turing machine running in time $O(t(n))$ for any $b(n)$, $1 \leq b(n) \leq t(n)/2$.*

Definition 5.2 (Pebbling Game)**.** Let $G = (V, E)$ be a directed acyclic graph, w be a special vertex of G, and P be a set of pebbles. We say a vertex u is a *predecessor* of vertex v in G if there is an edge from u to v. Then the pebbling game is defined as the following optimization problem: Place a pebble on the special vertex w such that the number of pebbles used from P is minimized, subject to the following constraints:

1. Place a pebble on a vertex v only if all predecessors of v have pebbles on them and
2. A pebble can be removed from a vertex at any time.

Hopcroft, Paul, and Valiant showed (by means of a clever divide-and-conquer algorithm) the following result about pebbling games. Refer to their paper and text by Balcázar, Diaz, and Gabarró [BDG90] for a proof.

Lemma 5.2. *Every bounded-degree graph with n vertices can be pebbled using $O(n/\log n)$ pebbles, and there is a deterministic algorithm S that does this pebbling in time $O(2^{n^2})$.*

Now we are ready to prove the main theorem (Theorem 5.2) of this section.

Proof of Main Theorem

Proof. Fix $k > 0$, and set $b(n) = n/\log^{(k+2)} n$. Let L be a language in BPTIME(n) and let M be an ℓ-tape, $b(n)$-block respecting probabilistic machine that accepts L in time $t(n) = O(n)$. Set $a(n) = t(n)/b(n)$. Without loss of generality, we can assume that M reads the contents of the random-tape in a one-way manner. The computation graph G_M of M is an edge-labeled

graph defined as follows. The vertex set is $V = \{1, \cdots, a(n)\}$. For $1 \leq i < a(n)$, $\langle i, i+1 \rangle \in E$ with label 0, implying the computation at time segment $i+1$ requires the computation of the time segment i. Assume that the tape heads are numbered $1, \cdots, \ell$. We place an edge from i to j with label $h \in [1, \ell]$ if the following holds: Suppose that during the time segment i the tape head h is in some block b and the next time segment that the tape head h revisits block b is j (*i.e.*, the computation at time segment j requires the content of the block b from the time-segment i). This process defines a *multigraph*.

Given $1 \leq i \leq a(n)$, let $B_1(i), \cdots, B_\ell(i)$ be the blocks that each of the ℓ tape heads are visiting during time segment i. Let $C(i)$ be a string that describes the contents of blocks $B_1(i), \cdots, B_\ell(i)$ and the state q at the end of time segment i. The following observation is crucial.

Observation 5.1. *Suppose a vertex j has predecessors $i_1, \cdots, i_r$ where $1 \leq r \leq \ell$. Then we can simulate the computation of M during time segment j by knowing $C(i_1), \cdots, C(i_r)$. Thus we can compute $C(j)$.*

Using this observation, as in [HPV77], we simulate M by a machine M' as follows. We describe a simulation algorithm that gets a bounded degree graph (degree $\leq \ell + 1$) G as input and attempts to simulate M.

> Call the pebbling algorithm S on graph G. If S places pebble on vertex i, then compute $C(i)$ and store $C(i)$. If S removes pebble from a vertex i, then erase $C(i)$.

Note that a priori, we do not know the correct computation graph G_M. We will first assume that the correct computation graph G_M is known to us and thus can be given as input to the above algorithm. Latter we will remove this restriction. By Observation 5.1, it follows that the above algorithm correctly simulates M (under the assumption that G_M is known). Now we bound the space, time, number of passes, and number of random bits required by the above simulation algorithm (under the assumption that G_M is known). We start with the following two claims.

Claim 5.1. *The total space used by the above simulation is $O\left(\frac{a(n)b(n)}{\log a(n)} + 2^{a^2(n)}\right)$.*

Proof. We are assuming that the graph G_M is known. Now the pebbling strategy places a pebble on a vertex i only when all its predecessors have pebbles on them. Thus, by the above observation, we can compute $C(i)$ when a pebble is placed on vertex i. Note that $C(i)$ can be described using $\ell b(n)$ bits. Since for each pebble we store $C(i)$ (for some i), the total space used is the number of pebbles times $\ell b(n)$. Now the number of pebbles used is $O(a(n)/\log a(n))$ as the size of the graph is $a(n)$. So the total space used by the simulating machine is $O\left(a(n)b(n)/\log a(n)\right)$ plus the space needed by the pebbling strategy S. Since the pebbling strategy runs in $O(2^{a^2(n)})$ time, the space used by the pebbling strategy is bounded by $O(2^{a^2(n)})$.

Note that we can reuse the space for each graph, and thus the total space remains same even when we remove our assumption. □

Claim 5.2. *The above simulation algorithm makes at most $O(2^{a^2(n)})$ passes over the random tape.*

Proof. As before, we are assuming that the graph G_M is known. Even though M reads the contents of the random tape in a one-way manner, the simulating machine may have to read some bits multiple times. Consider a block i on the random tape, the simulating machine needs to access the contents of that block each time the pebbling algorithm S places a pebble on vertex i. If block i is to the left of the current tape head position (of the random tape), then the simulating machine can make a pass over the random tape and access block i. Since the pebbling algorithm takes time $O(2^{a^2(n)})$, the total number of times a pebble is placed on any vertex is at most $O(2^{a^2(n)})$. Thus the simulating machine makes at most $O(2^{a^2(n)})$ passes over the random tape. □

Now we address the assumption that the computation graph G_M is known.

Observation 5.2. *Suppose that G is not the correct computation graph. If the above simulation algorithm gets G as input, then it will discover that G is not the correct computation graph, using $O(\frac{a(n)b(n)}{\log a(n)} + 2^{a^2(n)})$ space and by making $O(2^{a^2(n)})$ passes over the random tape.*

Proof. We can discover whether a graph G is the correct computation graph or not as follows. Observe that given any possible computation graph G, we can compute for each tape the block number the tape-head visits after each time-segment. This information can be stored using $O(a(n)\log a(n))$ bits. Also note that any correct computation graph must have edges of the form $(i, i+1)$, $1 \leq i < a(n)$. Thus, a pebble cannot be placed on vertex $(i+1)$ before placing a pebble on vertex i. This ensures that we will simulate $(i+1)$st time segment only after ith time segment is simulated. Now if G is an incorrect computation graph, then the following scenario happens: The tape-head positions computed based on G will claim that after time-segment i, tape head h is in block b; however, while simulating M, this does not happen. At this point, we will discover that G is not a correct computation graph. □

So our final simulation of M proceeds as follows: Iterate through all possible computation graphs; for each graph G, attempt to simulate M using the above algorithm. If it discovers that G is not a correct computation graph, then proceed to next graph.

By Claim 5.1 and Observation 5.2, each iteration needs $O(\frac{a(n)b(n)}{\log b(n)} + 2^{a^2(n)})$ space. Since we can reuse space from one iteration to the next iteration, total space is bounded by $O(\frac{a(n)b(n)}{\log b(n)} + 2^{a^2(n)})$. By Claim 5.2 and Observation 5.2, each iteration can be done making $2^{a^2(n)}$ passes. Since there are at most $2^{a^2(n)}$ possible computation graphs, the total number of passes is $2^{2a^2(n)}$.

By plugging in the values of $a(n)$ and $b(n)$ from above, we obtain that the space used by the simulating machine is $O(n/\log^{(k+3)} n)$ and the number of passes is $O(\log^{(k)} n)$. Finally, note that the number of random bits used by the simulating machine remains same as the number of random bits used by M, i.e., $O(n)$. This completes the proof of the theorem. □

5.2 Simulating Multiple Passes with Single Pass

An obvious question at this point is the following: Can we simulate multipass probabilistic machines with traditional one-pass probabilistic space bounded machines? The main result of this section shows that passes can be traded for space. This helps us to obtain an upper bound on the deterministic space to simulate a multipass probabilistic space-bounded machine. We first start with the following lemma.

Lemma 5.3. *If a language L is in $k(n)$-pass* BPSPACE$[s(n), r(n)]$*, then there is a probabilistic $O(k(n)s(n))$-space bounded machine N that has one-way access to the random tape and for every $x \in \Sigma^n$,*

$$\Pr[N(x) = L(x)] \geq \frac{1}{2} + \frac{1}{2^{O(k(n)s(n))}}.$$

Moreover, N uses $O(r(n) + k(n)s(n))$ random bits.

Proof. Let M be a bounded-error, probabilistic, $k(n)$-pass, $s(n)$-space-bounded machine that accepts L using $r(n)$ random bits. Consider the following machine N:

Procedure N

1: Input x, $|x| = n$.

2: Let C_0 be the initial configuration of M on input x.

3: Uniformly at random pick $k(n) - 1$ many configurations $C_1, C_2, \cdots, C_{k(n)-1}$, and store the configurations on the work tape.

4: For $0 \leq i \leq k(n) - 1$, in parallel DO

   (a) Simulate one pass of M with C_i as starting configuration.

   (b) Let C_i' be the configuration after this pass.

5: If there exists an i, $0 \leq i \leq k(n) - 2$, such that $C_i' \neq C_{i+1}$, then accept with probability $1/2$ and reject with probability $1/2$.

6: Otherwise, accept x if and only if $C_{k(n)-1}'$ is an accepting configuration.

Since each configuration of M is of length $O(s(n))$, the space needed to store all of C_i's is $O(k(n)s(n))$. Clearly, Step 4 can be done by accessing the random-tape in a one-way manner. The total space needed for all $k(n)$ simulations in Step 4 is $O(k(n)s(n))$. Thus N is an $O(k(n)s(n))$-space bounded machine that reads the random tape in a one-way manner. Note that the number of random bits used by the above machine is the number of random bits used in Step 3 (which is $O(k(n)s(n))$ plus the number of random bits used by M. Thus N uses $O(r(n) + k(n)s(n))$ random bits.

Now we bound the success probability of N. Let $x \in L$. For a fixed random string r, let $D_1^r, D_2^r, \cdots, D_{k(n)-1}^r$ be the configurations of $M(x)$ after each pass when r is written on the random tape. Consider the behavior of N when r is written on its random tape. Note that the behavior of $N(x)$ coincides with the behavior of $M(x)$ when for every $1 \leq i \leq k(n)-2$, C_i equals D_i^r. The probability of this event happening is exactly $1/2^{O(k(n)s(n))}$. When this event does not happen, N accepts x with probability $1/2$. Thus

$$\begin{aligned}
\Pr[N \text{ accepts } x] &= \frac{1}{2^{r(n)}} \sum_{r \in \Sigma^{r(n)}} \Pr[\ N \text{ accepts } x \text{ with } r \text{ on random tape}] \\
&= \frac{1}{2^{r(n)}} \sum_{r \in \Sigma^{r(n)}} \Pr[\ N \text{ accepts } x \mid \forall i C_i = D_i^r] \cdot \Pr[\forall i C_i = D_i^r] \\
&\qquad + \Pr[N \text{ accepts } x \mid \exists i C_i \neq D_i^r] \cdot \Pr[\exists i, C_i \neq D_i^r] \\
&= \frac{1}{2^{r(n)}} \sum_{r \in \Sigma^{r(n)}} \Pr[M \text{ accepts } x \text{ with } r \text{ on random tape}] \cdot \frac{1}{2^{O(k(n)s(n))}} \\
&\qquad + \frac{1}{2}\left(1 - \frac{1}{2^{O(k(n)s(n))}}\right) \\
&= \frac{1}{2^{O(k(n)s(n))}} \Pr[M \text{ accepts } x] + \frac{1}{2}\left(1 - \frac{1}{2^{O(k(n)s(n))}}\right) \\
&\geq \frac{1}{2} + \frac{1}{6 \cdot 2^{O(k(n)s(n))}}
\end{aligned}$$

This completes the proof. □

Using above Lemma, we obtain the following.

Theorem 5.3. *$k(n)$-pass* BPSPACE$(s(n)) \subseteq$ BPSPACE$(k(n)s(n))$.

Proof. Let L be a language that is accepted by a $k(n)$-pass BPSPACE$(s(n))$ machine M. By definition, this machine uses $2^{O(s(n))}$ random bits. Thus by Lemma 5.3, there is an $O(k(n)s(n))$-space bounded, one-pass, probabilistic machine N that uses $O(2^{O(s(n))}+k(n)s(n))$ random bits, and

$$Pr[L(x) = N(x)] \geq \frac{1}{2} + \frac{1}{2^{O(k(n)s(n))}}.$$

We can amplify the success probability of N to $2/3$ by simulating it $2^{O(k(n)s(n))}$ times and taking the majority vote. This will use $2^{O(k(n)s(n))}$ random bits. Thus we obtain a $O(k(n)s(n))$-space bounded machine that uses $2^{O(k(n)s(n))}$ random bits. Thus L is in BPSPACE$(k(n)s(n))$. □

Corollary 5.1. *$O(1)$-pass* BPL = BPL.

We relate the above lemma to the result of [DPS11].

Theorem 5.4. *Let M be a $k(n)$-pass, $s(n)$-space bounded machine that uses $r(n)$ random bits. Any pseudorandom generator that fools $k(n)s(n)$-space bounded machines (that read their input in a one-way manner) running on $r(n)$-bit input strings also fools M.*

Their result states that to deterministically simulate $k(n)$-pass, $s(n)$-space bounded probabilistic machines, a pseudorandom generator against standard $O(k(n)s(n))$-space bounded probabilistic machine suffices. Lemma 5.3 can be interpreted as an explanation of their result, as it shows that any $k(n)$-pass, $s(n)$-space bounded machine can indeed be simulated by a standard $O(k(n)s(n))$ space bounded machine.

Next, we consider the main result of this section: a deterministic simulation of the probabilistic class $k(n)$-pass BPSPACE($s(n)$). This is a subclass of BPSPACE($k(n)s(n)$) by Theorem 5.3. By the celebrated results of Nisan [Nis92] and Saks and Zhou [SZ99], it follows that this class is a subset of DSPACE($k^{3/2}(n)s^{3/2}(n)$). Observe below that we can get rid of the polynomial factor off the number of passes, more formally, we show the following.

Theorem 5.5. *$k(n)$-pass* BPSPACE($s(n)$) $\subseteq$ DSPACE($k(n)s^{3/2}(n)$).

Proof. Let L be a language that is accepted by a $k(n)$-pass BPSPACE($s(n)$) machine M. Since M halts in $2^{O(s(n))}$ time, $k(n)$ is bounded by $2^{O(s(n))}$ and M uses $2^{O(s(n))}$ random bits. Thus, by Lemma 5.3, there is an $O(k(n)s(n))$-space bounded, one-pass, probabilistic machine N that uses $2^{O(s(n))}$ random bits and

$$Pr[L(x) = N(x)] \geq \frac{1}{2} + \frac{1}{2^{O(k(n)s(n))}}.$$

Saks and Zhou [SZ99], building on Nisan's [Nis92] work showed that any language accepted by a probabilistic machine using $O(s(n))$-space, $2^{O(r(n))}$ random bits, with success probability as low as $1/2 + 1/2^{O(s(n))}$, is in deterministic space $O(s(n)r^{1/2}(n))$. Applying this to our case, we obtain that N can be simulated by a deterministic space-bounded machine that uses $O(k(n)s^{3/2}(n))$. □

Recall that our hypothesis in Theorem 5.1 states that for some $k > 0$ if the complexity class $\log^{(k)} n$-*pass* BPL($\log n \log^{(k+2)} n$) is in P. We obtain the following upper bound for this class, by applying Lemma 5.3 and the techniques of Saks and Zhou [SZ99].

Corollary 5.2. *For any constants $k \geq 3$ and $\epsilon > 0$,*

$$\log^{(k)} n\text{-pass BPL}[\log n \log^{(k+3)} n] \subseteq \text{DSPACE}(\log n (\log\log n)^{\frac{1}{2}+\epsilon}) \subseteq \text{DTIME}(n^{(\log\log n)^{1/2+\epsilon}}).$$

Proof. Let L be a language that is accepted by a $\log^{(k)} n$-pass logspace machine M that uses $O(\log n \log^{(k+3)} n)$ random bits. Thus, by Lemma 5.3, there is an $O(\log^{(k)} n \times \log n)$ space-bounded, one-pass, probabilistic machine N that uses $O(\log n \log^{(k+3)} n)$ random bits and

$$Pr[L(x) = N(x)] \geq \frac{1}{2} + \frac{1}{2^{O(\log^{(k)} n \times \log n)}}.$$

Using the result of Saks and Zhou [SZ99] from Theorem 5.5, we obtain that N can be simulated by a deterministic space-bounded machine that uses space in the order of

$$\log n \times \log^{(k)} n \times (\log\log n + \log\log^{(k+3)} n)^{1/2} \leq \log n (\log\log n)^{\frac{1}{2}+\epsilon}$$

for $k > 3$ and any $\epsilon > 0$. The second inclusion is straightforward. □

5.3 Deterministic Simulation of Multipass Machines with Linear Advice

Fortnow and Klivans [FK06] showed that standard (one-pass) randomized logspace machines (BPL) can be simulated by deterministic logspace machines that have access to a linear amount of advice. I.e., they showed that BPL is a subset of L/$O(n)$. On the other hand, using Adleman's technique [Adl78], it can be shown that randomized logspace machines with two-way access to the random tape can be simulated in deterministic logspace using a polynomial amount of advice [Nis93]. Thus, any multipass, randomized logspace machine can be simulated in deterministic logspace with polynomial amount of advice. Can we bring down the advice to linear? We first show that this is possible with a small increase in space.

Let M be an $O(\log n)$-pass, randomized logspace machine. By Theorem 5.3, M can be simulated by a one-pass randomized machine that uses $O(\log^2 n)$ space, and by applying the techniques of Fortnow and Klivans [FK06], it follows that M can be simulated in deterministic

space $O(\log^2 n)$ with linear advice. Below we show that we can improve the space bound of the deterministic machine to $O(\log n \log\log n)$. More formally, we prove the following general theorem.

Theorem 5.6. *For any $k(n) \in \omega(1)$, a $k(n)$-pass $s(n)$-space bounded randomized machine that uses $R(n) = 2^{r(n)}$ random bits can be simulated by a deterministic machine that uses space $O(s(n) + r(n)\log(k(n)s(n)))$ and an advice of size $O(r(n)k(n)s(n) + n)$. I.e.,*

$$k(n)\text{-pass BPSPACE}[s(n), 2^{r(n)}] \subseteq \text{DSPACE}(s(n) + r(n)\log k(n)s(n))/O(r(n)k(n)s(n) + n).$$

Before we prove the above theorem, we note some of its direct consequences. Since $r(n) = O(s(n))$, the following is straightforward.

Corollary 5.3. *$O(1)$-pass* BPL $\subseteq$ L/$O(n)$.

This clearly improves the result of Fortnow and Klivans. The above corollary can also be obtained by combining Fortnow-Klivans' result with Corollary 5.1.

Corollary 5.4. *For every constant $k > 0$, $O(\log^k n)$-pass* BPL $\subseteq$ DSPACE$(\log n \log\log n)/O(n)$.

That is, with the increase of only a factor of $\log\log n$ in space by the simulating machine, we can simulate a polylog-pass probabilistic logspace machine.

Corollary 5.5. *For every $0 < \epsilon < 1$, n^ϵ-pass* BPL $\subseteq$ DSPACE$(\log^2 n)/O(n)$.

Corollary 5.5 can be interpreted as follows: with the increase of only $\log n$ factor in space compared to Fortnow-Klivans, we are able to accommodate n^ϵ many passes.

First, we provide the proof idea of Theorem 5.6, following Fortnow-Klivans.

Proof Sketch

For simplicity, suppose we have a $\log n$-pass logspace probabilistic machine M that uses n^2 random bits and has error probability 1/3. Our goal is to simulate it by an $O(\log n \log\log n)$ space-bounded deterministic machine that uses advice of size only linear in n. Note that if we convert M to a 1-pass $O(\log^2 n)$ space-bounded machine using Theorem 5.3 and apply the following method, we will get an $O(\log^2 n)$ space-bounded machine instead.

First, we apply Adleman's technique on M; on input of length n, reduce the error of the machine to less than $1/2^n$ by running M $O(n)$ times, and then fix a "good" advice string that works for all inputs. This requires a logarithmic space machine, but polynomial in n amount of advice. To reduce the advice, we apply Fortnow-Klivans' idea. Use the pseudorandom bits from Nisan's generator to reduce the error. To get n pseudorandom strings, we need n seeds, each of length $O(\log^3 n)$. This is because our generator needs to fool a $\log n$-pass logspace machine, and by Theorem 5.3, we can achieve this by fooling a one-pass $O(\log^2 n)$-space machine instead. Since, we need to run M roughly n times to reduce the error down to 2^n, we need $O(n \log^3 n)$ size seed, which again increases the advice size (we can actually do slightly better, we can run for $n/\log^2 n$ times and still reduce the error exponentially, and thus, the advice length will be $O(n \log n)$). Clearly, we cannot store all the seeds. So the idea is to generate them online, only when the generator needs them. So we perform a random walk $(d_1, d_2, \ldots, d_n, v_0)$ of length n on a constant-degree expander with roughly $2^{\log^3 n}$ many nodes and assume each vertex as the seed of Nisan's generator. In this case, we need to store only the initial vertex of size $\log^3 n$ bits and $O(n)$-length random walk. There is still one concern. Nisan's theorem says that we need $O(\log^2 n)$ space to generate each pseudorandom bit, given the seed. We show that using convolution family of 2-universal hash function, we can do it in $O(\log n \log\log n)$ space. The idea is to compute each bit only when asked by Nisan's generator.

Now the challenge is to generate each seed in a space-efficient manner from the random walk and the advice. A special encoding of the expander of Margulis-Gaber-Galil comes handy. Roughly speaking, we represent each vertex by an $O(\log n)$ length tuple whose each entry is an integer modulo a prime number. The reason this representation is helpful is because each bit of the neighborhood vertex can be computed in constant space by simple modular arithmetic, given the encoding of the vertex and the index of the neighborhood vertex. But how do we compute the bits of the vertex (seed) from the above encoding? The idea is to start the walk every time from the first vertex (stored as advice) and redo the random walk (also stored as advice) until now, once a bit of a vertex is expected by the generator. That's why it's necessary to store them so that we can perform the same walk again and again. To successfully implement this, we need to maintain a global index to count the number of walks already performed and an

index of the vertex (seed) requested by Nisan's generator, both of which take at most $O(\log n)$ bits. This method requires multiple access to the advice, but can be performed in logspace. This space-efficient random walk is due to Gutfreund and Viola [GV04].

However, our encoding of the expander graph does not directly provide the seed. During the simulation, suppose Nisan's generator requires the ith bit of the vertex v_j of the expander at the jth step of random walk. We use an $O(\log n)$ space-computable algorithm, obtained as a consequence of a result of Chiu, Davida, and Litow [CDL01], to get each bit of the seed, But this algorithm requires the encoding of the vertex v_j. How does it get it? It basically starts the random walk every time it needs to find a bit of a vertex. The details follow in the next section.

Finally, note that since we store the seed, we can reuse the same random string for multiple passes of the computation, and each time the actual computation space is just $O(\log n)$.

Now we give the proof details.

First, we provide some necessary technical background in this subsection. Below, we give the formal definition of Nisan's space-efficient pseudorandom generator and some of its key properties. Then we give the space-efficient encoding of constant-degree expander graphs.

Nisan's Generator

Suppose we have a one-way $s(n)$-space bounded probabilistic machine that uses $R(n) = 2^{r(n)}$ random bits and we want to derandomize it using Nisan's generator [Nis92]. The seed of Nisan's generator consists of a string $x \in \{0,1\}^{s(n)}$ and $r(n)$ many hash functions represented by $h_1, h_2, \ldots, h_{r(n)}$ where $h_i(x) \in \{0,1\}^{s(n)}$. Both x and each h_i are picked uniformly at random. Then the pseudorandom bits of Nisan's generator can be represented by (in order)

$$x, h_1(x), h_2(x), h_1h_2(x), h_3(x), \ldots, h_1h_2 \ldots h_{r(n)}(x).$$

Each hash function (or, the composite hash function) generates $s(n)$ bits, but we require only the first bit from each of them. Now note that in order to get hold of the last bit of the generator's output, $h_1h_2 \ldots h_{r(n)}$, we need to first compute $y = h_{r(n)}(x)$, store its value, and then compute $y' = h_{r(n)-1}(y)$, and so on until we have computed the entire composite function.

Trivially, given x and the hash functions, we only need to store the intermediate computations y, y', etc. Since we can reuse the space, each of them consumes only $s(n)$ space. Thus each pseudorandom bit in Nisan's generator can be computed in space $s(n)$, given the seed. Note that we need 2-way access to the seed. The first step in proving Theorem 5.6 is to show that given multiple access to the seed, we can reduce the computation space of Nisan's generator (provided $r(n) = O(s(n))$.

Claim 5.3. *Given seed of Nisan's generator in a 2-way read-only tape, we can compute each pseudorandom bit in space $O(s(n) + r(n) \log s(n))$.*

Proof. Assume that the hash functions for the seed of Nisan's generator are picked uniformly at random from the following convolution family of hash functions:

$$H = \{(a * x) \oplus b \mid a \in \{0,1\}^{2s(n)-1}, b \in \{0,1\}^{s(n)}\},$$

where $x \in \{0,1\}^{s(n)}$, ith bit of $(a * x)$ is given by $c_i = \sum_{j=1}^{s} a_{i+j-1}x_j \pmod 2$, and $c \oplus b$ represents the bitwise exclusive-or of two $s(n)$-length strings c and b. Suppose we want to compute the last bit of Nisan's generator $h_1 h_2 \dots h_{r(n)}$ as before. The idea is to compute each bit of $h_i(z)$ only when it is required by the computation of the hash function h_{i+1}. We start with the computation of $h_{r(n)}$. It requires the values a_i's and b_i's which are stored in the tape, along with the value of $h_{r(n)-1}$. But we have not computed this value yet. So we halt the computation of $h_{r(n)}$, store its index, and pass the computation to $h_{r(n)-1}$. Note that it takes only $\log s(n)$ space to store the index. Similarly, the computation of $h_{r(n)-1}$ has its a_i's and b_i's stored in the tape, but the value of $h_{r(n)-2}$ is missing. So we store its index as before and pass the computation to the next hash function, and so on. Finally, h_1 has both x and its a_i's and b_i's stored in the 2-way read-only tape and can perform the computation. It only takes constant space to compute a bit asked by h_2 and passes the value to it, which again completes its computation and passes the value asked by h_3, and so on. Clearly, we need to store $r(n)$ many indexes, which is $O(r(n) \log s(n))$ bits, to compute any pseudorandom bit in Nisan's generator. This, along with the computation space of the simulating machine, gives us the required space bound. □

Space-efficient Expander Encoding

Remember the constant-degree expander of Margulis-Gaber-Galil from Section 2.7. We use the following encoding of the expander [FK06]. We encode each node (x, y) of the expander G_N by $O(\log N)$ bits and the index of each of its neighbors by 3 bits. Then it's easy to see that given a vertex (x, y) and an index i of a neighbor (input), every bit of the neighborhood vertex can be computed in constant space. Further, to make sure that the machine head run *one-way* (and we assume, left-to-right) on the input tape while doing the neighborhood computation, we need to make sure that the above modular arithmetic can be performed only with the one-way access to the input tape. For this, we assume that the encoding for vertex (x, y) follows bit-reversals, that is, $\langle x_1, y_1, x_2, y_2, \ldots, x_{\log m}, y_{\log m}\rangle$, where x_1 and y_1 are the least significant bits of x and y respectively.

We now define the following random walk on the expanders due to Ajtai, Komlos, and Szemeredi [AKS87].

Definition 5.3. We define a *random walk of length* $t(n)$, represented by $\langle d_1, d_2, \ldots, d_{t(n)}, v_0\rangle$, on the expander graph such that the walk starts at vertex v_0 and goes to the neighbor of it indexed by d_1 (say v_1), then the neighbor of v_1 indexed by d_2, and so on.

Note that the order in which the sequence of a random walk is encoded is important. It helps us to design a one-way machine that computes the neighborhood vertices efficiently.

Now assume that we have a constant-degree expander $G = (V, E)$ of N vertices from Theorem 2.4. Recall that we can describe each vertex (x, y) of G from a set $\mathbb{Z}_m \times \mathbb{Z}_m$. Assume $\ell(n) = r(n)k(n)s(n)$ and $N = 2^{\ell(n)}$. Note that $\ell(n) < n$ for $k(n)s(n) << n$ and a polynomial in n function $R(n) = 2^{r(n)}$. Our goal is to represent each vertex of the expander by the residues of primes and perform the neighborhood computation using Chinese Remaindering theorem. Consider the first $c.k(n)s(n)$ many primes $p_1, p_2, \ldots, p_{c.k(n)s(n)}$ (for sufficiently large integer c) each of size $O(r(n))$ bits. Thus the primes together consumes $O(\ell(n))$ space. For our choice of $k(n), s(n)$, and $r(n)$, we can enumerate these primes within the range of integers $[n]$ (by Prime Number theorem). Let m be the largest integer smaller than $\prod_{i=1}^{c.k(n)s(n)} p_i$. Then we can represent each vertex (x, y) of G by a pair of tuples $(x_1, x_2, \ldots, x_{c.k(n)s(n)}) \times$

$(y_1, y_2, \ldots, y_{c.k(n)s(n)})$, where each $x_i, y_i \in \mathbb{Z}_{p_i}$. Thus, each x_i and y_i can be represented by $O(r(n))$ bits and each vertex consumes $O(\ell(n))$ space.

Proof of Main Theorem

Now we are ready to formally prove the main theorem of this section.

Proof. Assume $\ell(n) = r(n)k(n)s(n)$. Suppose we have a $k(n)$-pass $s(n)$ space-bounded probabilistic machine that uses $R(n) = 2^{r(n)}$ random bits and errs with probability at most $1/3$. First, we convert the multipass machine to a one-pass $k(n)s(n)$-space machine A using Theorem 5.3. Thus derandomizing A will automatically derandomize the multipass machine. Now, uniformly at random pick a vertex v_0 in G and the neighborhood indexes $d_1, d_2, \ldots, d_{2n}$ for a random walk of length $2n$ on G. Store v_0, the indexes, and the prime numbers (to represent the expander G) as advice, which consume $O(\ell(n) + n)$ space together. Note that by storing these information, we have fixed our random walk for successive computations. This is crucial as we need to perform the same walk multiple times. Now given this advice, the one-pass machine A, and the above encoding of the expander G, our goal is to simulate A by a deterministic machine in $O(s(n) + r(n) \log k(n)s(n))$ space.

The idea is to use Nisan's generator $2n$ times and simulate A $2n$ times using those pseudorandom strings and perform majority test on them. We keep a global counter ($O(\log n)$ space) to count the number of times the simulation is performed. Suppose Nisan's generator requires a pseudorandom bit during jth run. It stores the index of the required bit ($r(n)$ bits) and goes on to compute it. Now given the seed, we can compute this bit in $O(s(n) + r(n) \log k(n)s(n))$ space by Claim 5.3. Note that each vertex of the expander acts as a seed for Nisan's generator. But the vertices (and edges) of the expander are not stored in order to reduce the advice size. So we start a random walk from v_0 every time, follow the indexes $\{d_j\}$'s stored as advice, while computing the bits of the next vertices through efficient edge computation of Theorem 2.4, and stop whenever we reach jth vertex in the walk and have computed the required pseudorandom bit. To compute that bit, we need to know the bits of the vertex (seed). Suppose at some point, we need the ith bit of the vertex v_j. Now it needs to be recovered from the previously

mentioned encoding of the expander. At this time we use the following theorem, which is a consequence of a result of Chiu-Davida-Litow [CDL01].

Lemma 5.4. *Suppose Nisan's generator asks for the ith bit of a vertex v_j on G. Then there is an algorithm D such that on input the Chinese remaindering representation of the vertex v_j: $(a_1, a_2, \ldots, a_{c.k(n)s(n)}) \times (b_1, b_2, \ldots, b_{c.k(n)s(n)})$, primes $p_1, p_2, \ldots, p_{c.k(n)s(n)}$, and the index i, D outputs the ith bit of the vertex v in $O(r(n) + \log \ell(n))$ space.*

Now recall that we only store the initial vertex and hence we need to compute the residual representation of the vertex v_j. To compute each of these a_i's and b_i's for vertex v_j, we follow our previous strategy, i.e., we compute them only when they are required by the above lemma. Note that each (a_i, b_i) for v_j depends on the pair (a_{i-1}, b_{i-1}), initial vertex v_0 and the index j of the walk. So every time we need a_i's and b_i's, we store (a_{i-1}, b_{i-1}), start the walk from the beginning using advice, compute this pair, and reuse it for next pair. This again requires $O(r(n) + \log \ell(n))$ space. So the total space required is bounded by $O(\log n + s(n) + r(n) + \log \ell(n) + r(n) \log k(n)s(n)) = O(s(n) + r(n) \log k(n)s(n))$.

Finally, note that to fix an advice according to Adleman's technique, we need to make sure that error of the machine is reduced to $1/2^n$ for any input string of length n . Here we have two error components. First, the error due to $t = 2n$ length random walk on the Expander G which is $1/2^t = 1/2^{2n}$. The other one is due to Nisan's generator which fools a $O(k(n)s(n))$ space machine and thus has error $1/2^{O(k(n)s(n))}$. Since we apply the random walk on top of it, the error becomes $1/2^{O(tk(n)s(n))} \leq 1/2^{2n}$. Thus, the total error is bounded by $1/2^n$ for any fixed string of length n, which by the union bound over all possible strings of length n is bounded away from 1. □

5.4 Deterministic Amplification in Multipass Machines

Deterministic amplification is the process of reducing the error probability of a randomized algorithm by means of adding more randomness. The goal is to add as few randomness as possible. Bar-Yossef, Goldreich, and Wigderson [BYGW99] showed that any probabilistic algorithm that uses $s(n)$ space, $R(n)$ many random bits, and has an error probability $\epsilon < 1/2$

can be amplified deterministically to a probabilistic algorithm that uses $O(t(n)s(n))$ space, $R(n) + O(t(n))$ random bits, and errs with probability $\epsilon^{\Omega(t(n))}$, for any function $t(n)$. This was the first result showing a non-trivial randomness-efficient deterministic amplification for a space-bounded machine. We use their ideas in the multipass settings to prove the following theorem:

Theorem 5.7. *Any $k(n)$-pass $s(n)$-space bounded probabilistic algorithm that uses $R(n)$ random bits and errs with probability $\epsilon < 1/2$ can be deterministically amplified to a $k(n)$-pass $O(t(n)s(n))$-space probabilistic algorithm that uses $R(n) + O(t(n))$ random bits and has an error probability $\epsilon^{\Omega(t(n))}$, for any function $t(n)$.*

As a straightforward corollary, we get the following randomness-efficient deterministic amplification for an $O(\log n)$-pass BPL machine.

Corollary 5.6. *For $R(n) > \log n$, any $O(\log n)$-pass* BPL *algorithm using $R(n)$ random bits can be deterministically amplified to an $O(\log n)$-pass* BPSPACE$(\log^2 n)$ *algorithm using $O(R(n))$ random bits with error probability* $1/\mathrm{poly}(n)$.

Proof of Main Theorem

Before we prove the main theorem of this section, we discuss various properties of expander graphs that are used in our proof. First, consider the expander encoding from Section 5.3. With this encoding at hand, we perform the following space-efficient one-way random walk on the expander, following Bar-Yossef *et al* [BYGW99].

Lemma 5.5. *There is a family of 5-regular expanders $\{G_N\}$ such that given an initial node (x, y) and a random walk of length $t(n)$ on G_N as input, there is an algorithm B that computes each bit of the end-vertex of the random walk in $O(t(n)s(n))$ space with only* one-way *access to the input.*

Proof. Consider the 5-regular expander graph G with N nodes from Theorem 2.4. Also consider the expander encoding scheme and the random walk on G from Section 5.3. Our goal is to compute the bits of the end-vertex $v_{t(n)}$ at the end of $t(n)$-length random walk $\langle d_1, d_2, \ldots, d_{t(n)}, v_0 \rangle$.

Now recall that we can describe each vertex v_i of G by a pair (x, y) from set $\mathbb{Z}_m \times \mathbb{Z}_m$. Since we are given the random walk of length $t(n)$ as the input of B, its input tape will consist of the tuple $\langle d_1, d_2, \ldots, d_{t(n)}, x_1, y_1, x_2, y_2, \ldots, x_{\log m}, y_{\log m} \rangle$ based on the aforementioned expander encoding scheme and the random walk on G. Then the algorithm B works in two stages. First, it scans through all the $t(n)$ neighborhood indexes $d_1, \ldots, d_{t(n)}$ from left to right and store them in a work tape (so as to access them multiple times in future), thus accumulating $O(t(n) \log 5)$ space. The second step is the actual computation step where we read the bits of vertex (x, y) only "on-demand" by the computation. We explain this in more detail in the next paragraph.

B starts computing the first bit of $v_{t(n)}$ with the last index in the sequence, $d_{t(n)}$. Whenever it needs a bit of the previous vertex $v_{t(n)-1}$, whose $d_{t(n)}$th neighbor is $v_{t(n)}$, it halts the computation with $d_{t(n)}$, stores the configuration, and starts computing bits of $v_{t(n)-1}$ with the index $d_{t(n)-1}$. Similarly, whenever it needs a bit of the previous vertex $v_{t(n)-2}$, it halts the current computation, stores the configuration, and starts computing the bits of vertex $v_{t(n)-2}$ with index $d_{t(n)-2}$, and so on, until it reaches the point where it needs to compute a bit of the vertex v_1, which is the d_1th neighbor of the starting vertex (x, y). At this point, the algorithm reads the bits of (x, y) from input tape one-way, computes the bit of v_1, resumes the computation of v_2, computes the bit of v_2, resumes the computation of v_3, and so on, until the bit for $v_{t(n)}$ is computed.

We had to halt $t(n)$ many computations in the process, thus storing $t(n)$ many configurations, which accounts for $O(t(n)s(n))$ space. Recall that we made sure in the encoding process that the neighboring vertices can be computed with only the one-way access to the input tape. □

We also need the following theorem by Ajtai-Komlos-Szemeredi [AKS87] in our final proof.

Theorem 5.8. *If we perform a $t(n)$-length random walk on the above expander G, given in Theorem 2.4, and use the vertices of it as the source of random bits to perform deterministic amplification of a probabilistic algorithm A with error probability $\epsilon < 1/2$, then the error probability of the simulating probabilistic algorithm is $\epsilon^{\Omega(t(n))}$.*

Now we provide the proof of the main theorem of this section.

Proof of Theorem 5.7. Suppose A is a $k(n)$-pass $s(n)$-space probabilistic algorithm that uses $R(n)$ random bits during its computation and has error probability $\epsilon < 1/2$. Suppose M is a black-box simulator that given $R(n)$ random bits as input, generates $t(n)$ pseudorandom strings $\{r_1, r_2, \ldots, r_{t(n)}\}$ each of length $R(n)$ from a random string of length $\ell(n)$, runs A for $t(n)$ times using these pseudorandom strings, and outputs the majority vote.

First, we show that the simulator M uses a random string of length only $O(R(n)+t(n)\log d)$ to generate those $t(n)$ random strings each of length $R(n)$. To start with, M picks up a sequence $\langle d_1, d_2, \ldots, d_{t(n)}, v_0\rangle$ of length $O(R(n) + t(n))$ uniformly at random from the expander G on N vertices, given by the Theorem 2.4, and performs a random walk on it. Here $N = 2^{R(n)}$ and let each node of the expander be represented by a string from $\{0,1\}^{R(n)}$. Recall that our encoding of the expander helps us to compute each bit of the neighborhood vertex in constant space with only one-way access to the random tape. After each walk on G, we reach a vertex of G, represented by $R(n)$ bits. We use them as the random source r_i for A, for each ith run.

Next, we show that M is a $k(n)$-pass $O(t(n)s(n))$-space computable machine. First, consider only the first pass of each of the $t(n)$ runs of A. Consider the $t(n)$-length random walk of M on G. By Lemma 5.5, we know that each bit of the end-vertex after the $t(n)$-length walk can be computed in $O(t(n)s(n))$ space. Since A computes in space $s(n)$, $t(n)$ runs of A by M assumes another $t(n)s(n)$ space. In the process, M reads the random tape only once and one-way. Now consider the second pass of each of the $t(n)$ runs. Note that once M has picked the random walk of length $t(n)$, they are stored in the random tape and fixed. Thus, if M behaves similarly as in the first pass, and goes over the random tape only once one-way again, it can generate the same set of vertices of G as in the first pass. So M can simulate the behavior of A in second pass exactly for all the $t(n)$ runs. The only thing that we had to store to start each of these computations is the configurations at the end of first pass of $t(n)$ simulations. This again requires $O(t(n)s(n))$ space. This same space can be reused for all the $k(n)$ passes of A for $t(n)$ runs. Thus the total space used by M is $O(t(n)s(n))$ and it goes over the random tape only $k(n)$ times one-way.

Finally, we call the Theorem 5.8 to compute that the error probability of M. Since the random strings $\{r_1, r_2, \ldots, r_{t(n)}\}$ are fixed once M has picked the initial vertex and the $t(n)$-

length random walk, and it simulates A exactly on those strings for all the $k(n)$ passes, the error probability of M is independent of the number of passes. □

5.5 Space Hierarchy in Multipass Machines

One of the primary goals of this chapter is to explore the relationships between probabilistic time and the probabilistic space. In particular, can we show that linear probabilistic time is a strict subset of linear probabilistic space, analogous to the deterministic world [HPV77]? One interesting inclusion result in this direction was shown in Section 5.1, where we showed that BPTIME(n) is a subset of $O(\log^{(3)} n)$-pass BPSPACE($o(n)$). Since there is no known hierarchy theorem for the traditional probabilistic classes, we cannot say directly from this result that linear probabilistic time is a strict subset of multipass linear probabilistic space. However, with the help of a result by Kinne and van Melkebeek [KvM09], we show the following in this section: If allowed a small number of passes, the linear probabilistic time is a strict subset of the linear probabilistic space class under a single bit of advice. More formally,

Theorem 5.9. BPTIME(n)/1 $\subsetneq$ $\log^{(3)} n$-*pass* BPSPACE(n)/1.

The first step in proving this theorem is to observe that Theorem 5.2 can be extended to work even under a single bit of advice to both the simulating and simulated machines.

Observation 5.3. BPTIME(n)/1 $\subseteq$ $\log^{(3)} n$-*pass* BPSPACE($n/\log^{(4)} n$)/1.

At this point, a tempting idea is to use the results of Nisan's or Saks-Zhou to show that for any constant $c > 1$ and $s'(n) \geq s(n)^c$, BPSPACE($s'(n)$) cannot be simulated by a BPSPACE($s(n)$) machine. This result can also be extended under the same number of multiple passes and the amount of advice for both the simulated and simulating machines. But for our need, that is not sufficient. We need it under the condition $s'(n) \in \omega(s(n+1))$. We adapt the result of Kinne and van Melkebeek [KvM09] which gives us the required separation under this weaker condition.

Theorem 5.10. *For every* $k(n) \in \omega(1)$, $\log n \leq s(n) \leq n$, *and* $s'(n) = \omega(s(n+1))$,

$$k(n)\text{-}pass \text{ BPSPACE}(s(n))/1 \subsetneq k(n)\text{-}pass \text{ BPSPACE}(s'(n))/1.$$

Once we have Observation 5.3 and Theorem 5.10, the proof of the main theorem is straightforward. Hence for rest of this section, we focus on proving the above theorem.

The proof is similar to the exposition in [KvM09] and we provide only the proof sketch.

Proof Sketch of Theorem 5.10

For simplicity, we give the exposition for one-pass randomized machines. The first step is to associate each randomized machine M_i with an interval of input lengths $I_i = [n_i, n_i^*]$. These intervals are disjoint so that each input length n belongs to only one interval. We show that N differs from every randomized machine M_i on interval I_i on some length $n \in [n_i, n_i^*]$. By restricting the simulation of M_i's to only $s(n)$ space on input length n gives us the required separation. Note that there are infinitely many possible intervals associated for each machine M_i as there are infinitely many possible encoding of a machine M_i. This is necessary to deal with the asymptotic behaviors of the functions $s(n)$ and $s'(n)$, as usual is the case for other hierarchy theorems. The goal is to show the contradiction in at least one of these intervals for each machine. We skip those details here and assume that we are given those "good" intervals as I_i for each machine M_i.

Now recall the hierarchy theorems in the non-deterministic setting. For the sake of simplicity, we work only with the tally strings. The (non-deterministic) universal simulator D goes over every non-deterministic machine N_i and tries to complement its computation, thereby showing the separation with each of them (given the time or space constraints). But it cannot just complement its computation on every input, rather it selects a range of input lengths as above for every machine N_i and input length and performs a *delayed diagonalization*. That is, for every input length $n \in [n_i, n_i^*)$, $D(n)$ simulates $N_i(n+1)$ and on input $0^{n_i^*}$, D accepts if and only if N_i rejects on input 0^{n_i}. Now D takes exponential time in n_i and thus n_i^* can be chosen to be 2^{an_i} for some appropriate constant $a > 0$ to make it polynomial in n_i^*. Now the claim is that there is a length $n \in [n_i, n_i^*]$ where D and N_i behave differently. Indeed, if D and M_i behave similarly on all input lengths in $[n_i, n_i^*]$, then by the way of construction $D(0^{n_i^*}) = N_i(0^{n_i^*}) = D(0^{n_i^*-1}) = N_i(0^{n_i^*-1}) = \ldots = M(0^{n_i})$, which is a contradiction. Note that we exactly don't know which length it is in the interval.

We adapt the similar proof strategies in the probabilistic setting. But we need to address few concerns. First, there is no known universal simulator for the randomized machines. Further, how do we make sure that the enumerated randomized machines M_i satisfy the promise? Finally, how do we contradict the output of a randomized machine on some input? To address the second question, we assume that we have one single bit advice b_n for each machine M_i for each input length n that basically tells us if the machine M_i satisfies the promise on the input length. If $b_n = 0$, we say that M_i does not satisfy the promise, otherwise it does. Then our strategy is the following. On input 0^{n_i}, N reduces the computation of $M_i(0^{n_i})$ to an instance of a hard language T of length m_i, for every input length $n \in [m_i, n_i^*)$, $N(n)$ simulates $N_i(n+1)$, and on input $0^{n_i^*}$, N simulates $L(0^{m_i})$.

For the details of the proof, please refer to [KvM09].

5.6 Conclusions

This chapter established that time-efficient derandomization of probabilistic, logspace machines that make a non-constant passes over the random tape yields a new non-trivial derandomization of probabilistic time. This result suggests that it is fruitful to further study multipass, probabilistic, space-bounded machines. There are many interesting questions to solve involving multipass machines. Can we show that a multipass machine with linear pass over its random tape can also be simulated by a deterministic logspace machine with linear advice? On the contrary, can we improve on the amount of advice bits of the simulating machine while simulating a standard BPL class? Note that this potentially requires improvement over the Adleman's techniques.

Another interesting question that arises is on error reduction. Let M be a $k(n)$-pass, $s(n)$-space bounded, bounded-error probabilistic space-bounded machine with error probability less than $1/3$. Can we reduce the error probability to $1/2^{e(n)}$ for a polynomial $e(n)$ without substantial increase in passes and space used? Note that by increasing the number of passes to $O(k(n)e(n))$ this is indeed possible. On the other hand, if we were not to increase the number of passes, then by increasing the space to $O(e(n)s(n))$ we can achieve the same reduction in error probability. Can we do better? Can we reduce the error probability to $1/2^{e(n)}$ while

keeping the number of passes to $O(k(n))$ and the space bound to $O(s(n))$?

Finally, can we prove the hypothesis of the main theorem of this chapter, Theorem 5.1? This will have many interesting consequences in the field of derandomization.

CHAPTER 6. DERANDOMIZATION UNDER BRANCHING PROGRAM LOWER BOUNDS

Circuit lower bounds imply the existence of pseudorandom generators and the pseudorandom generators can be used to derandomize various probabilistic time-bounded classes. Given the connection between circuit lower bounds and derandomization, it is natural to ask if we can derandomize a probabilistic algorithm without proving circuit lower bounds. In this chapter, we address this question. In particular, we ask the following question: Is there a constant $\delta > 0$ such that $\mathrm{BPTIME}(t(n)) \subseteq \mathrm{DTIME}(2^{t(n)^\delta})$ under the branching program lower bounds? Or, is it even possible to show that $\mathrm{BPTIME}(t(n))$ can be simulated deterministically in time $2^{o(t(n))}$ under branching program lower bounds? Branching programs capture nonuniform space-bounded computations and are believed to be a weaker model of computation than general Boolean circuits. For example, it is known that polynomial-size branching programs can be simulated by polynomial-size, $O(\log^2 n)$-depth Boolean circuits. Hence the hardness assumption on branching programs is believed to be a weaker condition than the typical circuit complexity lower bound assumptions.

Previous Work

There are results indicating that circuit lower bounds may be necessary to achieve derandomization. Buhrman, Fortnow, and Thierauf [BFT98] showed that $\mathrm{MA}_{\mathrm{EXP}}$ does not have polynomial-size circuits. From this it follows that if Promise-BPP can be simulated in deterministic polynomial time, then $\mathrm{MA}_{\mathrm{EXP}}$ is same as NEXP and thus, NEXP does not have polynomial-size circuits. Hence a *complete* derandomization of Promise-BPP yields circuit lower bounds. The natural question is: Can we prove a weak derandomization without proving circuit lower bounds? In a seminal work, Impagliazzo, Kabanets, and Wigderson [IKW02] showed

that almost any derandomization of Promise-BPP implies the existence of languages in NEXP that do not have polynomial-size circuits. For example, they showed that if Promise-BPP can be simulated in nondeterministic subexponential time (even at infinitely many lengths), then the circuit lower bounds against NEXP follows. We now know several results of the form that "derandomization implies circuit lower bounds". Kabanets and Imagliazzo [KI04] showed that a nondeterministic subexponential simulation of BPP implies certain arithmetic circuit lower bounds. This result has been improved by Kinne, van Melkebeek, and Shaltiel who showed that even a "typically correct" nondeterministic simulation of BPP also implies circuit lower bounds [KvMS11]. Similar results are also known for derandomization of other probabilistic complexity classes such as AM and MA [KvM02, MV05].

There are few exceptions where we obtain derandomization without the circuit lower bounds. From the result of Karakostas, Lipton, and Viglas [KLV03], we can obtain an interesting connection between finding an efficient algorithm for the intersection of automata and derandomization. Specifically, they showed the following: Suppose L is a language consisting of the tuples $\langle F_1, F_2, \ldots, F_k \rangle$ such that each of the k automata F_k has size s and the intersection of these automata is empty. Suppose there is a deterministic algorithm running in time $s^{(k/f(k))+d}$ for an unbounded function $f(\cdot)$ dependent only on k and $d > 0$ that decides L. Then for any $\epsilon > 0$, NTIME($t(n)$) (hence RTIME($t(n)$)) is in DTIME($2^{\epsilon t(n)}$). Note that L can be trivially decided in time $O(s^k)$. We also know some unconditional derandomizations of time-bounded probabilistic classes. For example, Santhanam and Melkebeek [SvM05] showed that any RTIME($t(n)$) machine can be derandomized to a deterministic machine running in time $o(2^{t(n)})$. Recently, Williams [Wil13] proved that there is a constant $\delta > 0$ such that any NTIME($t(n)$) machine can be simulated by a deterministic machine running in time $2^{(1-\delta)t(n)}$.

However, all of the above derandomizations are very weak in some sense.

Our Contributions

Our first result concerns the derandomization of low-degree polynomial identity testing, a problem known to be in probabilistic class co-RP. Low-degree polynomial identity testing is one of the central computational problems studied in time-bounded derandomization. Clearly,

derandomizing RP will derandomize polynomial identity testing, but it requires circuit lower bounds. Our main contribution in this chapter is to show that it suffices to derandomize 2-*way*RL, a space-bounded subclass of RP, to derandomize low-degree polynomial identity testing. Now, it is known that the existence of functions that are hard against branching programs can derandomize two-way space bounded machines. Klivans and Melkebeek [KvM02] showed that if the linear space does not have $2^{\epsilon n}$-size branching programs, then RL can be derandomized to logspace. Kouckýý observed that this hardness hypothesis in fact derandomizes 2-*way*RL to logspace [Kou03]. Thus, a branching program lower bound suffices for derandomizing low-degree polynomial identity testing. Note that the randomized circuit complexity class (uniform) RNC^1 is in 2-*way*RL, and derandomization of 2-*way*RL is at least as hard as derandomizing (uniform) RNC^1.

With the above result at hand, we ask if we can derandomize a probabilistic class, like BPP or RP rather than a specific problem in them, under the branching program lower bounds. As mentioned before, most of the known results in derandomization follows from circuit lower bounds, except some weak derandomizations. With a reasonable branching program lower bound, we prove the following: If E cannot be computed by a branching program of size $2^{\alpha n}$ for some $\alpha > 0$, then BPTIME($t(n)$) is in deterministic time $2^{t(n)/\log t(n)}$. Note that the same hypothesis was used to derandomize the above polynomial identity testing problem. So a natural step forward is to weaken the hypothesis and ask for more. What happens if there is an $\epsilon > 0$ such that E does not have only 2^{n^ϵ}-size branching programs? We show that under a believable hypothesis, the entire two-sided bounded error probabilistic class BPP collapses to the average complexity class Average-DTIME(2^{n^δ}) infinitely often, under a believable hypothesis. We hereby mention that probabilistic space-bounded computations with two-way access to the random tape plays a crucial role in all of our proofs in this chapter.

Organization of this Chapter

Rest of the chapter is organized as follows. Section 6.1 proves the derandomization of low-degree polynomial identity testing problem under a branching program lower bound. The same lower bound is used to prove a derandomization of the probabilistic class BPTIME($t(n)$) in

Section 6.2. The next section shows a derandomization of the class BPP. Finally, we conclude this chapter with a possible future direction of this work.

6.1 Derandomization of Polynomial Identity Testing

The goal of this section is to prove that low-degree polynomial identity testing problem can be derandomized under branching program lower bounds. More formally,

Theorem 6.1. *If there exists an $\epsilon > 0$ such that* E *does not have $2^{\epsilon n}$-size branching programs, then low-degree polynomial identity testing over $\mathbb{Q}$ can be solved deterministically in time $n^{O(\log n)}$.*

While it is known that existence of languages in E that are hard for general Boolean circuits implies a derandomization of low-degree polynomial identity testing, the above result states that hardness against branching programs suffice to derandomize polynomial identity testing.

The heart of the proof of the above theorem is the following derandomization result. Let 2-*way*RL denote the class of languages accepted by one-sided, bounded-error probabilistic-space bounded machines that are allowed two-way access to their random tape. An interesting question is whether we can derandomize 2-*way*RL to P. Clearly, a generic approach towards this question is to design time-efficient pseudorandom generators. We show that such "black-box" derandomization will imply that low-degree polynomial identity testing can be solved in $O(n^{\log n})$ time.

Theorem 6.2. *If* 2-*way*RL *has a polynomial-time, black-box derandomization, then low-degree polynomial identity testing over $\mathbb{Q}$ can be solved deterministically in time $O(n^{\log n})$.*

Assuming that the above theorem holds, let us first prove the main theorem of this section.

Proof of Theorem 6.1. Klivans and Melkebeek [KvM02] showed that if there exists an $\epsilon > 0$ such that DSPACE(n) does not have $2^{\epsilon n}$-size branching programs, then there is a pseudorandom generator that derandomizes RL to deterministic logspace. As observed by Kouckýý [Kou03], this hypothesis is strong enough to yield efficient pseudorandom generators (and hence hitting sets) for 2-*way*RL machines. From their work it follows that the existence of a language in E with

high branching program complexity implies that 2-*way*RL has polynomial-time computable hitting sets. □

Rest of this section is focused to prove Theorem 6.2. The proof of this theorem uses a combination of existing techniques. First, we give the proof idea. It is known that polynomial identity testing (PIT) for general (any depth) arithmetic circuits over $\mathbb{Z}$ is in co-RP. We observe that when the circuit is restricted to have constant-depth, PIT over $\mathbb{Z}$ can be solved in co-2-*way*RL. Then if 2-*way*RL has polynomial-time computable hitting sets, it follows that there is a polynomial-time computable hitting set for *constant-depth* PIT over $\mathbb{Z}$. Next we observe that, such hitting set generators exist even for arithmetic circuits over the field $\mathbb{Q}$. We then use a result of Agrawal and Vinay [AV08], who showed that the existence of polynomial-time computable hitting sets for low-degree, bounded-depth, arithmetic circuits (over $\mathbb{Q}$) implies that low-degree, polynomial identity testing (over $\mathbb{Q}$) can be solved in time $n^{O(\log n)}$.

We now proceed to give details.

Proof of Main Theorem

The first step is to show that bounded-depth PIT over $\mathbb{Z}$ is in co-2-*way*RL. The proof uses the standard technique of Chinese remaindering.

Lemma 6.1. *For any constant $k > 0$, depth-k PIT over $\mathbb{Z}$ is in* co-2-*way*RL.

Proof. Let C_n be an arithmetic circuit of depth k. By our notation the size of C_n is n, has low-degree, and it represents an n-variate polynomial. Let $p(x_1, \cdots, x_n)$ be the polynomial represented by C_n. Let S be a finite set of integers $\{1, 2, \ldots, 2n\}$. By Schwartz-Zippel lemma, if p is a non-zero polynomial, then

$$\Pr_{\langle r_1, \cdots, r_n \rangle \in S^n} [p(r_1, \cdots, r_n) = 0] \leq 1/2.$$

Given a $n \log 2n$ bit string x, let $tuple(x) = \langle x_1, \cdots, x_n \rangle$ where each x_i is a $\log 2n$ bit string. Thus each x_i is in the range $[1, 2n]$. Consider a 2-*way*RL machine that behaves as follows on input C_n: Let r be first $n \log 2n$ bits that appear on its random tape. The machine checks if $C_n(r_1, \cdots, r_n)$ equals zero, where $tuple(r) = \langle r_1, \cdots, r_n \rangle$. The machine accepts C_n

if $C_n(r_1, \cdots, r_n)$ equals zero, otherwise it rejects. If the the polynomial represented by C_n is identically zero, then the machine always accepts, otherwise by Schwartz-Zippel lemma, the machine accepts with probability at most $1/2$. This establishes the correctness of the algorithm.

Now we claim that the machine can check whether $C(r_1, \cdots, r_n)$ equals zero or not in $O(\log n)$ space, given a two-way access to the random string r. In general, we cannot hope to compute $C(r_1, \cdots, r_n)$ in logspace as the output of a gate could be exponentially large (in n) and may need more than $O(\log n)$ bits to store such a value. We get around this problem by appealing to the standard Chinese remaindering trick. Since the degree and the size of the arithmetic circuit C_n are bounded by n, the output of the circuit is bounded by $n(2n)^n$ which is less than 2^{n^2}.

Let $p_1, \cdots, p_{n^2}$ be the first n^2 primes. By Chinese remainder theorem we have that $C_n(r_1, \cdots, r_n)$ equals to zero if and only if $C_n(r_1, \cdots, r_n) \equiv 0 \pmod{p_i}$ for $1 \leq i \leq n^2$. Our 2-*way*RL machine behaves as follows: For each prime p_i, $1 \leq i \leq n^2$, it evaluates $C_n(r_1, \cdots, r_n) \bmod p_i$. The machine accepts if $C_n(r_1, \cdots, r_n) \bmod p_i$ equals zero for every i, $1 \leq i \leq n^2$. Otherwise the machine rejects.

Consider a prime p_i. We claim that $C_n(r_1, \cdots, r_n) \bmod p_i$ can be evaluated in $O(\log n)$ space. Consider a gate g_u at level ℓ, and let $g_{u_1}, \cdots, g_{u_m}$ are the input gates to it. By induction hypothesis, assume that output value of each gate g_{u_j} modulo p_i, denoted by v_{g_j}, $(1 \leq j \leq m)$ can be computed in $O(\log n)$ space. Let g_u be a multiplication gate. We compute the output of g_u modulo p_i by the following algorithm $\mathcal{A}$:

Procedure $\mathcal{A}$

1: Set $x = 1$.

2: for $j = 1$ to m:

   (a) $x = x \times v_{g_j} \bmod p_i$

3: $v_{g_u} = x$.

By prime number theorem, the value of the prime p_{n^2} is at most n^3, and thus each p_i an $O(\log n)$ bit prime. Thus all arithmetic operations over $\mathbb{Z}_{p_i}$ can be performed in $O(\log n)$ space.

By our induction hypothesis v_{g_j} can be computed in $O(\log n)$ space. Thus the space needed to compute $C(r_1, \cdots, r_n)$ mod p_i by the above algorithm is given by the following recurrence relation (expressed in terms of depth):

$$S(k) = S(k-1) + O(\log n).$$

Since k is a constant the total space used is $O(\log n)$. Note that two-way access to the random tape is crucial for this algorithm. □

Based on the above lemma, we claim that existence of polynomial-time computable hitting sets for 2-*way*RL implies the existence of polynomial-time computable hitting set for constant depth, PIT.

Lemma 6.2. *Suppose that* 2-*way*RL *has polynomial-time computable hitting sets. Then, for every $k > 0$, there exists a a polynomial-time computable hitting set for depth-k, PIT over $\mathbb{Z}$.*

Proof. Fix k and consider the 2-*way*RL machine M that decides depth-k, PIT over $\mathbb{Z}$ given by Lemma 6.1. Let C_n be a depth-k, arithmetic circuit input to M. Observe that the machine M when given C_n as input uses $n \log 2n$ bits of randomness, treats it as a random n-tuple of integers $\langle r_1, r_2, \cdots, r_n \rangle$ in the range 1 to $2n$, and accepts if and only if $C_n(r_1, r_2, \cdots, r_n) = 0$. By our hypothesis, 2-*way*RL has polynomial-time computable hitting sets. Thus there exists a function $f_{n \log 2n}$ from $c \log(n \log 2n)$-bit strings to $n \log 2n$-bit strings, for some absolute constant c, such that

$$C_n \text{ is identically } 0 \Leftrightarrow \forall s; M(C_n, f_{n \log 2n}(s) = 0$$

Consider the following set

$$H = \{tuple(f_{n \log 2n}(s)) \mid s \in \Sigma^{c \log(n \log 2n)}\},$$

where an element of H is an n-tuple $\langle r_1, \cdots, r_n \rangle$ (each r_i is interpreted as an integer in the integer 1 to $2n$). Clearly, H is computable in polynomial in n. Moreover, $M(C_n, f_{n \log 2n})(s) = 0$ if and only if $C_n(r_1, \cdots, r_n) = 0$. Thus H is a hitting set for depth-k PIT. □

For our application, we require the existence of a hitting set for bounded-depth arithmetic circuits over the field $\mathbb{Q}$. The following fact can be proven.

Proposition 6.1. *For every $s(n)$-size, depth-k, arithmetic circuit C over $\mathbb{Q}$, there exists a $O(s(n))$-size, depth-$2k$ arithmetic circuit D over $\mathbb{Z}$ such that for every tuple $(a_1, \ldots, a_n) \in \mathbb{Z}^n$, $C(a_1, \cdots, a_n) \neq 0$ if and only if $D(a_1, \cdots, a_n) \neq 0$.*

Combining the above proposition with Lemma 6.2 gives us a hitting set generator for constant-depth PIT over the field $\mathbb{Q}$, under the assumption that 2-*way*RL has polynomial-time computable hitting sets.

Lemma 6.3. *Suppose that* 2-*way*RL *has polynomial-time computable hitting sets. Then for every $k > 0$, there is a polynomial-time computable hitting set for depth-k, PIT over $\mathbb{Q}$.*

The rest of the proof is based on the work of Agrawal and Vinay [AV08]. They define the notion of an optimal pseudorandom generator for arithmetic circuits and show that a hitting set for PIT implies optimal pseudorandom generator for arithmetic circuits. They then show that if there is an optimal pseudorandom generator for *depth four* arithmetic circuits over a field F, then low degree PIT over F can be solved in time $n^{O(\log n)}$. The rest of this section makes these statements formal.

Definition 6.1. [AV08] Let k be a constant. A function $f : \mathbb{N} \to (\mathbb{Q}[y])^*$ is an *optimal pseudorandom generator* against depth-k, arithmetic circuits (over $\mathbb{Q}$) if

(a) $f(n) = (p_1^n(y), \cdots, p^n(y))$ is computable in polynomial-time, each $p_i^n(y)$ is a univariate polynomial over $\mathbb{Q}$ and the degree of each polynomial is bounded by a polynomial (in n).

(b) For any depth-k, arithmetic circuit C of size n that is computing a multi-variate polynomial over $\mathbb{Q}$, $C(x_1, x_2, \cdots, x_n) = 0$ if and only if $C(p_1^n(y), \cdots, p_n^n(y)) = 0$.

Theorem 6.3. [AV08]

(a) If there exists a polynomial-time computable hitting set for depth-k, PIT over $\mathbb{Q}$, then there exists an optimal pseudorandom generator against depth-k, arithmetic circuits.

(b) If there exist optimal pseudorandom generators against depth four arithmetic circuits over $\mathbb{Q}$, then low-degree, PIT over $\mathbb{Q}$ can solved in time $n^{O(\log n)}$.

Combining Lemma 6.3 with Theorem 6.3, we obtain that if 2-*way*RL has a polynomial-time computable hitting sets, then low-degree PIT over $\mathbb{Q}$ can be solved in time $O(n^{\log n})$. This completes the proof of Theorem 6.2.

6.1.1 Connections to Circuit Lower Bounds

Kabanets and Impagliazzo [KI04] showed that if polynomial identity testing over integers can be solved in deterministic time $O(n^{\log n})$, then either Permanent (of integer matrices) does not have polynomial-size arithmetic circuits over $\mathbb{Z}$, or NEXP cannot be computed by Boolean circuits of size $s(n)$. Since a polynomial-time, black-box derandomization of 2-*way*RL implies derandomization of low-degree PIT, it follows that our hypothesis also has the same consequences. We now state this formally.

We use the following result (stated with our choice of parameters) due to Aaronson and van Melkebeek [AvM11] which is a quantitative improvement over the result of Kabanets and Impagliazzo. Let SIZE($s(n)$) denote the class of languages accepted by circuits of size $s(n)$. Dvir, Shpilka, and Yehudayoff [DSY09] observed that a dichotomy similar to [KI04] holds even for the bounded-depth arithmetic circuits.

Theorem 6.4. *Suppose that low-degree, polynomial identity testing is in* NTIME($n^{\log n}$). *Then one of the following must be true:*

(a) NEXP $\cap$ co-NEXP $\not\subseteq$ SIZE($o(2^n/n)$).

(b) Permanent does not have low-degree arithmetic circuits of size $O(n^{\log n})$.

Combining above with Theorem 6.1, we obtain a similar connection among derandomization of 2-*way*RL, the Boolean circuit complexity of NEXP $\cap$ co-NEXP, and the arithmetic circuit complexity of Permanent (of a matrix).

Corollary 6.1. *If* 2-*way*RL *has polynomial-time, black-box derandomization, then one of the following must be true:*

(a) NEXP $\cap$ co-NEXP *cannot be computed by Boolean circuits of size* $o(2^n/n)$.

(b) Permanent over $\mathbb{Z}$ *does not have low-degree arithmetic circuits of size* $O(n^{\log n})$ *over* $\mathbb{Q}$.

Note that 2-*way*RL lies between uniform-RNC_1 and uniform-RNC_2.Thus derandomizing 2-*way*RL is at least as hard as derandomizing uniform-RNC_1 and perhaps easier than derandomizing uniform-RNC_2. Kabanets and Impagliazzo showed that derandomizing RNC also implies circuit lower bounds. The following result, attributed to Kabanets, is an improvement over the original result of [KI04]. For a proof see cstheory stack exchange[1].

Theorem 6.5. *If uniform-*RNC *is in* $\mathrm{NTIME}(n^{\log^k n})$ *(for some constant* $k > 0$*), then either* NEXP *is not computable by Boolean circuits of size* $o(2^n/n)$ *or Permanent does not have (division-free) arithmetic formulas over* $\mathbb{Z}$ *of size* $O(n^{\log n})$.

The above theorem concerns with the derandomization of (uniform) RNC, whereas Corollary 6.1 concerns with derandomization of 2-*way*RL which is a subclass of RNC. However, the derandomization assumed in Corollary 6.1 is much stronger than the derandomization assumed in the above theorem.

Agrawal [Agr05] showed that if there exist optimal pseudorandom generators against bounded-depth arithmetic circuits, then there exist a class of polynomials computable in EXP that requires bounded-depth, exponential-size, arithmetic circuits. Thus, as shown by Agrawal and Vinay [AV08], existence of optimal pseudorandom generators against bounded-depth arithmetic circuits implies that exponential time requires exponential-size arithmetic circuits. Thus we obtain the following interesting corollary that connects branching program and arithmetic circuits complexities of exponential time classes.

Corollary 6.2. *If there exist a language in* E *that does not have* $2^{\epsilon n}$*-size branching programs, then there exist class of multilinear polynomials* $\{p_n\}_{n\geq 0}$ *over* $\mathbb{Q}$ *that are (uniformly) computable in* EXP*, and require arithmetic circuits of size* $2^{\delta n}$*-size for some* $\delta > 0$.

We conclude this section by observing that derandomization of (promise) 2-*way*RL also implies certain circuit lower bounds for NEXP. Consider the following promise problem:

Yes Instances: All NC^1 circuits that accept at least 1/2 fraction of their inputs.

No Instances: All NC^1 circuits that do not accept any input.

[1] `http://cstheory.stackexchange.com/questions/18444/is-uniform-rnc-contained-in-polylog-space`

Since any circuit has two-way access to its input bits and NC^1 circuits (over n inputs) can be evaluated in $O(\log n)$ space, the above problem is in Promise-2-*way*RL. Williams [Wil11, Wil13] showed that any subexponential algorithm for the above promise problem implies that NEXP is not in NC^1. In our framework, his result can be restated as:

Theorem 6.6. *If* Promise-2-*way*RL *is in subexponential time, then* NEXP *is not in* NC^1.

6.2 Derandomization of BPTIME(t(n))

Now we move on to prove the derandomization results for the complexity classes, rather than a specific problem. The goal of this section is to prove the following derandomization of the bounded-error probabilistic time class under a branching program lower bound.

Theorem 6.7. *If* E *cannot be computed by a branching program of size* $2^{\alpha n}$ *for some* $\alpha > 0$*, then for any time constructible function* $t(n) \geq n$*,*

$$\mathrm{BPTIME}(t(n)) \subseteq \mathrm{DTIME}(2^{t(n)/\log t(n)}).$$

The proof of the above theorem consists of two parts. First, we show that "derandomization of *two-way*, probabilistic space bounded computations yield derandomization of probabilistic time". Next, we use a known result by Klivans van Melkebeek [KvM02] which essentially states that lower bounds against branching programs can be used to derandomize probabilistic space-bounded computations even with two-way access to random tape. Now we give the details.

The first step is to see the following easy corollary of the celebrated result of Hopcroft, Paul, and Valiant [HPV77] for the probabilistic classes.

Lemma 6.4. *For any time constructible function* $t(n) \geq n$*,*

$$\mathrm{BPTIME}(t(n)) \subseteq 2\text{-}way\mathrm{BPSPACE}(t(n)/\log t(n)).$$

Proof. Consider a language $L \in \mathrm{BPTIME}(t(n))$ and suppose the probabilistic machine M decides it. Our simulating machine N uses the offline model of computation and thus stores the random string on a special two-way random tape. The random string r is of length at most $t(|x|)$ on input x. On input (x, r), the machine N can simulate M in space $O(t(n)/\log t(n))$, by the results of [HPV77], given two-way access to the input. □

Klivans and van Melkebeek [KvM02] showed that if DSPACE(n) cannot be computed by a branching program of size $2^{\epsilon n}$ for some $\epsilon > 0$, then 2-*way*BPSPACE($\log n$) = L. We can follow the same proof to show the following lemma.

Lemma 6.5. *If* E *cannot be computed by a branching program of size* $2^{\epsilon n}$ *for some* $\epsilon > 0$*, then* 2-*way*BPSPACE($\log n$) $\subseteq$ P.

A simple padding technique completes the proof of Theorem 6.7.

Lemma 6.6. *If* E *cannot be computed by a branching program of size* $2^{\epsilon n}$ *for some* $\epsilon > 0$*, then for any time constructible function* $t(n) \geq n$*,*

$$2\text{-}way\text{BPSPACE}(t(n)/\log t(n)) \subseteq \text{DTIME}(2^{t(n)/\log t(n)}).$$

Proof. By Lemma 6.5, we can assume 2-*way*BPSPACE($\log n$) $\subseteq$ P under the hypothesis. Now suppose a language $L \in$ 2-*way*BPSPACE($t(n)/\log t(n)$) and BPSPACE machine M decides L in space $t(n)/\log t(n)$. Define a new language

$$L_{pad} = \{\langle x, 0^{2^{t(|x|)/\log t(|x|)} - |x|}\rangle : x \in L\}.$$

Then we claim that $L_{pad} \in$ 2-*way*BPSPACE($\log n$) and the machine M' that decides it is the following: on input y, verify if there is a z such that $y = \langle z, 0^{2^{t(|z|)/\log t(|z|)} - |z|}\rangle$ and reject otherwise. Run $M(z)$ and accept if and only if $M(z) = 1$. Suppose, $|z| = n$, implying $|y| = 2^{t(n)/\log t(n)} = n'$ (say). Clearly, M' takes $t(n)/\log t(n) = \log n'$ space. Then by the hypothesis, $L_{pad} \in \text{DTIME}(n^c)$ on input length n and some constant $c > 0$. Suppose N is the DTIME($\cdot$) machine that decides L_{pad}. Now we show that $L \in \text{DTIME}(2^{ct(n)/\log t(n)})$. We define a machine M'' on input x as follows: pad with $0^{2^{t(|x|)/\log t(|x|)} - |x|}$ and suppose the new string is y. Then run $N(y)$ and accept if and only if $N(y) = 1$. The machine M'' runs for time $(2^{t(n)/\log t(n)})^c$. □

6.3 Derandomization of BPP

Furst, Lipton, and Stockmeyer [FLS85] showed that any non-trivial simulation of deterministic time by deterministic space will automatically imply a non-trivial simulation of probabilistic time by deterministic space. More specifically, they showed that under the assumption that

polynomially-expanding pseudorandom generator exists, for every $\epsilon > 0$, if P is in DSPACE(n^ϵ) then for every $\delta > 0$, RP is in DSPACE(n^δ).

We make a similar assumption in this section.

Hypothesis A. If there exists a constant $c > 0$ such that P $\subseteq$ DSPACE(n^c), then there exists a constant d such that BPP $\subseteq$ 2-*way*BPSPACE(n^d).

Given the above hypothesis, our goal is to prove the following theorem.

Theorem 6.8. *Suppose that the Hypothesis A holds. If there is an $\epsilon > 0$ such that* E *does not have 2^{n^ϵ}-size branching programs, then for every $\delta > 0$*

$$\text{BPP} \subseteq_{\text{io}} Average - \text{DTIME}(2^{n^\delta}).$$

Proof. If EXP $\not\subseteq$ P/poly, then by the well-known result of Babai, Fortnow, Nisan, and Wigderson [BFNW93], we have BPP $\subseteq_{\text{io}}$ DTIME(2^{n^δ}), for every $\delta > 0$. So the conclusion follows.

Now suppose EXP $\subseteq$ P/poly, then there exists an absolute constant $D > 0$ such that P has n^D-size circuits. The tableau method [NW91] shows that P has fixed polynomial-size circuits. Let L be a language in P and let M be a machine that decides L. Then consider the following tableau function:

$$f(x, i, j) = \text{Contents of cell } j \text{ at time } i \text{ when } M(x).$$

Since L is in P, f is a function in P. Thus f has n^C-size circuits for a fixed constant C.

Consider the following algorithm $\mathcal{A}$ to decide L.

Procedure $\mathcal{A}$

1: Input x, $|x| = n$.

2: Cycle through all circuits of size n^C. For each such circuit c,

(a) Check if it is a correct circuit for L, by performing consistency check.

(b) If c is a correct circuit, then accept x if and only if $c(x, t(n), t(n))$ contains accept state.

Number of circuits of size n^C is at most $2^{n^{2C}}$. Given a circuit, consistency check can be performed in space $O(\log n)$. Thus the total space taken by the above algorithm is bounded by n^{3C}. So P is in DSPACE(n^{3C}).

By our Hypothesis A, we have that BPP $\subseteq$ 2-*way*BPSPACE(n^d) for a fixed constant $d > 0$. Further, if there is an $\epsilon > 0$ such that E does not have branching programs of size 2^{n^ϵ}, then for any $d > 0$, 2-*way*BPSPACE(n^d) is in DTIME(2^{n^e}) for a fixed e [BFNW93, Nis93]. This implies that EXP $\neq$ BPP. Impagliazzo and Wigderson [IW98] showed that if EXP differs from BPP, then for every $\delta > 0$, for every language L in BPP, there exists a 2^{n^δ}-time bounded machine M such that for infinitely many input lengths $n > 0$, M correctly decides L on more than $1 - \frac{1}{n}$ fraction of inputs. Thus the theorem follows. □

6.4 Conclusions

In this chapter we studied the derandomization under branching program lower bounds, rather than the standard circuit lower bounds. We showed that the well-known low-degree polynomial identity testing can be derandomized under believable branching program lower bounds, which was previously unknown. We further showed that the standard time-bounded probabilistic complexity classes can also be derandomized, albeit weakly, under the hypotheses involving branching program lower bounds. An obvious question at this point is if we can prove the Hypothesis A. It will be interesting to see if we can obtain better unconditional derandomization of BPTIME than Santhanam and Melkebeek [SvM05] with any of the ideas mentioned in this chapter.

BIBLIOGRAPHY

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

[Adl78] L.M. Adleman. Two theorems on random polynomial time. In *Proc. 19th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 75–83, 1978.

[Agr02] M. Agrawal. Pseudo-random generators and structure of complete degrees. In *Proc. 17th IEEE Conference on Computational Complexity*, pages 139–147, 2002.

[Agr05] M. Agrawal. Proving Lower Bounds Via Pseudo-random generators. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, pages 92–105, 2005.

[AKS87] M. Ajtai, J. Komlós, and E. Szemerédi. Deternministic Simulation in LOGSPACE. In *Proc. 19th ACM Symposium on Theory of Computing (STOC)*, volume 2, pages 132–140, 1987.

[AM77] L. Adleman and K. Manders. Reducibility, randomness, and intractability. In *Proc. 9th ACM Symp. Theory of Computing*, pages 151–163, 1977.

[ASB00] K. Ambos-Spies and L. Bentzien. Separating NP-Completeness Notions under Strong Hypotheses. *J. Comp. System Sci.*, 361(2):335–361, 2000.

[ASTW01] S. Aida, R. Schuler, T. Tsukiji, and O. Watanabe. On the difference between polynomial-time many-one and truth-table reducibilities on distributional problems. In *Proc. 18th International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2001.

[AV08] M. Agrawal and V. Vinay. Arithmetic Circuits: A Chasm at Depth Four. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 67–75. Ieee, October 2008.

[AvM11] S. Aaronson and D. van Melkebeek. On circuit lower bounds from derandomization. *Theory Of Computing*, 7:177–184, 2011.

[AW09] M. Agrawal and O. Watanabe. One-Way Functions and the Berman-Hartmanis Conjecture. In *Proc. 24th IEEE Conference on Computational Complexity*, pages 194–202, 2009.

[Bar02] B. Barak. A Probabilistic-Time Hierarchy Theorem for "Slightly Non-uniform" Algorithms. In *Proc. RANDOM*, pages 194–208. Springer, 2002.

[BCP83] A. Borodin, S. Cook, and N. Pippenger. Parallel Computation for Well-Endowed Rings and Space-Bounded Probabilistic Machines. *Information and Control*, 58(1-3):113–136, 1983.

[BDG90] J. Balcázar, J. Diaz, and J. Gabarró. *Structural Complexity II.* Springer-Verlag New York, Inc., 1990.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity*, 1991.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has Subexponential Time Simulations unless EXPTIME has Publishable Proofs. *Computational Complexity*, 3(4):307–318, 1993.

[BFT98] H. Buhrman, L. Fortnow, and T. Thierauf. Nonrelativizing Separations. In *Proc. 13th Annual IEEE Conference on Computational Complexity*, pages 8–12, 1998.

[BHT91] H. Buhrman, S. Homer, and L. Torenvliet. Completeness notions for nondeterministic complexity classes. *Mathematical Systems Theory*, 24:179–200, 1991.

[BL99] L. Babai and S. Laplante. Stronger separations ofor random-self-reducibility, rounds, and advice. In *Proc. 14th IEEE Conference on Computational Complexity*, pages 98–104, 1999.

[BT94] H. Buhrman and L. Torenvliet. On the structure of complete sets. In *Proc. 9th IEEE Annual Conference on Structure in Complexity Theory*, pages 118–133, 1994.

[BYGW99] Z. Bar-Yossef, O. Goldreich, and A. Wigderson. Deterministic Amplification of Space-Bounded Probabilistic Algorithms. In *Proc. 14th Annual IEEE Conference on Computational Complexity, Atlanta, Georgia, USA, May 4-6*, 1999.

[CDL01] A. Chiu, G. Davida, and B. Litow. Division in Logspace-uniform NC^1. *Theoretical Informatics and Applications*, 35:259–275, 2001.

[Coo71] S. Cook. The Complexity of Theorem-proving Procedures. In *Proc. 3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[CRV11] K.M. Chung, O. Reingold, and S. Vadhan. S-T connectivity on digraphs with a known stationary distribution. *ACM Transactions on Algorithms*, 7(3):30, 2011.

[CW89] A. Cohen and A. Wigderson. Dispersers, Deteministic Amplification, and Weak Random Sources. In *Proc. 30th IEEE Symposium on Foundations of Computer Science*, pages 1–37, 1989.

[DNPS11] M. David, P. Nguyen, P.A. Papakonstantinou, and A. Sidiropoulos. Computationally Limited Randomness. In *Proc. Innovations in Theoretical Computer Science (ITCS)*, 2011.

[DPS11] M. David, P.A. Papakonstantinou, and A. Sidiropoulos. How strong is Nisan's pseudo-random generator? *Information Processing Letters*, 111(16):804–808, 2011.

[DSY09] Z. Dvir, A. Shpilka, and A. Yehudayoff. Hardness-randomness tradeoffs for bounded depth arithmetic circuits. *SIAM Journal on Computing*, 39(4):1279–1293, 2009.

[ESY84] S. Even, A.L. Selman, and Y. Yacobi. The Complexity Of Promise Problems With Applications To Public-Key Cryptography. *Information and Control*, 61(2):159–173, May 1984.

[FFLN96] J. Feigenbaum, L. Fortnow, S. Laplante, and A. Naik. On Coherence, Random-self-reducibility, and Self-correction. In *Proc. 11th Annual IEEE Conference on Computational Complexity*, pages 224–232, 1996.

[FFLS92] J. Feigenbaum, L. Fortnow, C. Lund, and D. Spielman. The Power of Adaptiveness and Additional Queries in Random-Self-Reductions. In *Proc. 7th Annual Conference on Structure in Complexity Theory*, pages 338–346, 1992.

[FK06] L. Fortnow and A.R. Klivans. Linear advice for randomized logarithmic space. In *Proc. Symposium on Theoretical Aspects of Computer Science (STACS)*, 2006.

[FLS85] M. Furst, R.J. Lipton, and L. Stockmeyer. Pseudorandom number generation and space complexity. *Information and Computation*, 64:43–51, January 1985.

[FS04] L. Fortnow and R. Santhanam. Hierarchy Theorems for Probabilistic Polynomial Time. In *Proc. 45th IEEE Symposium on Foundations of Computer Science*, page 324. IEEE Computer Society, 2004.

[FS06] L. Fortnow and R. Santhanam. Guest column: Recent work on hierarchies for semantic classes. *ACM SIGACT News*, 37(3):36–54, 2006.

[FS07] L. Fortnow and R. Santhanam. Time Hierarchies : A Survey. *ECCC*, 4(4), 2007.

[FST05] L. Fortnow, R. Santhanam, and L. Trevisan. Promise Hierarchies. *Electronic Colloquium on Computational Complexity*, pages 1–11, 2005.

[GG81] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981.

[GHP11] X. Gu, J.M. Hitchcock, and A. Pavan. Collapsing and Separating Completeness Notions Under Average-Case and Worst-Case Hypotheses. *Theory of Computing Systems*, 51(2):248–265, October 2011.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proc. 21st ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[Gol08] O. Goldreich. *Computational Complexity - A Conceptual Perspective*. Cambridge University Press, 2008.

[GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–355, April 1988.

[GSSZ04a] C. Glaßer, A. Selman, S. Sengupta, and L. Zhang. Disjoint NP-pairs. *SIAM J. Comput.*, 33(6):1369–1416, 2004.

[GSSZ04b] C. Glaßer, A.L. Selman, S. Sengupta, and L. Zhang. Disjoint NP-pairs. *SIAM Journal on Computing*, 33(6):1369–1416, 2004.

[GSZ07] C. Glaßer, A. Selman, and L. Zhang. Canonical disjoint NP-pairs of propositional proof systems. *Theoretical Computer Science*, 370(1):60–73, 2007.

[GV04] D. Gutfreund and E. Viola. Fooling parity tests with parity gates. In *Proc. APPROX and RANDOM*, pages 381–392. Springer-Verlag, 2004.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HKR93] S. Homer, S. Kurtz, and J. Royer. On 1-truth-table-hard languages. *Theoretical Computer Science*, 115(2):383–389, 1993.

[HNOS96] L.A. Hemaspaandra, A. Naik, M. Ogihara, and A.L. Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM J. Comput.*, 25(4):697–708, 1996.

[Hom97] S. Homer. Structural properties of complete problems for exponential time. In L Hemaspaandra and A Selman, editors, *Complexity Theory Retrospective II*, pages 135–153. Springer-Verlag, 1997.

[HP08] J.M. Hitchcock and A. Pavan. Hardness Hypotheses, Derandomization, and Circuit Complexity. *Computational Complexity*, 17(1):119–146, April 2008.

[HPRS12] A. Hughes, A. Pavan, N. Russell, and A.L. Selman. A Thirty Year Old Conjecture about Promise Problems. In *39th International Colloquium on Automata, Languages, and Programming*, pages 473–484, 2012.

[HPV77] J.H. Hopcroft, W.J. Paul, and L.G. Valiant. On Time Versus Space. *Journal of the ACM*, 24(2):332–337, April 1977.

[HPV08] J.M. Hitchcock, A. Pavan, and N.V. Vinodchandran. Partial Bi-immunity, Scaled Dimension, and NP-Completeness. *Theory of Computing Systems*, 42(2):131–142, July 2008.

[IKW02] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002.

[INW94] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for Network Algorithms. In *Proc. 26th ACM Symposium on Theory of Computing (STOC)*, pages 356–364, 1994.

[ISW06] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Reducing The Seed Length In The Nisan-Wigderson Generator. *Combinatorica*, 26(6):647–681, December 2006.

[IW97] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proc. 29th ACM symposium on Theory of computing*, pages 220–229, 1997.

[IW98] R. Impagliazzo and A. Wigderson. Randomness vs. time: de-randomization under a uniform assumption. In *39th Annual Symposium on Foundations of Computer Science: proceedings, 1998*, pages 734–743, 1998.

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proc. 30th IEEE Symposium on Foundations of Computer Science*, volume pp, pages 248–253. IEEE Comput. Soc. Press, 1989.

[JL95] D.W. Juedes and J.H. Lutz. Weak completeness in E and E_2. *Theoretical Computer Science*, 143:149–158, 1995.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations: The IBM Research Symposia Series*, pages 85–103. Springer US, 1972.

[KI04] V. Kabanets and R. Impagliazzo. Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[KLV03] G. Karakostas, R.J. Lipton, and A. Viglas. On the complexity of intersecting finite state automata and NL versus NP. *Theoretical Computer Science*, 302:257–274, 2003.

[KM81] K. Ko and D. Moore. Completeness, Approximation And Density. *SIAM Journal on Computing*, 10(4):787–796, 1981.

[Kou03] M. Koucký. *On Traversal Sequences, Exploration Sequences and completeness of Kolmogorov Random Strings*. PhD thesis, Rutgers, The state university of New Jersy., 2003.

[KV85] M. Karpinski and R. Verbeek. There Is No Polynomial Deterministic Space Simulation of Probabilistic Space with a Two-Way Random-Tape Generator. *Information and Control*, 67(1985):158–162, 1985.

[KvM02] A.R. Klivans and D. van Melkebeek. Graph Nonisomorphism Has Subexponential Size Proofs Unless the Polynomial-Time Hierarchy Collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

[KvM09] J. Kinne and D. van Melkebeek. Space Hierarchy Results for Randomized and other Semantic Models. *Computational Complexity*, 19(3):423–475, November 2009.

[KvMS11] J. Kinne, D. van Melkebeek, and R. Shaltiel. Pseudorandom Generators, Typically-Correct Derandomization, and Circuit Lower Bounds. *Computational Complexity*, 21(1):3–61, September 2011.

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

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, October 1992.

[LLS75] R.E. Ladner, N.A. Lynch, and A.L. Selman. A Comparison of Polynomial Time Reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.

[LM96] J.H. Lutz and E. Mayordomo. Cook versus Karp-Levin: Separating completeness notions if NP is not small. *Theor. Comput. Sci.*, 164(1-2):141–163, 1996.

[LV03] R.J. Lipton and A. Viglas. Non-uniform Depth of Polynomial Time and Space Simulations. In *International Symposium on Fundamentals of Computation Theory (FCT)*, pages 311–320. Springer, 2003.

[LY90] L. Longpré and P. Young. Cook reducibility is faster than Karp reducibility. *Journal of Computer and System Sciences*, 41:389–401, 1990.

[Mar75] G.A. Margulis. Explicit Constructions of Concentrators . *Problems of Information Transmission (English Translation)*, 9(4):325–332, 1975.

[MM11] C. Moore and S. Mertens. *The Nature of Computation*. Oxford University Press, 2011.

[MR96] R. Motwani and P. Raghavan. Randomized algorithms. *ACM Computing Surveys*, 28(1):33–37, December 1996.

[MU05] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[MV05] P.B. Miltersen and N.V. Vinodchandran. Derandomizing Arthur–Merlin Games using Hitting Sets. *Computational Complexity*, 14(3):256–279, December 2005.

[Nis92] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[Nis93] N. Nisan. On read once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107(1):135–144, 1993.

[NW91] N. Nisan and A. Wigderson. Rounds in communication complexity revisited. In *Proc. 23rd annual ACM symposium on Theory of computing*, pages 419–429. ACM Press, 1991.

[NW94] N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[NZ96] N. Nisan and D. Zuckerman. Randomness is Linear in Space. *Journal of Computer and System Sciences*, 52(1):43–52, February 1996.

[Pap94] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Pav03] A. Pavan. Comparison of reductions and completeness notions. *Guest column: SIGACT News Complexity Theory Column 40*, pages 1–14, 2003.

[PS02] A. Pavan and A.L. Selman. Separation of NP-completeness notions. *SIAM Journal on Computing*, 31(3):906–918, 2002.

[PS04] A. Pavan and A.L. Selman. Bi-immunity separates strong NP-completeness notions. *Information and Computation*, 188(1):116–126, January 2004.

[Pud01] P. Pudlak. On Reducibility and Symmetry of Disjoint NP-pairs. In *Electronic Colloquium on Computational Complexity, technical reports*, 2001.

[Raz94] A. Razborov. On provably disjoint NP pairs. Technical Report 94-006, ECCC, 1994.

[Rei08] O. Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):1–24, 2008.

[RR99] R. Raz and O. Reingold. On recycling the randomness of states in space bounded computation. In *Proc. 31st ACM Symposium on Theory of Computing (STOC)*, pages 159–168, 1999.

[RTV06] O. Reingold, L. Trevisan, and S. Vadhan. Pseudorandom walks on regular digraphs and the RL vs. L problem. In *Proc. 38th ACM Symposium on Theory of Computing (STOC)*, page 457, 2006.

[Sak96] M. Saks. Randomization and Derandomization in Space-Bounded Computation. In *Proc. 11th IEEE Conference on Computational Complexity*, 1996.

[Sch60] J. Schoenfield. Degrees of models. *Journal of Symbolic Logic*, 25:233–237, 1960.

[Sel79] A.L. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Math. Systems Theory*, 13:55–65, 1979.

[SG77] I. Simon and J. Gill. Polynomial Reducibilities and Upward Diagonalizations. In *Proc. 9th Annual ACM Symposium on Theory of Computing*, pages 186–194, 1977.

[Sha92] A. Shamir. IP=PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

[SvM05] R. Santhanam and D. van Melkebeek. Holographic proofs and derandomization. *SIAM Journal on Computing*, 35(1):59–90, 2005.

[SY82] A. Selman and Y. Yacobi. The complexity of promise problems. In *Proc. 8th Colloq. on Automata, Languages, and Programming, Lecture Notes in Computer Science*, volume 140, pages 502–509, Berlin, 1982. Springer-Verlag.

[SZ99] M. Saks and S. Zhou. BPSPACE(S) $\subseteq$ DSPACE($S^{3/2}$). *Journal of Computer and System Sciences*, 403:376–403, 1999.

[TFL93] S. Tang, B. Fu, and T. Liu. Exponential Time and Subexponential Time Sets. *Theoretical Computer Science*, 115:371–381, 1993.

[Tod91a] S. Toda. On polynomial-time truth table reducibility of intractable sets to P-selective sets. *Mathematical Systems Theory*, 24:69–82, 1991.

[Tod91b] S. Toda. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[Uma03] C. Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419–440, 2003.

[VV86] L.G. Valiant and V.V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[Wat87] O. Watanabe. A comparison of polynomial time completeness notions. *Theoretical Computer Science*, 54:249–265, 1987.

[Wil11] R. Williams. Guest column: A casual tour around a circuit complexity bound. *ACM SIGACT News*, 42(3):54, 2011.

[Wil13] R. Williams. Improving Exhaustive Search Implies Superpolynomial Lower Bounds. *SIAM Journal on Computing*, 42(3):1218–1244, January 2013.